MULTIAGENT MANIPULATOR CONTROL

By

Simon Philip Monckton

B. Sc. (Mechanical Engineering) University of Alberta

M. Sc. (Mechanical Engineering) University of Alberta

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 1997

[illegible]

Abstract

The objective of this research is to define, specify, and implement a new, robust, and extensible manipulator control founded upon recent developments in multiagent robot control architectures..

Historically manipulator controllers serve within an idealized monolithic"sense-model-plan-act" (SMPA) control cycle that is both difficult and expensive to design for real time implementation. Recently, however, robotic systems have achieved remarkable performance through the combination of multiple, relatively simple, task specific controllers. These *agents* are arguably more reliable, robust, and extensible than SMPA architectures exhibiting similar performance. Furthermore, complex tasks have been achieved through *multiagent* teams, often exhibiting self organinzing or *emergent* behaviour. Despite these benefits and the growing popularity of these techniques, a formal model of agent and/or multiagent systems has not been proposed nor has any such architecture been applied to classical manipulation robotics. This thesis attempts to address these omissions through the analysis and application of multiagent design principles to manipulator control.

After an introduction to problems in real time supervisory robot control, an overview of manipulator and controller dynamics establishes a reference robot model. With this model as background, experimental high performance robot architectures are examined and concepts common to these systems identified. Multiagent manipulator control strategies are then discussed and global goal distribution mechanism introduced. The design and implementation of a complementary multiprocess manipulator simulator is then described.

With a global goal distribution definition, the design of a manipulator model-free global goal generator is discussed. Results from a multiagent manipulator control simulation are then presented and evaluated. The focus then turns to multiple global goal operation, discussing self organization, multiple global goals, and the impact of simultaneously active local and global goal systems on stability and arbitration. Results demonstrate multiple global and local goal operation including combinations of end effector trajectory tracking, joint failure, obstacle avoidance, joint centering, and joint limit avoidance. Finally, the significance of these results is discussed in the context of general multiagent control.

Table of Contents

Abstract ii

List of Tables ix

List of Figures xi

Acknowledgement xviii

Nomenclature xix

1 Introduction 1

1.1 Motivation 5

1.2 Survey of Related Research 6

1.2.1 Multiagent Control 6

1.2.2 Real-Time Manipulator Control 9

1.3 Contributions of the Thesis 9

1.4 Thesis Outline 11

2 The Manipulator Model 13

2.1 Introduction 13

2.2 The Manipulator Model 14

2.2.1 The Kinematic Model 14

2.2.2 Manipulator Inertia Matrix 17

2.2.3 Manipulator Dynamics 18

2.2.4 Properties of The Manipulator Jacobian 19

2.2.5 The Motor Model 20

3 Real Time Robot Control 24

3.1 Introduction 24

3.2 Task Functions 25

3.3 Manipulator Control and the Inverse Solution . . . 26
3.3.1 Resolved Motion Position Control . . . 26
3.3.2 Resolved Motion Rate Control . . . 28
3.3.3 Resolved Motion Acceleration Control . . . 29
3.3.4 Jacobian Transpose Control, JTC . . . 30
3.4 Real Time Supervisory Control Architectures . . . 31
3.4.1 Manipulation . . . 32
3.4.2 High Speed Motion . . . 36
3.4.3 Mobile Robot Navigation . . . 37
3.4.4 Computer Graphics . . . 41
3.4.5 Common Denominators in Real Time Robot Control . . . 42
3.5 Multiple Robot Control . . . 44
3.5.1 Common Denominators in Real Time Robot Team Control . . . 47
3.6 Remarks . . . 49

4 The Agent and Multiagent Model **50**
4.1 Introduction . . . 50
4.2 Behaviour Control . . . 50
4.2.1 Goals . . . 52
4.2.2 Behaviours . . . 53
4.2.3 Behaviour Control . . . 54
4.3 A General Model of Agency . . . 57
4.3.1 Information Exchange . . . 59
4.3.2 Arbitration . . . 61
4.3.3 Example: A Linear Arbitration Model . . . 65
4.3.4 From lone Agents to Agent Teams . . . 67
4.4 A General Model of Multiagency . . . 67
4.4.1 Local Behaviour . . . 68
4.4.2 Global Behaviour . . . 69
4.4.3 Multiagent Coordination . . . 71
4.4.4 Global Goals . . . 74

4.4.5 Multiagent Control 76

4.4.6 Emergent Multiagent Control 76

4.5 Linear Arbitration and Multiagent Control 77

4.6 Multiagent Manipulation 78

4.6.1 Links as Agents 78

4.6.2 Manipulators as Multiagent Systems 79

4.6.3 Global Goal Distribution 80

4.6.4 The Global Goal Proxy 82

4.7 A Link Agent Specification 85

4.8 Summary 87

5 The Multiprocess Manipulator Simulator **89**

5.1 Introduction 89

5.1.1 Requirements Overview 90

5.1.2 Design Specification Overview 91

5.2 Foundations 92

5.2.1 Quaternions 92

5.2.2 Cartesian State 93

5.2.3 Cartesian Trajectories and Paths 94

5.2.4 Parsers 94

5.2.5 LinkModel 95

5.2.6 Interprocess Communications 96

5.3 The Process Manager and Synthetic Clock, `SI` 97

5.3.1 Execution 97

5.4 The Dynamic Modeler, `DM` 98

5.4.1 `ManipulatorModel` 98

5.4.2 Obstacle Modelling 103

5.4.3 Physical Realizability 103

5.4.4 Execution 104

5.5 The Global Goal Process, `GP` 105

5.5.1 The `GlobalAgent` Object 105

5.5.2 Execution 107
5.6 The Link Process,`LP` 107
5.6.1 The Agent Object 107
5.6.2 The Kinematic Bus 108
5.6.3 The Global Goal `IOStream` 109
5.6.4 Controller Objects 109
5.6.5 Arbitration 110
5.6.6 Execution 111
5.7 Operational Overview 111
5.8 Data Logs and Viewing 111
5.9 Verification 113
5.9.1 A 3R Reference Model 114
5.10 Summary 116

6 Global Goal Generation 117

6.1 Introduction 117
6.2 Global Goal Generation 118
6.2.1 Resolved Motion Decentralized Adaptive Control 119
6.3 Multiagent Controller Performance 123
6.3.1 Trajectory Tracking Performance 126
6.3.2 Model Free Goal Regulation 128
6.3.3 Actuator Saturation 129
6.4 Summary 137

7 Multiple Goal Systems 140

7.1 Introduction 140
7.2 Auxiliary Behaviour and the Jacobian Null Space 141
7.3 Auxiliary Global Goal Systems 145
7.3.1 Obstacle Avoidance 145
7.3.2 Multiagent Obstacle Avoidance 149
7.3.3 Results 154
7.3.4 Discussion 162

7.4 Local and Global Goals Combined 163

7.4.1 Why Local Goals? 163

7.4.2 Continuous Nonlinear Joint Limit Repulsion 164

7.5 Linear Local Goal Design 167

7.6 Local Goal Strategies 168

7.6.1 Fail Safe Locking and Robustness 169

7.6.2 Constant Compliance Centering 172

7.6.3 Constant Compliance Retraction 179

7.6.4 Variable Compliance Centering 190

7.6.5 Combinations: Trajectory Tracking, Obstacle Avoidance, and Centering 191

7.7 Arbitration and Compliance 205

7.8 Summary 208

8 Multiagent Control Compared **211**

8.0.1 Multiagent vs. Centralized Control 211

8.0.2 Multiagent Manipulator Control and the Multiagent Context 212

8.1 Demonstrating The Advantages 214

8.1.1 Performance 214

8.1.2 Structure 215

8.2 Stability 218

8.2.1 Combining Locally and Globally Compliant Goal Systems 220

8.2.2 Combining Globally Adaptive and Locally Compliant Goal Systems 224

8.3 Summary 225

9 Conclusions, Contributions, and Recommendations **226**

9.1 Conclusions 226

9.1.1 Caveats 226

9.2 Contributions 227

9.3 Recommendations 230

Bibliography **232**

Appendices

A Quaternions **240**

A.1 Definition 240

A.2 Properties 240

A.3 Relation to the Rotation Matrix 241

B Orientation Error **242**

B.1 Differential Motion Vector or Euler Rotations 242

B.2 Quaternion Orientation Error 243

C Scenario Specification Database **245**

C.1 Manipulator Description Files (MDF): `theRobot.rbt` 246

C.2 Global Goal Description Files (GDF): `theGoals.dat` 246

C.3 Trajectory Description Files: `theTrajectory.spt` 246

C.4 Obstacle Description Files: `theObstacle.obs` 248

D RMDAC Gain Selection **250**

D.1 Control Rates and RMDAC 253

E Global Goal Generators Compared **257**

E.1 The Operational Space Formulation 257

E.1.1 Theoretical Performance 258

E.1.2 Simulation 259

E.1.3 Discussion 260

E.2 Resolved Motion Force Control 260

E.2.1 Theoretical Performance 264

E.2.2 Simulation 264

E.2.3 Discussion 265

List of Tables

3.1 The temporal scope of the NASREM layers. Each layer represents an increasingly (approximately an order of magnitude) longer planning horizon 36

5.2 Structure of the `CartesianState`. 94

5.3 Structure of the `LinkModel`. 96

5.4 Reference Link, Joint, and Motor parameters. Centroid is relative to link frame. 114

6.5 Reference Link, Joint, and Motor parameters. Centroid is relative to link frame. 124

6.6 Reference Payload parameters. Centroid is relative to gripper frame. 125

6.7 Knotpoints on the reference trajectory. 125

6.8 The reference RMDAC 'Best Gains' determined through trial and error. 129

6.9 'Best Gains' RMS error in position and orientation for 'step', 'reference' and 'spiral' trajectories. 129

6.10 Tabulation of position and orientation RMS L_2 error and normalized execution time as a function of manipulator degrees of freedom. 137

7.11 RMS error for a combined tracking and obstacle avoidance strategy. 154

7.12 Comparison of end effector RMS position and orientation error performance between reference and fail-safe behaviour of link 9. 172

7.13 Tabulation of local joint centering PD gains versus RMS task space position error and RMS centering error for a simple joint centering strategy. 179

7.14 Tabulation of joint centering PD gains and setpoint for the constant compliance retraction local goal strategy. 190

7.15 RMS error for the constant compliance retraction strategy. 190

7.16 Tabulation of *increasing* and *decreasing* joint centering PD gains for the variable compliance local goal strategy. 196

7.17 RMS error for "increasing" and "decreasing" variable centering gain strategies. 196

7.18 Comparison of RMS errors for a tracking, obstacle avoidance ($\eta = 10.0$), and centering ($k_p = 100\ k_d = 20$) combined strategies. 197

8.19 RMAC and Multiagent Control RMS end effector and joint centering performance. 218

D.20 A tabulation of initial values (italic) and integral gains (bold) for a set of scenarios. Each integral can use up to three gains, u_0, u_1, and u_2. In this study u_0 is omitted while u_2 never varies, thus the listed gains represent the value of u_1 in each integral. 250

D.21 Tabulation of position and orientation RMS L_2 error with variation in error weights,$\mathbf{W}_p$. . . 252

D.22 Tabulation of position and orientation RMS L_2 error as a function of variation of the weight $\mathbf{W}_v$ given $w_{pi} = 900$. 253

D.23 Tabulation of position and orientation RMS L_2 error as a function of the Feedback integral gain, α_{1i} given $\beta_{1i} = 1.0$. 253

D.24 Tabulation of position and orientation RMS L_2 error as a function of the Feedback integral gain, β_{1i} given $\alpha_{1i} = 1.0$. 253

D.25 Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, λ_{1i} given $\nu_{1i} = 0.0$, $\gamma_{1i} = 0.0$. 253

D.26 Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, γ_{1i}, given $\nu_{1i} = 0.0$, $\lambda_{1i} = 2.0$. 254

D.27 Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, ν_{1i} given $\gamma_{1i} = 2.0$, $\lambda_{1i} = 2.0$.Note that cases 23 and 24 were unstable in the step response. 254

D.28 Tabulation of position and orientation RMS L_2 error as a function of the auxiliary integral gain, δ_{1i}, given $\gamma_{1i} = 2.0$, $\lambda_{1i} = 2.0$ 254

List of Figures

1.1 The Sense-Model-Plan-Act (SMPA) manipulator control cycle. An example of the prototypical robot control cycle borne out of early AI research. 2

2.2 A 6 degree of freedom elbow manipulator. Note coordinate systems conform to Denavit Hartenberg conventions. 15

2.3 The Range and Null spaces of the Jacobian and Jacobian Transpose. 20

2.4 An armature controlled dc drive joint motor and gear train model. 22

3.5 Fisher's corrected Hybrid Position Force Control. **S** is the selection matrix. $\mathbf{J}^+$ is the pseudoinverse of the Jacobian and $\mathbf{z_q}$ and $\mathbf{z}_\tau$ are arbitrary vectors. 33

3.6 A block diagram of the Visual servoing approach to target tracking. Desired reference values are in feature space $\mathbf{f}_{\mathrm{ref}}$. Image feature changes are computed through a feature Jacobian $\mathbf{J}_{\mathrm{feat}}$. 33

3.7 Khatib's operational space formulation. Feedforward cartesian dynamics and obstacle avoidance forces are summed to drive a manipulator along a stable collision free trajectory. Colbaugh's configuration control replaces the potential field model with sensing and the feedforward dynamics with adaptive control. 35

3.8 The NASA/NIST Standard Reference Model for telerobot control. 35

3.9 Carnegie Mellon's Task Control Architecture for the Ambler hexapod. Note the centralized planner and distributed reactive controller. 38

3.10 Carnegie Mellon's Distributed Architecture for Mobile Navigation for the NAVLAB series of robots. Each behaviour votes on every possible command (e.g. steering radii) and all votes are processed within the arbiter. 39

3.11 A subsumption network. Perception (P) drives augmented finite state machines (M#) to output messages. Suppression nodes substitute horizontal line messages with vertical (tap) messages. Similarly Inhibition nodes disable line messages if a tap message is received. 40

3.12 A SAN network example. Note the interconnections sensor (S), hidden (H) nodes and Actuator (A) nodes. Each node has the structure expanded at right. The hidden nodes act as an interactuator information mechanism. 42

3.13 By applying both direct $\bigoplus$ and temporal $\bigotimes$ combination to the same basic behaviours higher-level behaviours can be generated. In this example, safe wandering is used to generate both flocking and foraging. Similarly aggregation is used in both foraging and in surrounding. . . . 45

4.14 A generic plant, $\mathbf{G}_j(\cdot)$, with control input $\mathbf{u}_i(t)$ and behaviour $\mathbf{y}_j(t)$. 53

4.15 A goal $\mathbf{r}_i(t)$ input to the generic controller, $\mathbf{H}_i(\cdot)$, with output $\mathbf{u}_i(t)$ controls the plant, $\mathbf{G}_j(\cdot)$ making the behaviour, $\mathbf{y}_j(t)$. 55

4.16 A goal $\mathbf{r}_i(t)$ input to the generic controller, $\mathbf{H}_i(\cdot)$, observes disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t)$ and control the plant, $\mathbf{G}(\cdot)$ making the behaviour, $\mathbf{y}_j(t)$. 56

4.17 A set of goals $\mathbf{r}_i(t) : 1 \leq i \leq b$ input to generic controllers, $\mathbf{H}_i(\cdot) : 1 \leq i \leq b$, each observing select disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t) : 1 \leq i \leq b$ to the arbitrator $\mathsf{A}_j(\cdot)$. The composite output $\mathbf{u}_{cj}$ controls the plant, $\mathbf{G}_j(\cdot)$ to make the behaviour, $\mathbf{y}_j(t)$. 58

4.18 A set of behaviour controllers $\mathbf{r}_i(t)$ and $\mathbf{H}_i(\cdot)$, each observe select disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t)$ to the arbitrator $\mathsf{A}_\mathsf{j}(\cdot)$. $\mathsf{A}_\mathsf{j}(\cdot)$ arbitrates amongst these controllers to track a desired goal $\mathbf{r}_{cj}(t)$. The composite output $\mathbf{u}_{cj}(t)$ controls the plant, $\mathbf{G}_j(\cdot)$ to make the behaviour, $\mathbf{y}_j(t)$. . 59

4.19 A simple linear agent arbitration model. 65

4.20 A multiagent controller model. Note that each agent acts upon the global plant $\mathbf{G}$, though each agent may apply control effort to only a portion of the plant $\mathbf{G}_j$. 70

4.21 A decomposed model of a multiagent controller. Though each agent acts upon a local plant $\mathbf{G}_j$, disturbances dynamically bind plants to form a single global plant $\mathbf{G}$. The aggregate relation $\mathbf{f}(\cdot)$ represents the binding between the local and global behaviour states $\mathbf{y}_j$ and $\mathbf{y}$ respectively. 72

4.22 The rows of the Jacobian transpose are a projection operator from the global goal (a force vector) to the local goal space (an actuator moment or thrust vector). 83

4.23 With a global goal $\mathbf{f}_d$ and the global goal proxy, $\mathbf{J}_i^T(\mathbf{p}_{i-1}, \mathbf{p}_N)$, manipulator end effector control can be distributed amongst N *agents*, each with local and global goals $\mathbf{r}_d$ and $\mathbf{x}_d$ respectively. 87

5.24 The interprocess data flow of the Multiprocess Manipulator Simulator. As `GP` is not a mandatory process, it appears as a dashed circle. 99

5.25 Interprocess data flow between the link processes $1 \ldots N$, global goal process `GP`, process manager `SI`, and dynamic model `DM`. Note that the link state $\mathbf{x}_i = [q_i \; \dot{q}_i]^T$. 112

5.26 The 3R planar reference model used to verify `DM` against `NEDYN`. 114

5.27 Here, the superimposed displacement responses of both the symbolic NEDYN and numerical 3R planar reference manipulator (link 1, top; link 2, middle; link 3, bottom) indicate excellent agreement between the two methods. 115

6.28 The Reference Manipulator: a ten degree of freedom planar revolute manipulator initially lying along the **x** axis. 124

6.29 The Reference end effector position (top) and orientation (bottom) trajectories. Each segment employs a parabolic velocity profile in both position and orientation. The position trajectory plot is annotated with time milestones. 126

6.30 End effector trajectory tracking under 'Best Gains' RMDAC. Note oscillation prior to convergence. 130

6.31 Trajectory tracking history for the reference trajectory under 'Best Gains' RMDAC. 131

6.32 End effector tracking performance over an unusual spiral trajectory in which the radius and angular displacement vary according to a parabolic velocity profile. 132

6.33 Trajectory tracking history for the spiral trajectory under 'Best Gains' RMDAC. 133

6.34 End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 5 degree of freedom planar manipulator. 134

6.35 End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 15 degree of freedom planar manipulator. 135

6.36 End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 20 degree of freedom planar manipulator. 136

6.37 End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint force saturation ($\mathbf{u}_{\text{sat}} = 1000$ N-m). 138

7.38 A simple link agent controller model. 150

7.39 Semilog plots of the potential field (top) and repulsive force (bottom) as a function of normalized range. 152

7.40 The reference manipulator avoids a small stationary sphere while tracking the reference trajectory. The sphere is 0.250m in diameter at $\mathbf{x}_{\text{obs}} = [1.00 \quad -0.750]^T$. The RMOA controller uses a clearance of 0.75 m and gain of $\eta = 100.0$. 155
7.41 End effector trajectory tracking performance of the reference manipulator engaging in multiagent trajectory tracking and obstacle avoidance. 156
7.42 Evolution of the Global Goals within links 1(top) and 2 (bottom). 157
7.43 Evolution of the Global Goals within links 3(top) and 4 (bottom). 158
7.44 Evolution of the Global Goals within links 5(top) and 6 (bottom). 159
7.45 Evolution of the Global Goals within links 7(top) and 8 (bottom). 160
7.46 Evolution of the Global Goals within links 9(top) and 10 (bottom). 161
7.47 End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint limit avoidance ($\eta = 5.0$ N-m). 165
7.48 Both RMDAC end effector global and joint limit local goals are engaged. 166
7.49 End effector trajectory error for the reference manipulator and payload under the failure of link 9. 170
7.50 The manipulator configurations at the knotpoints of the reference trajectory for the reference manipulator and payload under the failure of link 9. The 'fail safe' braking behaviour locks link 9 to link 8. 171
7.51 The structure of a multiagent system in which each agent has both local centering and global trajectory tracking (and other) behaviours. 173
7.52 End effector trajectory for the reference manipulator and payload under both RMDAC end effector global and joint centering local goals. 180
7.53 The manipulator configurations at the knotpoints of the reference trajectory for the reference manipulator and payload under both RMDAC end effector global and joint centering local goals. Note the manipulator *simultaneously* adopts a 'leaf spring' configuration while tracking the reference trajectory, indicating that joint centering forces are acting in $\mathbf{N}(\mathbf{J})$. 181
7.54 Centering and tracking torques for links 1(top) and 2 (bottom) of the reference manipulator tracking the reference trajectory. 182
7.55 Centering and tracking torques for links 3 (top) and 4 (bottom) of the reference manipulator tracking the reference trajectory. 183

7.56 Centering and tracking torques for links 5 (top) and 6 (bottom) of the reference manipulator tracking the reference trajectory. 184

7.57 Centering and tracking torques for links 7 (top) and 8 (bottom) of the reference manipulator tracking the reference trajectory. 185

7.58 Centering and tracking torques for links 9 (top) and 10 (bottom) of the reference manipulator tracking the reference trajectory. 186

7.59 Both RMDAC end effector global and joint centering local goals with alternating setpoints are engaged. 188

7.60 End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering 'retraction' local goals. 189

7.61 End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering local goals of linearly *increasing* stiffness. 192

7.62 Both RMDAC end effector global and joint centering local goals with increasing stiffness are engaged. 193

7.63 End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering local goals of linearly *decreasing* stiffness. 194

7.64 Both RMDAC end effector global and joint centering local goals with decreasing stiffness are engaged. 195

7.65 End effector trajectory error for the reference manipulator and payload under both RMDAC end effector global, RMOA obstacle avoidance, and joint centering local goals. 198

7.66 Reference manipulator configurations at reference trajectory knotpoints under RMDAC end effector and obstacle avoidance global goals ($\eta = 10.0$) and joint centering local goals($k_q = 100$, $k_w = 20$). Note 'leaf spring' configuration and avoidance reside in $\mathbf{N}(\mathbf{J})$. 199

7.67 Range to surface, avoidance, centering, and tracking torques for links 1(top) and 2 (bottom) of the reference manipulator tracking the reference trajectory. 200

7.68 Range to surface, avoidance, centering, and tracking torques for links 3 (top) and 4 (bottom) of the reference manipulator tracking the reference trajectory. 201

7.69 Range to surface, avoidance, centering, and tracking torques for links 5 (top) and 6 (bottom) of the reference manipulator tracking the reference trajectory. 202

7.70 Range to surface, avoidance, centering, and tracking torques for links 7 (top) and 8 (bottom) of the reference manipulator tracking the reference trajectory. 203

7.71 Range to surface, avoidance, centering, and tracking torques for links 9 (top) and 10 (bottom) of the reference manipulator tracking the reference trajectory. 204

7.72 End effector tracking performance overconstrained by RMOA and N Centering controllers ($\mathbf{K}_q$ = diag(100) , $\mathbf{K}_w$ = diag(20)) and the agents arbitration vector $\mathbf{k}_{\mathsf{A}_j} = [k_{\text{track}}\ k_{\text{avoid}}\ k_{\text{centering}}] =$ [1.0 1.0 0.5]. 207

8.73 Convergence of techniques that form Multiagent Manipulator Control. 213

8.74 End effector trajectory for the reference manipulator and payload under centralized RMAC end effector and joint centering task assigned to the null space. 216

8.75 End effector trajectory for the reference manipulator and payload under centralized RMAC end effector and decentralized joint centering auxiliary tasks ($\mathbf{K}_q$ = diag(100.0) , $\mathbf{K}_w$ = diag(20.0)). 219

8.76 About the end effector position $\mathbf{x}(t)$, the desired global goal $\mathbf{x}_d$ is related to the desired local goal $\mathbf{x}_{qd}$ through the vectors $\mathbf{x}_e$, $\mathbf{x}_{qe}$ and $\mathbf{e}$. 222

C.77 A robot Specification Database flat (ASCII) file. Note that the `link` list order is from manipulator base (frame 1) to end effector (frame N). 246

C.78 A Global Goal Database flat (ASCII) file. 246

C.79 An Trajectory Database flat (ASCII) file. 247

C.80 An Obstacle Database flat (ASCII) file. 248

D.81 Absolute end effector step response under 'Best Gains'. 255

E.82 RMAC end effector trajectory tracking performance of the reference trajectory with exact manipulator parameter estimates. 260

E.83 RMAC (equivalent to OSF) configuration history of the reference trajectory for the reference manipulator and payload. Note that despite precise trajectory tracking, the manipulator adopts convoluted configurations, a clear indication of free motion in $N(\mathbf{J})$. 261

E.84 The Force Convergent Controller 262

E.85 End effector trajectory error performance during the reference trajectory *without* Force Convergent Control at 120hz.The manipulator carries the reference payload and the controller employs PD gains of $\mathbf{K}_p = 100$ and $\mathbf{K}_d = 20$. 265

E.86 End effector trajectory tracking performance during the reference trajectory under Force Convergent Control (RMFC nominal rate of 120hz, actual control rate 480Hz). 266
E.87 End effector trajectory performance during the reference trajectory under Force Convergent Control(RMFC nominal rate of 30hz, actual control rate 120Hz). 267
E.88 Trajectory tracking history for the reference trajectory under Force Convergent Control (RMFC nominal rate of 30hz, actual control rate 120Hz). 268

Acknowledgement

Entering a doctoral engineering program, a daunting task at best, would have been unthinkable without the care, thoughtful guidance, and support of my supervisor, Dr. Dale Cherchas. Dale's kindness, optimism, and vision remain a standard I can only hope to follow.

The understanding, patience, and advice of the Research and Examination Commitees, Dr. Vinod Modi, Dr. C. Ma, Dr. Clarence de Silva, Dr. Peter Lawrence, Dr. William Gruver, Dr. Farrokh Sassani, Dr. Jim Little, and Dr. Xi of the National Research Council supplied thought provoking questions and suggestions for which I am grateful.

I am grateful, too, for the contributions of both the National Research Council and NSERC to this research. However, the Alberta Research Council deserves special recognition for their collaboration and support. In particular, I should thank Keith Crystall both for supporting the collaboration and, with Pat Feighan, suffering my intrusions in their busy schedule. Their perspective and sense of humour did much to preserve my sanity.

Above all, I must thank my better half, Simone, for her enduring patience and boundless confidence; our precocious daughter, Valerie, a source of vital distraction; and of course, Mum, Dad, Stephen and Elizabeth – providers of solid advice and often a good debate!

Like many before me, I must finally pay tribute to Isaac Asimov, the father of robotics, for his legacy:

Asimov's Three Laws of Robotics

I. (Safety) *A robot may not injure a human being, or, through inaction, allow a human being to come to harm.*

II. (Service) *A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.*

III. (Prudence) *A robot must protect its own existence, as long as such protection does not conflict with the First or Second Laws.*

–Isaac Asimov, *Runaround*, Astounding Science Fiction, March 1942.

Nomenclature

In the following nomenclature section significant *acronyms* are listed alphabetically. This is followed by a nomenclature listing of significant *symbols* grouped into sections by topic.

With few exceptions, this dissertation adopts the following notational conventions:

- italic roman symbols, (e.g. x), signify *scalars.*
- small case bold roman symbols, (e.g. $\mathbf{x}$), signify *column vectors*,
- capitalized bold roman symbols, (e.g. $\mathbf{X}$), signify *matrix* quantities .

Acronyms

APF	Artificial Potential Field (from OSF).
OSF	the Operational Space Formulation (resolved motion computed torque).
JTC	Jacobian Transpose Control.
FCC	Force Convergent Control (from RMFC).
PSH	Point Subject to Hazard (similar to PSP).
PSP	Point Subject to Potential (from OSF).
RMAC	Resolved Motion Acceleration Control.
RMDAC	Resolved Motion Decentralized Adaptive Control or Configuration Control
RMFC	Resolved Motion Force Control.
RMOA	Resolved Motion Obstacle Avoidance.
RMPC	Resolved Motion Position Control.
RMRC	Resolved Motion Rate Control.

Manipulator Kinematics and Dynamics.

$\mathbf{q}$	the configuration space position vector.
$\mathbf{x}$	the task space position vector.

$\mathbf{f}(\mathbf{q})$	the Forward Solution, a map from configuration space to task space.
$\mathbf{J}(\mathbf{q})$	$= \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}}$ the Jacobian of the Forward Solution.
$\mathbf{D}(\mathbf{q})$	the inertia matrix in configuration space coordinates.
$\mathbf{g}(\mathbf{q})$	the gravitational forces in configuration space coordinates.
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	coriolis, centrifugal forces in configuration space coordinates.
$\mathbf{M}(\mathbf{q})$	the inertia matrix in cartesian space coordinates.
$\mathbf{N}(\mathbf{q})$	the gravitational forces in cartesian space coordinates.
$\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}})$	coriolis, centrifugal forces in cartesian space coordinates.
u	link control forces
$\mathbf{P}(\mathbf{x})$	the gravitational forces in task space coordinates.
t	time.
τ	manipulator joint torque.

Denavit Hartenberg Parameters.

θ_i	revolute actuator displacement about $\mathbf{z}_i$
d_i	prismatic actuator displacement about $\mathbf{z}_i$
α_i	link twist
a_i	link length

The Link Model.

r_k	gear train coefficient for the kth link.
r	the redundancy rate $r = n - m$
d_k	link disturbance forces
J_m	link motor inertia
J_{eff}	effective link inertia
$\mathbf{p}_i$	the position of link frame i in world coordinates
$\mathbf{p}_i^*$	the position of the ith link's centroid in *link* coordinates
$\mathbf{R}_i$	the orientation of link frame i in world coordinates

$\mathbf{w}_i$	the angular velocity of link frame i in world coordinates

The Agent Model.

$\mathcal{G}$	a goal space.
$\mathbf{r}(t)$	a trajectory or *goal* in $\mathcal{G}$.
$\mathcal{B}$	a behaviour space.
$\mathbf{y}(t)$	a trajectory or *behaviour* (or response) of some dynamic system in $\mathcal{B}$.

Agent Definitions

$\mathsf{A}(\cdot)$	an *agent.*
b	the number of behaviour controllers in agent control system.
$\mathbf{G}_j(\cdot)$	the jth subplant of a dynamic system.
$\mathbf{y}_j(t)$	the behaviour (or response) of $\mathbf{G}_j(\cdot)$.
$\mathbf{H}_i(\cdot)$	an agent's ith controller.
$\mathbf{r}_i(t)$	the goal of $\mathbf{H}_i(t)$.
$\mathbf{u}_i(t)$	the output of $\mathbf{H}_i(t)$.
$\mathbf{r}_c(t)$	a *composite* goal of A.
$\mathbf{u}_c(t)$	the *composite* output of A.
$\mathbf{r}_{\mathsf{A}}(t)$	$= [\mathbf{r}_1(t)\ldots\mathbf{r}_b(t)]^T$ a column vector of all $\mathbf{r}_i$.
$\mathbf{u}_{\mathsf{A}}(t)$	$= [\mathbf{u}_1(t)\ldots\mathbf{u}_b(t)]^T$ a column vector of all $\mathbf{u}_i$.
$\mathbf{k}_{\mathsf{A}}(t)$	a linear arbitration vector such that $\mathbf{u}_c(t) = \mathbf{k}_{\mathsf{A}}^T(t)\mathbf{u}_{\mathsf{A}}$.

Multiagent Definitions

N	the number of agents in multiagent system.
$\mathsf{A}_j(\cdot)$	the jth *agent* where $1 \leq j \leq N$.
b_j	the number of behaviour controllers in $\mathsf{A}_j(\cdot)$.
$\mathbf{H}_{ij}(\cdot)$	the ith controller of A_j.
$\mathbf{r}_{ij}(t)$	the goal of $\mathbf{H}_{ij}(t)$.
$\mathbf{u}_{ij}(t)$	the output of $\mathbf{H}_{ij}(t)$.
$\mathbf{r}_{\mathsf{A}_j}(t)$	$= [\mathbf{r}_{j1}(t)\ldots\mathbf{r}_{jb_j}(t)]^T$ a column vector of all $\mathbf{r}_i$ in A_j.
$\mathbf{u}_{\mathsf{A}_j}(t)$	$= [\mathbf{u}_{j1}(t)\ldots\mathbf{u}_{jb_j}(t)]^T$ a column vector of all $\mathbf{u}_i$ in A_j.

$\mathbf{r}_{cj}(t)$	a *composite* goal of $\mathsf{A}_j(t)$.
$\mathbf{r}_{\mathsf{nA}}(t)$	$= [\mathbf{r}_{\mathsf{A}_1}(t) \ldots \mathbf{r}_{\mathsf{A}_N}(t)]^T$, the vector of controller setpoints for all $\mathsf{A}_j(t) : 1 \leq j \leq N$.
$\mathbf{r}_{c\mathsf{nA}}(t)$	$= [\mathbf{r}_{c1}(t) \ldots \mathbf{r}_{cN}(t)]^T$, the vector of the *composite* goals for all $\mathsf{A}_j(t) : 1 \leq j \leq N$.
$\mathbf{u}_{cj}(t)$	the *composite* output of $\mathsf{A}_j(t)$.
$\mathbf{u}_{\mathsf{nA}}(t)$	$= [\mathbf{u}_{\mathsf{A}_1}(t) \ldots \mathbf{u}_{\mathsf{A}_N}(t)]^T$, the vector of controller outputs for all $\mathsf{A}_j(t) : 1 \leq j \leq N$.
$\mathbf{K}_{\mathsf{nA}}(t)$	a linear arbitration matrix such that $\mathbf{u}_{c\mathsf{nA}}(t) = \mathbf{K}_{\mathsf{nA}}(t)\mathbf{u}_{\mathsf{nA}}$.
$\mathbf{u}_{c\mathsf{nA}}(t)$	$= [\mathbf{u}_{c1}(t) \ldots \mathbf{u}_{cN}(t)]^T$, the vector of *composite* output for all $\mathsf{A}_j(t) : 1 \leq j \leq N$.
$\mathcal{G}_j$	the goal space of $\mathsf{A}_j(t)$.
$\mathcal{G}^j$	the image of $\mathcal{G}_j$ in $\mathcal{G}$.
$\mathcal{B}_j$	the behaviour space of $\mathsf{A}_j(t)$.
$\mathcal{B}^j$	the image of $\mathcal{B}_j$ in $\mathcal{B}$.
$\mathbf{G}(\cdot)$	a plant composed of n subplants.
$\mathbf{y}_{\mathsf{nA}}(t)$	$= [\mathbf{y}_1 \ldots \mathbf{y}_N]^T$ a vector of the behaviours (or dynamic response) of N agents.
$\mathbf{y}(t)$	$= \mathbf{f}(\mathbf{y}_{\mathsf{nA}})$, the global or *aggregate* behaviour (or dynamic response) of $\mathbf{G}(\cdot)$.
$\mathbf{f}(\mathbf{y}_{\mathsf{nA}})$	e.g. $\mathbf{y}(t) = \mathbf{f}(\mathbf{y}_{\mathsf{nA}})$: $\mathcal{B}_1 \times \ldots \times \mathcal{B}_N \rightarrow \mathcal{B}$ – an *aggregate relation.*
$\mathbf{g}(\mathbf{r}(t))$	e.g. $\mathbf{r}_{c\mathsf{nA}}(t) = \mathbf{g}(\mathbf{r}(t))$: $\mathcal{G} \rightarrow \mathcal{G}_1 \times \ldots \times \mathcal{G}_N$ – an *inverse aggregate relation.*

Cartesian Controllers

G	a global goal couplet.
$\mathbf{i}_i, \mathbf{j}_i, \mathbf{k}_i$	the basis vectors of the ith coordinate frame.
$\mathbf{p}_{\text{app}}$	the objective position of a global goal couplet.
$\mathbf{x}(t)$	a state in task space $\{\mathbf{p}, \mathbf{R}, \dot{\mathbf{p}}, \omega\}$, often $\mathbf{x}(t) = \mathbf{x}_N(t)$, the end effector state
$\mathbf{x}_d(t)$	the desired task space position with elements $\mathbf{x}_{di}(t) : 1 \leq i \leq M$
$\mathbf{x}_e(t)$	$= \mathbf{x}_d(t) - \mathbf{x}(t)$ the task space position error vector.
X_i	$= [q_i \; \dot{q}_i]^T$ the joint state.
X	$= [\mathbf{x}_N \; \dot{\mathbf{x}}_N]^T$ the end effector state.
$\mathbf{f}_d(t)$	the desired force at the end effector.
$\mathbf{f}_a(t)$	the applied force at the end effector.

OSF

U_{art}	an artificial potential field in OSF
ρ	the range to an obstacle surface
ρ_0	the trigger range to an obstacle surface
∇	the vector gradient

RMDAC

$\mathbf{d}(t)$	an auxiliary gain.
$\mathbf{C}(t)$	an feedforward gravitational gain.
$\mathbf{B}(t)$	an feedforward Coriolis gain.
$\mathbf{A}(t)$	an feedforward Inertia gain.
$\mathbf{K}_p(t)$	an adaptive cartesian position error gain.
$\mathbf{K}_v(t)$	a adaptive cartesian velocity error gain.
$d_i(t)$	the ith auxiliary gain.
$c_i(t)$	the ith feedforward gravitational gain.
$b_i(t)$	the ith feedforward Coriolis gain.
$a_i(t)$	the ith feedforward Inertia gain.
$K_{pi}(t)$	the ith cartesian position error gain.
$K_{vi}(t)$	the ith cartesian velocity error gain.
$r_i(t)$	the ith component of the weighted cartesian error.
w_{pi}	the ith component of the position error weighting gain
w_{vi}	the ith component of the velocity error weighting gain

RMFC

b	a convergence coefficient for the Robbins Monroe Stochastic Estimator.
k	a counter for the Robbins Monroe Stochastic Estimator.

Chapter 1

Introduction

As robot control engages progressively more complex problems, centralized supervisory architectures encounter barriers to real time performance, specifically: computational complexity coupled with insufficient computing power and impoverished sensor resources. Despite startling advances in hardware and software technology and similarly surprising cost reductions, these fundamental barriers remain unchanged. This thesis investigates an alternative to centralized supervisory control of manipulation robots known as multiagent control.

Centralized supervisory robot control architectures are usually based on variants of the sense-model-plan-act (SMPA) cycle [Brooks, 1989c] [1]. The dominance of this architecture can be traced to early AI [Brooks, 1991a] in which artificial intelligence was broken into four distinct problems: sensing, modelling, planning, and action. This decomposition stemmed from the belief that cognition was a monolithic process in which the relatively simple phases of sensing and action supported complex symbolic modelling and planning engines. This 'reductionist' view of intelligent action focussed research into symbolic manipulation of the environment and task-level robot programming techniques (e.g. Stanford Research Institutes's 'Shakey' in which logic and planning were of primary interest). Task-level programming describes tasks through a hierarchy of symbolic operators fundamentally founded upon basic geometric relationships (i.e. homogeneous transformations, trajectories, etc.) within some coherent representation of the environment, a *world model.* Within a *structured,* slowly varying environment like an assembly line, world modelling and task level programming methods are more than sufficient to support safe, accurate, predictable, and cost effective automation.

The advantage of SMPA is that, *if* the environment is known a priori, so too is the robot's behaviour. The disadvantage, of course, is that *unstructured environments* (UE) are intrinsically unpredictable. Real time SMPA relies on a fragile chain of events: fast planning, precise modelling, and rich sensing. Since the latter tends to be expensive and/or unreliable in a UE, strictly following an SMPA model inevitably ensures a *robotic system's* performance never exceeds that of the worst sensor. Such systems become *input* and *compute*

[1] Brooks original papers describe this as the Sense-Think-Act cycle

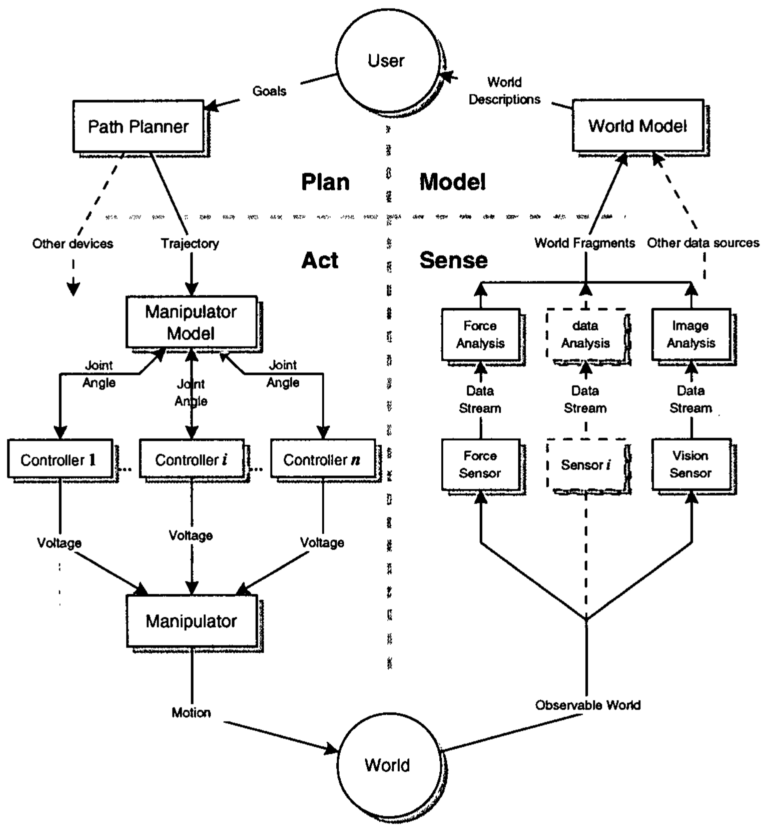


Figure 1.1: The Sense-Model-Plan-Act (SMPA) manipulator control cycle. An example of the prototypical robot control cycle borne out of early AI research.

bound, awaiting *all* relevant data prior to distilling plans and world models into a single action. In maximizing SMPA performance in a UE, computing resources can become sufficiently expensive and/or physically large that elements must be housed remotely, adding the burdens of communication delay, bandwidth limits, and tethered operations to input and compute bound performance.

It is not surprising, then, that research into practical, real time robotics has diverged from the SMPA model. Research, in artificial intelligence [Minsky, 1990], mobile robotics [Brooks, 1989b]-[Brooks, 1991b], [Connell, 1990], and high performance manipulation [Khatib, 1987, Colbaugh, 1989, Seraji, 1996a] in particular have challenged the reductionist model. While not entirely rejecting the utility of recognition, modelling, and compensation of environmental events, practical real time controllers are often designed to react to the world through the adoption of control arbitration *networks.* Using high speed sensors, minimal signal post processing, multiple inexpensive *embedded* computers execute one or more control strategies.

Such sensor driven, distributed, control arbitration processes are increasingly referred to as *agents.* In agent based systems, the *semantics* or meaning traditionally embodied in a world model is condensed into a sense-act relationship distributed over multiple application specific computing elements. In effect, task level *cognitive* elements are transformed into a set of control level *instinctive* responses or *behaviours* and combined through some *arbitration mechanism*, a greatly simplified model-planning system. It is important to note that this method seeks not to remove centralized cognitive or supervisory elements from the cycle but to distribute model based control to simpler application specific controllers and, in so doing, both reduce expense and improve autonomous performance. Proponents argue that agent architectures both simplify the design and improve robustness of robot systems by basing behaviour on the interaction of the robot with its environment.

Based on his research experience, an outspoken critic of traditional AI, Rodney Brooks, proposed a set of properties [Brooks, 1991a] he believed crucial to intelligent real time behaviour [2]:

Situatedness *The world is its own best model.* A robot should 'model' the world through sensing.

Embodiment *The world grounds regress.* A robot should reside *within* a physical system. Simulated robot capabilities usually regress in the real world.

Intelligence *Intelligence is determined by the dynamics of the interaction with the world.*

Emergence *Intelligence is in the eye of the beholder.* Intelligence emerges from the interaction of parts of the system.

[2] Italicized remarks are quotes from *Intelligence without Reason* [Brooks, 1991a], perhaps his most controversial paper.

Though unproven, these properties are, nevertheless, a useful summary of this new breed of simple, capable, real time robots. An example from manipulator control clarifies the difference between SMPA and *agent* control.

An Example

Consider the common tasks in redundant manipulator control: end effector trajectory tracking and obstacle avoidance.

At each control time step, an SMPA system uses high level machine vision and world modelling to capture and extract surrounding hazards for construction of object models. The manipulator's configuration space trajectory is determined through a centralized process such as resolved motion acceleration controller (RMAC). In RMAC the desired cartesian acceleration $\ddot{\mathbf{x}}_d(t)$, is transformed into joint torques, $\mathbf{u}$, based on the pseudoinverse of the end effector Jacobian, $\mathbf{J}^\dagger(\mathbf{x})$, a PD cartesian controller with gains k_p and k_v, and an estimate of the manipulator dynamic model ($\hat{\mathbf{D}}(\mathbf{q})$, $\hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$, and $\hat{\mathbf{g}}(\mathbf{q})$):

$$\begin{aligned} \ddot{\mathbf{q}}_d(t) &= \mathbf{J}^\dagger \left[\ddot{\mathbf{x}}_d(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v \dot{\mathbf{x}}_e + k_p \mathbf{x}_e \right] \\ &\quad -(\mathbf{I} - \mathbf{J}^\dagger \mathbf{J})(\mathbf{K}_{\text{obs}} \sum_{j=1}^{j=\eta} \mathbf{J}_j^T \mathbf{f}_{\text{model}}) \\ \mathbf{u} &= \hat{\mathbf{D}}(\mathbf{q})\ddot{\mathbf{q}}_d + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}) \end{aligned}$$

where η and κ are the number of joints and obstacles respectively, $\mathbf{f}_{\text{model}}$ is a model based obstacle avoidance strategy, and $\mathbf{J}_j^T$ is the Jacobian of a nearest point of contact on link j. In this approach the obstacle avoidance control is explicitly inserted into the null space of the Jacobian, ensuring the end effector trajectory is not perturbed. To achieve obstacle free trajectory tracking, sensing and modelling must work in concert to supply the centralized controller with accurate data.

In contrast, an agent-like system employs embedded low level range sensing in each link to trigger obstacle avoidance *behaviours*. No attempt at obstacle mapping or representation is made. By combining obstacle avoidance and end effector trajectory tracking control behaviours an obstacle free configuration space trajectory can be achieved. In this thesis this is achieved through an adaptive end effector trajectory controller $\mathbf{f}_d(\mathbf{x}_d(t))$, and a reactive obstacle avoidance controller embedded into each link, $\mathbf{f}_{\text{PSH}_k}(\cdot)$. For the jth link :

$$\mathbf{u}_{cj} = [1 \;\; \delta_{\text{avoid}}(t)] \begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{avoid}} \end{bmatrix}$$

$$\delta_{\text{avoid}}(t) = \begin{cases} 1 & \text{if } |\mathbf{x}_{ik}| \leq c \\ 0 & \text{if } |\mathbf{x}_{ik}| > c \end{cases}$$

where c is a triggering range and $|\mathbf{x}_{ik}|$ is magnitude of the distance from the kth link to the ith obstacle. The tracking and avoidance controllers respectively are:

$$\begin{aligned} \mathbf{u}_{\text{track}} &= \mathbf{J}_j^T(\mathbf{p}_n, \mathbf{x}_{j-1})\mathbf{f}_d(\mathbf{x}_d(t)) \\ \mathbf{u}_{\text{avoid}} &= \sum_{k=1}^{N-j} \mathbf{J}_j^T \mathbf{f}_{\text{PSH}_k}(\sum_{i=1}^{i=\kappa} \mathbf{x}_{ik}) \end{aligned}$$

where $\mathbf{J}_j^T$ is the jth row of a generic Jacobian Transpose or *global proxy* and N is the number of links. In this approach, the adaptive end effector controller serves two purposes normally requiring a manipulator model. It ensures accurate trajectory tracking and forces all additional behaviours into the null space of the Jacobian. Sensing and control are decentralized and need not be synchronous, while modelling of both manipulator and environment are drastically reduced.

1.1 Motivation

This research was motivated by the desire to understand multiagent systems and by the demand for more capable 'real world' robotic manipulation systems. The inspiration for this work and perhaps the most famous example of agent based robotic systems is Brooks' subsumption architecture [Brooks, 1989c]. While subsumption represents only one approach in agent design, the simplicity and effectiveness of the methodology represented a wake up call to the robotics community. Similarly, the early work of Wu and Paul [Wu, 1982], Khatib[Khatib, 1985], and more recent research by Seraji et al [Seraji, 1989b, Colbaugh, 1989, Glass, 1995] at JPL acted as both foundation and guiding light for much of the new material presented here.

Despite the remarkable behaviour exhibited by agent and multiagent systems and the simplicity of their design, the theoretical foundations for these systems remains relatively shallow. While significant effort has been made to set agent control in the context of hybrid systems (e.g. Zhang and Mackworth [Zhang, 1995]), the fundamental problems of arbitrator and controller design, coupling are unresolved. Few investigators have explicitly described their design methods in control theoretic terms. Thus agent design remains inconsistent, ad hoc, and experimental.

In real time manipulation, Jacobian Transpose systems have achieved an important balance between computing resources, performance and cost within monolithic reactive architectures. This thesis will show that these same systems represent an avenue to an area gaining popularity in mobile robotics – the use of multiagent teams to control complex systems.

A key motivation for this work was the potential impact of a genuine autonomous multiprocess manipulator control architecture on manipulator performance and capabilities. If possible, the configuration of a given manipulator need not be limited to static nonredundant serial manipulators. Indeed the arrangement and construction of the manipulator can be both arbitrary and time varying, presenting some interesting possibilities including expandable, modular manipulators. All of which pose significant barriers to traditional, centralized supervisory control (or, for that matter, single agent) systems. Thus the application of multiagent control to manipulation expands the definition of manipulation systems while providing simple, robust, manipulation architectures at lower cost.

1.2 Survey of Related Research

1.2.1 Multiagent Control

The term *agent* originally appeared in distributed computation research (e.g. agent oriented programming [Shoham, 1993]) and decentralized artificial intelligence. In these systems, independent processes or *intelligent agents* act autonomously to perform tasks in the interest of a user (e.g. an electronic mail filter).

As applied to robots, *agent* often implies a number of relatively recent techniques, including reactive control, behavioural control, subsumption, sensor actuator networks, motor schema, discrete event or hybrid systems and others.

Robots as Agents

The concept of robot agent has evolved from a large number of noteworthy 'agent-like' robotic systems. Perhaps the earliest demonstration of this design philosophy appears in Walter's [Walter, 1950a, Walter, 1950b] published an account of reactive mobile robots, Elmer and Elsie [3] in 1951. These entirely analog 'shoe box' robots exemplified pure reactive control. With the widespread use of computers in the '60's and '70's such analog methods were supplanted by model based techniques. Perhaps the most famous of these are SRI's Shakey and the allied planner STRIPS, the top down philosophy of which that has dominated research robot.

Despite early successes, the demand for real world performance has driven the evolution of robot architectures away from a serial sequence or *single threaded*, model driven SMPA architectures to parallel or *multithreaded*, reactive architectures. In 1984 Raibert and Brown implemented a hopping robot that adopted a pragmatic, sensor driven, finite state machine architecture. By interacting with the surroundings,

[3] Walter inserts these into a mock biological species *Machina Speculatrix*

the robot could hop in place, travel and leap obstacles. In 1986 Brooks applied subsumption, a hierarchical behaviour arbitration system to a mobile cart, *Allen*[Brooks, 1989c]. and later refined on a walking robot, *Genghis* [Brooks, 1989b, Brooks, 1989a]. A student of Brooks, Connell, published an extensive description of a can collecting robot, *Herbert*, based on a heterogeneous behaviour network [Connell, 1990]. Further experiments using this architecture include learning in Mahadevan's mobile carts [Mahadevan, 1992]. Similarly Verschure investigated self organization and learning within a mobile cart [Verschure, 1992]. Hartley and Pipitone investigated subsumption and carrier aircraft landing systems [Hartley, 1991]. Using Motor schema or behaviours driven by a simple potential field world model, Arkin implemented a navigation system for a mobile robots [Arkin, 1987]. All demonstrated that autonomous behaviour often arises from the interaction of instinctive controllers, simple arbitration networks and the real world. The success of these techniques encouraged Brooks to make fervent arguments against traditional AI [Brooks, 1991b, Brooks, 1991a] and others to extend the technique [Jones, 1993].

A new robot control design philosophy seems to be emerging in which parallel *behaviours*, some reactive and some model driven, are combined, algebraicly or discretely, in real time to achieve a composite behaviour. Despite the current popularity of such approaches, a number of technical issues are unresolved or poorly defined:

i *What is a behaviour?* Variously understood to mean either an input plan, an applied control effort, or the response of a robotic system. Terminology remains a lingering problem in agent research.

ii *What is an agent?* There is no formal definition of agent other than 'a behaviour arbitration process' (e.g: [Mataric, 1994]) though agent characteristics seem widely recognized.

iii *How should agent activity evolve over time?* The popularity of subsumption has encouraged Kosecka and Bajcsy [Kosecka, 1994] to apply Discrete Event Systems[Ramadge, 1989] to behaviour sequence design. Similarly, Zhang and Mackworth have placed hybrid systems (i.e. possessing both discrete and continuous dynamics) into a formal mathematical context [Zhang, 1995], *Constraint Nets* though arbitration remains unexamined. Van de Panne and Fiume determine sequencing and behaviour design in Sensor-Actuator-Networks (SAN) through genetic algorithms [van de Panne, 1992]. A formal relationship between behaviours, the environment, and arbitration strategies has not been established.

iv *What constitutes emergent behaviour?* Most agree emergence is a qualitative, even subjective property of an agent based system [Brooks, 1989c, Mataric, 1994]. Indeed, only Steels [Steels, 1991] informally addressed the problem.

Multiagent Systems

Perhaps the first recognizable account of an *artificial* multiagent system appears in Walter's description of the analog robots *Elmer* and *Elsie*[Walter, 1950a, Walter, 1950b]. Significantly, Walter documents emergent or spontaneous organized behaviour in which simple controllers are combined to generate complex navigation, avoidance, and exploration behaviours.

In a seminal 1987 paper, Reynolds at Symbolics Corp., devised a multiprocess reactive method for animating flocking behaviour and obstacle avoidance in bird-like agents, *boids* [Reynolds, 1987]. This inspired Mataric's effort to assess agent interaction and behaviour [Mataric, 1992, Mataric, 1994] and Parker's investigation of troop movement [Parker, 1992b, Parker, 1992a]. Similarly, Tu and Grzeszczuk [Tu, 1994] designed surprisingly realistic reactive fish animations. To date all multiagent research has focussed on subjective behaviour assessment of teams (or herds) of mobile robots leaving additional unknowns to be itemized for multiagent control:

i *What constitutes global behaviour?* If agent behaviour is ill defined, so too is global multiagent behaviour. Though generally accepted that agent systems possess both individual behaviour and global team behaviour, there is no formal definition of collective behaviour.

ii *What is a multiagent system?* Specifically, what properties of an agent colony forms a recognizable, coordinated team of agents?

iii *How do agents interact?* Agent interaction is the crucial mechanism that enables agent teams to accomplish group objectives. Though agent interaction is widely observed, the mechanisms of interaction remain largely unscrutinized.

iv *What is the origin of emergent multiagent behaviour?* Without knowledge or consistent representation of interagent dynamics exploration of emergent or desirable, unplanned, group behaviour has been limited to broad characterizations such as [Steels, 1991]

1.2.2 Real-Time Manipulator Control

Though the literature on manipulator control is enormous (e.g. [Lewis, 1993]), virtually all control methods rely on at least one of four fundamental supervisory control strategies: joint position control, resolved motion rate control, resolved motion acceleration control, and Jacobian transpose control. Traditional industrial position control first thoroughly treated by Paul[Paul, 1981], remains the foundation of modern industrial manipulator control. Resolved motion rate control was established by Whitney in [Whitney, 1969] and extended in resolved motion acceleration control by Luh, Walker, and Paul [Luh, 1980b]. The latter now forms the basis of most redundant manipulator control schemes. Achieving the desired performance from these techniques characterizes virtually all research in manipulator control, from simple PD control to linearizing feedforward, feedback, optimal and adaptive centralized and decentralized control strategies. Another, less common, class of manipulator end effector control, Jacobian Transpose Control, was initially popularized by both Khatib [Khatib, 1985] and Wu [Wu, 1982] in the early 1980's and later promoted by Seraji at JPL [Seraji, 1989c].

1.3 Contributions of the Thesis

The primary contribution of this thesis is the development of a manipulation system that controls an N d.o.f. manipulator in the execution of multiple tasks using N independent processes or *agents* relying neither upon a centralized manipulator parameter model nor an auxiliary task distribution mechanism. The models and techniques used in the course of this research leading to the achievement of this task include:

Theoretical Foundation Agent and multiagent systems theory has little foundation in traditional control theory and, furthermore, is often obscured by inconsistent, vague terminology. To classify and formalize both agent and multiagent architectures required the application of standard control theoretic definitions to terms and components commonly referenced in reactive, behavioural, and other agent based control schemes. The translation of these terms into a formal control theoretic context is new and facilitates precise specification, comparison and reproduction of agent architectures.

Agent and Multiagent Architectures With inconsistent terminology and weak theoretical foundations, a control theoretic specification of agent and multiagent architectures was not previously possible. By formally translating and defining terms and components, a formal architecture could be proposed in which an *arbitrator* combines a set of task specific *behaviour controllers* into a composite command which, applied to the *plant*, generates a response or *behaviour*. It was further shown that to achieve a desired behaviour, an arbitrator could use *explicit*, *implicit*, and *emergent* arbitration to combine controllers through either model based local goal assignment, broadcast global goal assignment, or autonomous local goal assignment, respectively.

A formal multiagent architecture was proposed based on this agent model. A set of agents were shown to generate *global behaviour* through the combination of *local behaviours* through a binding or *aggregate relation*. It was also shown that *inversions* of this relation, governed by the Inverse Function Theorem, could be classified according to *explicit*, *implicit*, and *emergent coordination* strategies.

Cartesian Decentralization To create a multiagent manipulation system composed of N link agents, manipulator control must be decomposed into N independent processes. By examining the problem of end effector control it was shown that traditional inverse kinematic solution methods such as resolved motion position, rate and acceleration controllers (RMPC, RMRC, and RMAC respectively) were fundamentally centralized systems. However, Jacobian Transpose Control was shown to be a function solely of link and end effector coordinate frames. By assuming a distributed Newton Euler kinematic computation structure, JTC was shown to be cartesian decentralizable and a distributed manipulation architecture specified.

Multiagent Manipulator Control Using N autonomous link processes, this thesis demonstrated for the first time the parallel control of a N d.o.f. manipulator without estimation or reliance upon a centralized manipulator parameter model. In combination with a well established task space adaptive controller, the innovation of cartesian decentralization removes the robot's structural description from the goal generation system. *In effect, no manipulator model of any kind is required to derive setpoints for link agents to track a desired end effector trajectory.* Rather, the combination of "natural" manipulator dynamics, cartesian decentralization, and adaptive task space control are sufficient to determine the necessary control forces for end effector trajectory tracking.

Behaviours With each link agent acting autonomously to fulfill the end effector trajectory, it was shown how individual agents may act on local goals and "assert" global goals to the agent team. In selecting a simple PD joint space position controller as a local goal generator, new manipulator configuration policies were demonstrated including near minimal joint displacement trajectory tracking, fail safe trajectory tracking, retraction and variable compliance behaviours. A global goal assertion protocol enabled multiagent obstacle avoidance. Though some of these behaviours are well established (e.g. joint centering and obstacle avoidance), none have been decentralized over N processes.

Multiple Goal Interaction Auxiliary task tracking is not new to redundant manipulator control. However, these auxiliary tasks, traditionally developed and assigned to regions of configuration space through a centralized supervisory controller, were distributed over the multiagent manipulator control system without any centralized task assignment process. This study identified a number of significant properties of goal interaction that made this possible:

1. Compliance as Priority. Adaptive task space goals always suppress local compliant (PD) goal systems. Similarly, conflicting compliant systems may be biased through manipulation of PD gains in each system.
2. Null Space Self Organization. Adaptive global goal enforcement was shown to functionally *equivalent* to an explicit Jacobian null space task assignment. If local goals perturb the global goal system, adaptive global goal enforcement drives these goals automatically into the Jacobian null space.

Emergent Coordination The combination of global goal regulation and compliant local goals results in emergent manipulator configuration policy. By enforcing a global goal, local disturbances that propagate to the global behaviour are corrected and transmitted (as a correction) to the agent team. This local perturbation/global correction mechanism thus becomes a communication medium between agents.

1.4 Thesis Outline

To clarify agent and multiagent structures and permit an in depth dynamic analysis, chapter two introduces a standard plant by reviewing the kinematics, dynamics, and joint motor models of an arbitrary serial manipulation system. Chapter three then examines the real time control of these robots by examining the common supervisory centralized manipulator control strategies. This is contrasted by a brief overview of high performance single and multirobot real time supervisory controllers. Relating these agent-like controllers to traditional control theory, chapter four develops a theoretical foundation and introduces structural components to provide a consistent understructure for agent based control. The resulting agent and multiagent models are then applied to manipulator control and candidate global goals are considered. Chapter five documents the design and structure of a manipulator simulation platform, the Multiprocess Manipulator Simulator. Chapter six presents a detailed exploration of a global goal implementation and is followed in Chapter seven by results documenting several global, local and global goal combinations and identifies a mechanism for apparent emergent behaviour in combined systems. Chapter eight offers a comparison between centralized model based manipulator control and multiagent control, citing the advantages of simplicity, extensibility, and freedom from a manipulator parameter model and exploring the stability of multiple goal systems. Chapter nine summarizes the conclusions of the research and points out new directions possible through the techniques developed.

Chapter 2

The Manipulator Model

2.1 Introduction

While producing capable and robust robots, the investigation of agent and multiagent systems has been hampered by unique, underdocumented robot platforms, poorly understood agent and interagent dynamics, and inconsistent, confusing terminology. Despite the success of these systems, the variety of robots upon which agent and multiagent techniques are demonstrated greatly complicates the independent corroboration of results and the exploration of intra- and interagent dynamics. Terminology, hopefully clarified in chapter 4, further obscures this otherwise promising area of investigation. Therefore, this chapter introduces the dynamic structure of a well established 'benchmark' robotic platform, a serial manipulator, upon which multiagent control will be applied in subsequent chapters.

Compared to most agent or multiagent systems, typically one or more mobile robots, the selection of a manipulator robotic platform may seem unusual. After all, manipulator control is well established and few operate in time varying unstructured environments. These observations are certainly true. The kinematics and dynamics of manipulators are well understood with an overwhelming selection of capable industrial manipulator controllers. Yet, the same problems that once limited mobile robots to cautious laboratory excursions continue to limit manipulation systems to slowly changing, structured environment applications.

As discussed earlier, reliable motion in a changing, unknown environment requires close integration of real time sensing, modelling, planning, and control components. However, in assuming that configuration space positions, velocities, or forces are specified externally, the great majority of industrial manipulator control systems surrender sensing, modelling, and planning to other systems. In short, the attraction of a serial manipulator model as a foundation for multiagent control is threefold:

i Manipulator tasks are well formed, i.e. the movement of the end effector along a prescribed, collision free, task space trajectory.

ii Kinematics and dynamics of serial kinematic chains are well understood.

iii SMPA manipulator real time controllers for unstructured environments are complex, compute intensive, and scale poorly.

Manipulator trajectory tracking stands as an effective benchmark against which controllers of all kinds have been tested. By applying new controllers to this benchmark, results can be placed within a familiar, control theoretic context and compared against established methods. So the selection of a manipulator model as a basis for multiagent control can be justified as a well defined control problem that will profit from such methods.

This chapter will, therefore, briefly review the kinematics and dynamics of manipulators and introduce the joint motor model.

2.2 The Manipulator Model

The 'atomic' unit of the serial manipulator is the *link*, and, depending on the research objective, can have many characteristics such as flexible motor shafts and/or elastic material properties. For simplicity, this study will treat a link as a *rigid* body connecting two single degree of freedom revolute (rotating) or prismatic (linear) joints driven by some joint motor system. The *manipulator model* describes the dynamic characteristics of a kinematic chain of link/motor systems [Fu, 1987, de Silva, 1989].

2.2.1 The Kinematic Model

A *manipulator* is a serial kinematic chain composed of an arbitrary number of links driven by either prismatic or revolute joints. The *base* of the manipulator is fully constrained or *fixed* while the *end effector* is unconstrained or *free*. A manipulator with N such links may adopt a *configuration* described by a joint displacement vector $q \in \mathcal{Q} \subset \mathbf{R}^N$ composed of either angular or linear displacements, θ_i or d_i respectively.

The position of the link in cartesian space is described by an homogeneous transformation from an arbitrary world coordinate system to a *link frame*. A homogeneous transformation, $\mathbf{A}$, is composed of a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, a position vector $\mathbf{p} \in \mathbf{R}^3$ and a scaling vector $\mathbf{s} \in \mathbf{R}^4$ arranged in the following 4×4 matrix:

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{s}^T & \end{bmatrix} \tag{2.1}$$

Since the position of the ith link frame is a function of *superior* link frame positions, or those between the base link, 0, and the link, $i : 0 < i \leq N$, it is convenient to employ relative transformations between adjacent link frames: $\mathbf{A}_i^{i-1}$.

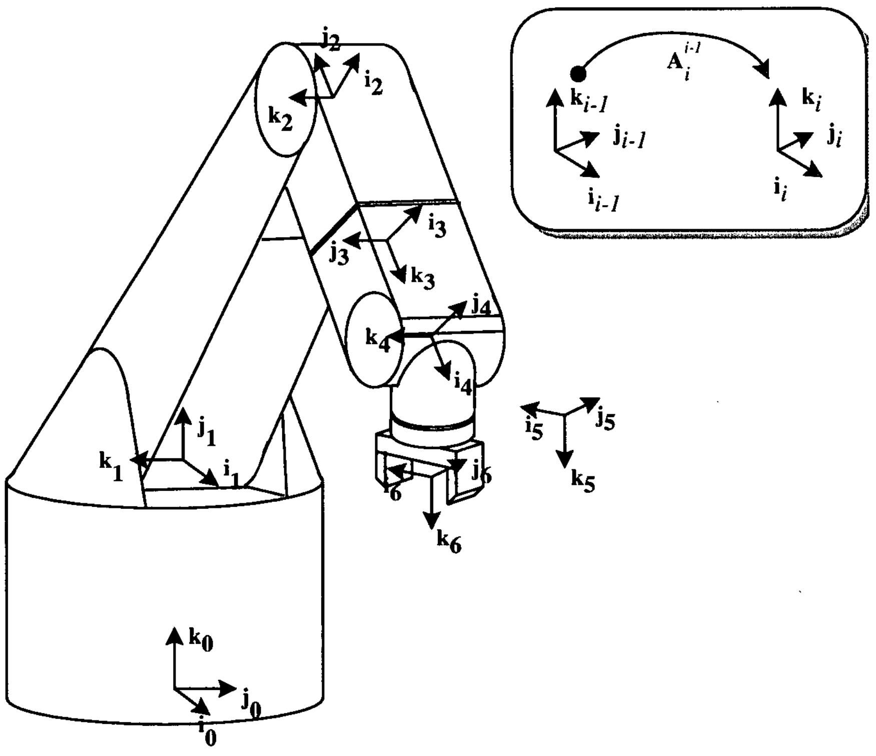


Figure 2.2: A 6 degree of freedom elbow manipulator. Note coordinate systems conform to Denavit Hartenberg conventions.

The transformation process is further simplified through the adoption of the widely used Denavit Hartenberg homogeneous transformation [Denavit, 1955] characterizing both revolute and prismatic joints with the standardized matrix expression:

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)a_i\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

given revolute displacements, θ_i, or prismatic displacements, d_i and the relative rotation, α_i, about the x axis between adjoining frames.

Conventions for the placement of the link frame vary, but most apply the link frame at the distal (inferior) joint. This differs with the convention in dynamic analysis, that places the frame at the link centroid.

The Forward Solution

Given a fixed *world* coordinate system at the base of the manipulator, the world position of the ith frame in the manipulator is the product of all the transformations from frame 0, the base coordinate frame, to the link:

$$\mathbf{A}_i = \Pi_{j=1}^{i} \mathbf{A}(q_i)_j^{j-1} \tag{2.3}$$

With a common abuse of notation, the world position and orientation of any point is presented as a *vector* in $\mathbf{R}^6$, though it is clearly a 4×4 matrix in the above expression. The world position of the end effector, $\mathbf{x}$, may be computed through *forward solution*, $\mathbf{f}(\mathbf{q})$, a function of the joint displacements, $\mathbf{q}$:

$$\mathbf{x} = \mathbf{f}(\mathbf{q}) = \Pi_{j=1}^{N} \mathbf{A}(q_i)_j^{j-1} \tag{2.4}$$

Thus the forward solution may be characterized as a map between *configuration space*, $\mathcal{Q} \subset \mathbf{R}^N$, and *task space*, $\mathcal{X} \subset \mathbf{R}^M$, or $\mathbf{f}(\mathbf{q}) : \mathcal{Q} \to \mathcal{X}$.

The Manipulator Jacobian

End effector velocity vector, $\dot{\mathbf{x}}$, may be computed through the chain rule:

$$\begin{aligned} \dot{\mathbf{x}} &= \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} \dot{\mathbf{q}} && (2.5) \\ &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} && (2.6) \\ \mathbf{J}(\mathbf{q}) &= \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} && (2.7) \end{aligned}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian of the forward solution. However, the rate of change of orientation (a matrix) complicates this expression and is better expressed through a position subJacobian $\mathbf{J}_p$ and orientation subJacobian $\mathbf{J}_o$ [Lewis, 1993]:

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{bmatrix} \mathbf{J}_p(\mathbf{q}) \\ \mathbf{J}_o(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} && (2.8) \\ \mathbf{J}_p(\mathbf{q}) &= \frac{\partial \mathbf{f}_p(\mathbf{q})}{\partial \mathbf{q}} && (2.9) \\ \mathbf{J}_o(\mathbf{q}) &= [\kappa_1 \mathbf{z}_0 \ldots \kappa_N \mathbf{z}_{N-1}] && (2.10) \end{aligned}$$

where κ_i is a selection operator (1 for revolute, 0 for prismatic) and $\mathbf{z}_{i-1}$ is the axis of rotation for the ith link in world coordinates.

A still more useful expression reveals the column-wise structure of a generic Jacobian [Spong, 1989]:

$$\mathbf{J}_i = \begin{cases} [\,\mathbf{k}_{i-1} \times (\mathbf{p}_n - \mathbf{p}_{i-1}) \quad \mathbf{k}_{i-1}\,]^T & \text{if revolute} \\ [\,\mathbf{0} \quad \mathbf{k}_{i-1}\,]^T & \text{if prismatic} \end{cases} \tag{2.11}$$

Just as in the forward solution, the Jacobian is a map $\mathbf{J}(\mathbf{q}) : \mathcal{Q} \rightarrow \mathcal{X}$. By reapplying the chain rule, end effector acceleration, $\ddot{\mathbf{x}}$ may be determined:

$$\ddot{\mathbf{x}}(t) = \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} \tag{2.12}$$

2.2.2 Manipulator Inertia Matrix

The *link* inertia matrix, $\mathbf{I}_i$, for link i in inertial coordinates is a 3×3 is a diagonal matrix of constant principal inertias:I_{x_i}, I_{y_i} and I_{z_i} of the ith link about the centroid. Assuming the principal axes are aligned with the coordinate frame and the Denavit-Hartenberg conventions are followed, I_{z_i} is related to J_{eff_i}, the effective moment of inertia about the actuator axis, through the parallel axis theorem while I_{x_i} and I_{y_i} are a function of link geometry. The inertia matrix of the ith link about its centroid in *world* coordinates, $\mathbf{D}_i$ is therefore:

$$\mathbf{D}_i = \begin{bmatrix} m_i & 0 \\ 0 & \mathbf{R}_i\mathbf{I}_i\mathbf{R}_i^T \end{bmatrix} \tag{2.13}$$

where m_i is the mass of the link and $\mathbf{R}_i$ is the orientation of the link frame with respect to the world coordinate system.

The *manipulator* inertia matrix, $\mathbf{D}(\mathrm{q})$, is an expression of the inertia of the kinematic chain expressed in world coordinates. The matrix is a function of the mass and the link inertias, $\mathbf{D}_i$, of the N links. From the principle of conservation of energy:

$$\begin{aligned} \dot{\mathbf{q}}^T\mathbf{D}(\mathbf{q})\dot{\mathbf{q}} &= \sum_{i=1}^{N} \dot{\mathbf{x}}_{c_i}^T \mathbf{D}_i \dot{\mathbf{x}}_{c_i} && (2.14) \\ &= \sum_{i=1}^{N} \dot{\mathbf{q}}^T \mathbf{J}_{c_i}^T \mathbf{D}_i \mathbf{J}_{c_i} \dot{\mathbf{q}} && (2.15) \\ \mathbf{D}(\mathbf{q}) &= \sum_{i=1}^{n} \mathbf{J}_{c_i}^T \mathbf{D}_i \mathbf{J}_{c_i} && (2.16) \end{aligned}$$

where $\mathbf{J}_{c_i}$ is the $6 \times N$ Jacobian of the ith centroid, the subscript c_i indicates the centroid of the ith link and $\mathbf{D}_i$ is the ith inertia matrix in world coordinates.

2.2.3 Manipulator Dynamics

As stated earlier, the manipulator dynamics equations of motion are usually formed through either Newton-Euler methods, a recursive link by link application of Newtons Laws, or Lagrange methods in which potential and kinematic energy is conserved within the kinematic chain. Developing the manipulator Lagrangian equations [Spong, 1989, Goldstein, 1981], the *Lagrangian*, L, is:

$$\begin{aligned} L &= K - V && (2.17)\\ &= \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{D}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{q}) && (2.18)\\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{d}_{ij}\dot{q}_i\dot{q}_j - V(\mathbf{q}) && (2.19)\end{aligned}$$

where K is *kinetic energy*, and $V(\mathbf{q})$ is *potential energy* and d_{ij} is the ijth element of $\mathbf{D}(q)$. The Lagrangian equations:

$$\tau_k = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} \tag{2.20}$$

Computing the components of this equation

$$\begin{aligned} \frac{\partial L}{\partial \dot{q}_k} &= \sum_{j=1}^{n} \mathrm{d}_{kj}\dot{q}_j && (2.21)\\ \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} &= \sum_{j=1}^{n} \mathrm{d}_{kj}\ddot{q}_j + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \mathrm{d}_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j && (2.22)\\ \frac{\partial L}{\partial q_k} &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \mathrm{d}_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j - \frac{\partial V(\mathbf{q})}{\partial q_k} && (2.23)\end{aligned}$$

Thus the LE equations can be rewritten:

$$\begin{aligned} \tau_k &= \sum_{j}^{n} \mathrm{d}_{kj}\ddot{q}_j + \sum_{i,j}^{n}\frac{\partial \mathrm{d}_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j - \frac{1}{2}\sum_{i,j}^{n}\frac{\partial \mathrm{d}_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j + \frac{\partial V(\mathbf{q})}{\partial q_k} && (2.24)\\ &= \sum_{j}^{n} \mathrm{d}_{kj}\ddot{q}_j + \sum_{i,j}^{n}\left[\frac{\partial \mathrm{d}_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial \mathrm{d}_{ij}}{\partial q_k}\right]\dot{q}_i\dot{q}_j + \frac{\partial V(\mathbf{q})}{\partial q_k} && (2.25)\end{aligned}$$

realizing that through reorganization the term $\sum_{i,j}^{n}\frac{\partial \mathrm{d}_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j$ becomes:

$$\sum_{i,j}^{n}\frac{\partial \mathrm{d}_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j = \frac{1}{2}\sum_{i,j}^{n}\frac{\partial \mathrm{d}_{kj}}{\partial q_i} + \frac{\partial \mathrm{d}_{ki}}{\partial q_j} \tag{2.26}$$

which when substituted into the second term of 2.25 becomes:

$$\sum_{i,j}^{n}\left[\frac{\partial \mathrm{d}_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial \mathrm{d}_{ij}}{\partial q_k}\right]\dot{q}_i\dot{q}_j = \frac{1}{2}\sum_{i,j}^{n}\left[\frac{\partial \mathrm{d}_{kj}}{\partial q_i} + \frac{\partial \mathrm{d}_{ki}}{\partial q_j} - \frac{\partial \mathrm{d}_{ij}}{\partial q_k}\right]\dot{q}_i\dot{q}_j \tag{2.27}$$

This term produces the *Christoffel* symbols:

$$C_{ijk}(q) = \frac{1}{2}\left[\frac{\partial \mathrm{d}_{kj}}{\partial q_i} + \frac{\partial \mathrm{d}_{ki}}{\partial q_j} - \frac{\partial \mathrm{d}_{ij}}{\partial q_k}\right]\dot{q}_i\dot{q}_j \tag{2.28}$$

The last term of equation (2.25), the change in potential energy, is often expressed as:

$$\phi_k(\mathbf{q}) = \frac{\partial V(\mathbf{q})}{\partial q_k} \tag{2.29}$$

the LE equations finally becoming:

$$\tau_k = \sum_j^n \mathrm{d}_{kj}\ddot{q}_j + \sum_{i,j}^n C_{ijk}(\mathbf{q})\dot{q}_i\dot{q}_j + \phi_k(\mathbf{q}) \tag{2.30}$$

where $\tau_k \in \mathcal{Q}$.In a gravitational potential field $\phi_k(\mathbf{q}) = \mathbf{g}(\mathbf{q})$. However if the end effector applies an external force, $\mathbf{f}_{\mathrm{ext}} \in \mathcal{X}$, at the end effector then the work done:

$$V(\mathbf{q}) = \delta\mathbf{x}^T\mathbf{f}_{\mathrm{ext}} = \delta\mathbf{q}^T\mathbf{J}(\mathbf{q})^T\mathbf{f}_{\mathrm{ext}} \tag{2.31}$$

and $\phi_k(\mathbf{q})$ attributed to this work:

$$\frac{\partial V(\mathbf{q})}{\partial q_k} = \left[\frac{\partial \mathbf{f}(\mathbf{q})}{\partial q_k}\right]^T \mathbf{f}_{\mathrm{ext}} \tag{2.32}$$

$$\tau_{\mathrm{force}} = \mathbf{J}_k^T(\mathbf{q})\mathbf{f}_{\mathrm{ext}} \tag{2.33}$$

where the $\mathbf{J}_k^T(\mathbf{q})$ is the kth row of the Jacobian Transpose. The manipulator joint forces expressed in matrix form:

$$\tau = \mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \mathbf{J}^T\mathbf{f}_{\mathrm{ext}} \tag{2.34}$$

where the k, j element of $\mathbf{C}(q, \dot{q})$ is

$$\mathrm{C}_{kj} = \frac{1}{2}\sum_{i=1}^n \left[\frac{\partial \mathrm{d}_{kj}}{\partial q_i} + \frac{\partial \mathrm{d}_{ki}}{\partial q_j} - \frac{\partial \mathrm{d}_{ij}}{\partial q_k}\right]\dot{q}_i \tag{2.35}$$

Through appropriate control of joint forces, τ, a manipulator can execute a desired task at the end effector.

2.2.4 Properties of The Manipulator Jacobian

Since the properties of the Jacobian are frequently cited throughout this text, it is important to establish a clear understanding of the concepts surrounding Jacobian null and range spaces. Referring to figure 2.3, recall that the instantaneous domain of the Configuration space is the volume that bounds all possible joint

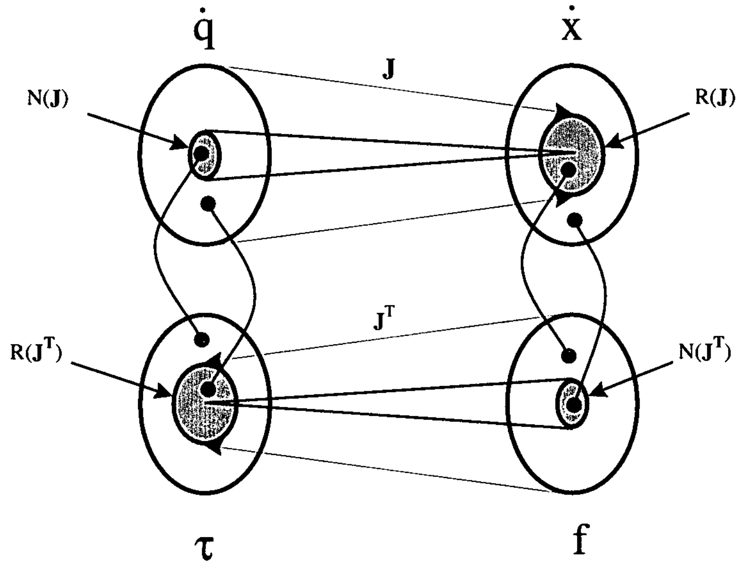


Figure 2.3: The Range and Null spaces of the Jacobian and Jacobian Transpose.

velocities, $\dot{\mathbf{q}} \in \mathcal{Q}$, within a particular configuration, $\mathbf{q} \in \mathcal{Q}$. The range of the Jacobian, $R(\mathbf{J})$, is the space occupied by all end effector velocities $\dot{\mathbf{x}} \in \mathcal{X}$ produced through joint motion. The null space of the Jacobian, $N(\mathbf{J})$, is the space occupied by all joint velocities $\dot{\mathbf{q}}$ for which no end effector motion results.

Conversely the range of the Jacobian *transpose*, $\mathbf{R}(\mathbf{J}^T)$, is the space within which all joint forces must be applied to balance an end effector force. While the null space of the Jacobian transpose, $\mathbf{N}(\mathbf{J}^T)$, is the region of configuration space for which no joint forces are required to balance an end effector force. Now the *duality* [Asada, 1986] of manipulator kinematics and statics means that the, $\mathbf{R}(\mathbf{J}^T)$, and $\mathbf{N}(\mathbf{J})$ are orthogonal complements or $\mathbf{R}(\mathbf{J}^T) \cup \mathbf{N}(\mathbf{J}) = \mathcal{Q}$.

2.2.5 The Motor Model

Each link is driven by either a linear or revolute powered mechanism. A popular model for direct drive joint motors is the armature controlled dc servomotor [Fu, 1987]. In this model, the servomotor dynamics are described by:

$$\tau_{m_k} = J_{m_k}\ddot{q}_{m_k} + B_{m_k}\dot{q}_{m_k} \tag{2.36}$$

where J_{m_k} is the motor shaft inertia and B_{m_k} is a viscous force coefficient . An identical expression governs the link itself:

$$\tau_{l_k} = J_{l_k}\ddot{q}_{l_k} + B_{l_k}\dot{q}_{l_k} \tag{2.37}$$

The torque required to drive the motor-link system is simply the sum:

$$\tau_k = \tau_{m_k} + \tau^*_{l_k} \tag{2.38}$$

where $\tau^*_{l_k}$ is the torque applied by the link via the gear train on the motor shaft. Given the *gear train ratio*:

$$r_k = \frac{N_m}{N_l} \tag{2.39}$$

where N_m and N_l are (for example) the number of teeth on each gear(see figure 2.4a), τ_{l_k} can be rewritten in terms of q_{m_k} producing the expression:

$$\tau^*_{l_k} = r_k^2(J_{l_k}\ddot{q}_{m_k} + B_{l_k}\dot{q}_{m_k}) \tag{2.40}$$

For the total torque:

$$\begin{aligned} \tau_k &= (J_{m_k} + r_k^2 J_{l_k})\ddot{q}_{m_k} + (B_{m_k} + r_k^2 B_{l_k})\dot{q}_{m_k} && (2.41)\\ &= J_{\text{eff}_k}\ddot{q}_{m_k} + B_{\text{eff}_k}\dot{q}_{m_k} && (2.42)\end{aligned}$$

Small gear ratios r_k tends to isolate the motor from the kth link's dynamics.

Now it is common to assume that the motor torque is directly proportional to armature current, i_{a_k}:

$$\tau_k = K_{a_k} i_{a_k} \tag{2.43}$$

Applying Kirchoff's law to the motor circuit in figure 2.4a:

$$V_{a_k}(t) = R_{a_k} i_{a_k}(t) + L_{a_k}\frac{di_{a_k}(t)}{dt} + e_{b_k}(t) \tag{2.44}$$

The back electromotive force, $e_{b_k}(t)$, is proportional to the motor's angular shaft velocity.

$$e_{b_k}(t) = K_{b_k}\dot{q}_{m_k} \tag{2.45}$$

Taking the laplace transform of equations (2.42), (2.43), (2.44), (2.45) a transfer function relating shaft position to motor voltage can be developed [Fu, 1987].

$$\frac{Q_{m_k}(s)}{V_{a_k}(s)} = \frac{K_{a_k}}{s[s^2 J_{\text{eff}_k} L_{a_k} + (B_{\text{eff}_k} L_{a_k} + R_{a_k} J_{\text{eff}_k})s + R_{a_k} B_{\text{eff}_k} + K_{a_k} K_{b_k}]} \tag{2.46}$$

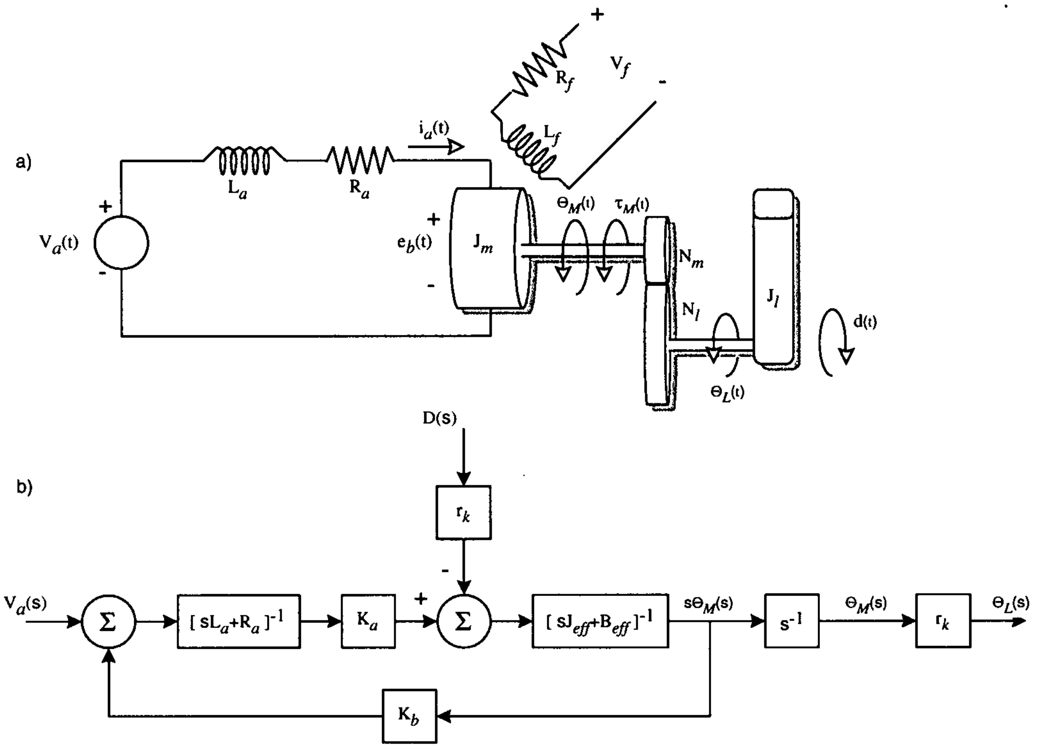


Figure 2.4: An armature controlled dc drive joint motor and gear train model.

Since the mechanical time constants are generally far larger than the electrical time constants, the motor inductance L_{a_k} is usually ignored.Thus the servo control block diagram in figure 2.4b, is usually reduced to the transfer function [Fu, 1987]:

$$\frac{Q_{m_k}(s)}{V_{a_k}(s)} = \frac{K_k}{[s^2 J_{\text{eff}_k} + sB_{\text{eff}_k} + K_k K_{b_k}]} \tag{2.47}$$

where $K_k = K_a/R_a$. Within a manipulator, the link does not act in isolation. Unmodelled ***disturbances*** d_k include off-axis inertial and coriolis/centrifugal forces (i.e. imposed by other links on link k) or from the manipulator dynamics:

$$d_k = \sum_{j \neq k} d_{jk}\ddot{q}_j + \sum_{i,j} C_{ijk}\dot{q}_i\dot{q}_j \tag{2.48}$$

With these link disturbances included, the link motor dynamic equation becomes:

$$J_{\text{eff}_k}\ddot{q}_{m_k} + (B_{\text{eff}_k} + K_k K_{b_k})\dot{q}_{m_k} = K_k u_k - r_k d_k(t) \tag{2.49}$$

where $u_k = V_{a_k}(t)$, the command voltage. The state space equation for the kth link:

$$\mathbf{x}_k = \mathbf{A}_k\mathbf{x}_k + \mathbf{b}u_k + \mathbf{d}\mathbf{v} \tag{2.50}$$

$$\mathbf{x}_k = \begin{bmatrix} q_{m_k} \\ \dot{q}_{m_k} \end{bmatrix} \quad \mathbf{A}_k = \begin{bmatrix} 0 & I \\ 0 & -J_{\text{eff}_k}^{-1}(B_{\text{eff}_k} + K_k K_{b_k}) \end{bmatrix} \tag{2.51}$$

$$\mathbf{b}_k = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{u}_k = J_{\text{eff}_k}^{-1} K_k u \tag{2.52}$$

$$\mathbf{d}_k = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{v}_k = -J_{\text{eff}_k}^{-1} r_k d_k \tag{2.53}$$

It is not surprising that given the relative simplicity of link servo control and the dynamic regime of industrial robots, most manipulator controllers ignore link dynamics entirely. Nonlinear coupling and manipulator inertia are significant only during high velocity or acceleration, commonly found in low precision maneuvers in industrial applications. High precision maneuvers are generally performed at low velocity and with minimal acceleration. Therefore, pure PD control without dynamic compensation (the cancellation of d_k) is adequate for the majority of industrial manipulator tasks. However, high performance manipulation (e.g. high speed, precise positioning), redundant manipulation, and unstructured environments (UEs) challenge such joint space robot control methods as explained in the following chapter.

Chapter 3

Real Time Robot Control

3.1 Introduction

This chapter will explore the relationship between the inverse problem, the specification of robot action based on a desired robot behaviour, and real time control. In so doing, the argument will be proposed that real time robot *controller* design and robotic *system* design are closely related problems. Though directed at manipulation robotics in particular, much of this chapter is applicable to robotics in general.

As mentioned in the last chapter, the primary objective of most manipulation systems is to track an end effector trajectory in *task space*. Since control of the end effector is performed indirectly through the control of joint motor controllers, a desired task space trajectory must first be transformed into configuration space coordinates before it can be realized at the end effector. This transformation must invert the relationship between configuration space and task space, a process complicated by the nonlinear structure of the forward kinematic solution.

For typical manipulators, the transformation between task and configuration spaces is usually a geometric inversion of the forward kinematic solution, the *inverse kinematic solution*, and often considered separate from the control problem. However, if the number of degrees of freedom available to the manipulator, N, exceed the number required to accomplish the task, M, the manipulator becomes *redundant*, capable of additional simultaneous tasks. Unfortunately, as the *degree of redundancy*, $N - M$, grows, geometric inverse solution strategies become increasingly complex. In general, the solution embeds the inverse solution within the manipulator controller.

To explore these issues further, this chapter adopts a task formalism to discuss inversion strategies and show that, in general, model based inverse solution methods influence the structure of redundant manipulation *systems*. Finally a discussion of general robot control *architectures* will show that idealized, single threaded, SMPA robot control is rarely used in practical real time systems and that robot architectures are evolving towards pragmatic multithreaded reactive designs: an *agent architecture*.

3.2 Task Functions

The colloquial definition of the term task, much like the term *goal*, is simply a desired action. However, in robotics, tasks are more precisely defined in terms of one or more trajectories in some *task space* that, with the application of appropriate tools, accomplish some change in the environment. Typically, tasks reside in task space, $\mathbf{R}^M$, often (but not limited to) a subset of cartesian space, $\mathbf{R}^6$. Though a number of task specification techniques exist, the objective of task generation is the construction of a desired task space trajectory, $\mathbf{x}_d(t)$ and velocity $\dot{\mathbf{x}}_d(t)$.

Samson et al.[Samson, 1991] further formalize this specification within Task Functions. Briefly, a task function is the error between the desired task space coordinate and the forward solution over a time interval $[0, T]$ or:

$$\mathbf{x}_e(\mathbf{q}, t) = \mathbf{x}_d(t) - \mathbf{f}(\mathbf{q}) \tag{3.54}$$

Samson et al. further define the *feasibility of a task* by:

$$\mathbf{x}_e(\mathbf{q}, t) = 0 \quad \forall t \in [0, T] \tag{3.55}$$

By recognizing that the specification of *configuration space* positions based on a *task space* coordinate requires the inversion of the *forward* solution, Samson applies the inverse function theorem [Vidyasagar, 1993] to characterize inverse solutions. Reiterating the inverse function theorem:

Theorem 1 (Inverse Function Theorem) *Suppose* $\mathbf{f} : \mathbf{R}^n \to \mathbf{R}^n$ *is* C^1 *at* $\mathbf{x}_0 \in \mathbf{R}^n$ *and let* $\mathbf{y}_0 = \mathbf{f}(\mathbf{x}_0)$. *Suppose* $\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}_0}$ *is nonsingular. Then there exist open sets* $U \subseteq \mathbf{R}^n$ *containing* $\mathbf{x}_0$ *and* $V \subseteq \mathbf{R}^n$ *containing* $\mathbf{y}_0$ *such that* $\mathbf{f}$ *is a diffeomorphism* [1] *of* U *onto* V. *If, in addition,* $\mathbf{f}$ *is smooth and* $\mathbf{f}^{-1}$ *is smooth then* $\mathbf{f}$ *is a smooth diffeomorphism*

This theorem was then used by Samson to describe three factors influence task *feasibility*:

i *A unique solution exists.* If $\frac{\partial \mathbf{x}_e(\mathbf{q},t)}{\partial \mathbf{q}}$ is invertible [2] a position $\mathbf{x}_d$ can be mapped to a unique position $\mathbf{q}_d$. This task is feasible, since a unique inverse kinematic solutions exists that will map $\mathbf{x}_d$ onto $\mathbf{q}_d$.

ii *Infinite solutions exist.* If $\frac{\partial \mathbf{x}_e(\mathbf{q},t)}{\partial \mathbf{q}}$ is *not* invertible but *onto* [3] , a position $\mathbf{x}_d$ can be mapped to a *region* of dimension corank$(\frac{\partial \mathbf{x}_e(\mathbf{q},t)}{\partial \mathbf{q}})$. In effect an infinite number of positions $\mathbf{q}_d$ exist. The

[1] A differentiable mapping with a differentiable inverse.
[2] or *injective*: For every member in the range there is a unique member in the domain.
[3] or *surjective*. Define the *codomain* of a function as the simply connected region that bounds the values of the function and

manipulator has fewer constraints, m, than degrees of freedom, n. The manipulator is said to possess $r = n - m$ redundant degrees of freedom. This case is feasible if the redundant degrees of freedom are somehow constrained to determine a unique solution.

iii *No solution exists.* If $\frac{\partial \mathbf{x}_e(\mathbf{q},t)}{\partial \mathbf{q}}$ is neither invertible nor onto. This is the singular case and, by the Inverse Function Theorem no inverse, $\mathbf{g}(\mathbf{x}_d(t))$, exists.

So task feasibility is a measure of whether an inverse solution exists. If a task is *feasible* and $\frac{\partial \mathbf{x}_e}{\partial \mathbf{q}}$ is either invertible or simply onto, some form of *inverse kinematic solution* $\mathbf{g} : \mathcal{X} \to \mathcal{Q}$ can be constructed that will map desired task space coordinates into configuration space or

$$\mathbf{q}_d(t) = \mathbf{g}(\mathbf{x}_d(t)) \tag{3.56}$$

The next section will examine four methods of inverse solution.

3.3 Manipulator Control and the Inverse Solution

Given the feasibility of a task, four basic inversion methods are used to map desired task space coordinates into configuration space through three classes of inverse function: geometric inverse, integrable inverse, and projection inverse solutions. Geometric solutions map task space positions directly to configuration space positions and here will be referred to as Resolved Motion Position Control. Integrable solutions include Resolved Motion Rate and Acceleration Control, both of which map task space to configuration space derivatives of position. Finally, Jacobian Transpose Control projects task forces onto configuration space generalized force axes.

3.3.1 Resolved Motion Position Control

The most common form of inverse kinematic solution is the inverse geometric solution. The analytical inverse map:

$$\mathbf{q}_d(t) = \mathbf{g}(\mathbf{x}_d(t)) \tag{3.57}$$

is derived off-line, enabling fast on-line determination of $\mathbf{q}_d$. However, as the forward solution is usually a nonlinear product of transcendental functions, this off-line inversion process requires both mathematical and

the set of values itself is the *range* of the function. Then the *surjective* function, $\frac{\partial \mathbf{x}_e}{\partial \mathbf{q}}$, associates two sets, $\mathcal{X}_e$ and $\mathcal{Q}$, such that every member, $\dot{\mathbf{x}}_e$, of the *codomain*, $\mathcal{X}_e$, is the image of at least one member, $\dot{\mathbf{q}}$, of the *domain*, $\mathcal{Q}$, though there may be members of the domain that are not mapped to the codomain (i.e. some values of $\dot{\mathbf{q}}$ may not map to any $\dot{\mathbf{x}}_e$). Thus the *range* of the surjective mapping $R(\frac{\partial \mathbf{x}_e}{\partial \mathbf{q}})$ is the entire codomain, $\mathcal{X}_e$.

geometric insight to rapidly achieve an efficient solution [Paul, 1981, Fu, 1987] and is unique to a particular manipulator design. Indeed, inverse solution functions are rarely simple mathematical procedures and usually include considerable flow control logic to achieve physically realizable solutions. Given the complexity of manipulator geometry, errors in inverse kinematic solutions are common and thorough testing of these solutions throughout the workspace is mandatory prior to installation.

Once established, the desired configuration space vector, $\mathbf{q}_d$, is passed to the manipulator's joint control processes. The majority of manipulator control techniques rely on such setpoint specification. In particular, industrial manipulator control often applies basic PD control to the setpoints q_{d_k}:

$$u_k = k_q q_{e_k} + k_w \dot{q}_{e_k} \tag{3.58}$$

where k_q and k_w are position and velocity gains, respectively, and $q_{e_k} = q_{d_k} - q_{m_k}$ is the joint error for the kth link.

Redundancy Resolution

As mentioned earlier, there exist tasks for which either an exact inverse kinematic solution exists, an infinite number of solutions exist, or no solutions exist. When an exact inverse kinematic solution exists, the manipulator is *fully constrained* and any inverse kinematic solution method will produce unique solutions. However, most realistic end effector trajectories enter regions of task space for which the manipulator becomes redundant, possessing degrees of freedom in excess of those required to accomplish the task. Technically, joint space becomes divided between two regions, the range and the null space of the manipulator Jacobian, $\mathbf{R}(\mathbf{J})$ and $\mathbf{N}(\mathbf{J})$ respectively.

A manipulator becomes redundant when an *infinite number of solutions form a dense set in joint space*[Samson, 1991]. While a 2R planar manipulator may adopt one of two configurations for any point in planar cartesian space, it is not possible to move on a continuous path in configuration space from one solution to the other while maintaining the desired end effector position. Thus the solution space for this manipulator is bifurcated but not redundant. In contrast, a 3R planar manipulator can adopt a continuous path through a subspace of Q while maintaining a planar end effector position. While these 'extra' degrees of freedom permit a wide selection of configurations and therefore flexibility, determining inverse solutions for such systems can be complex.

The simple answer to finding a unique inverse solution for an underconstrained manipulator is to add constraints to $N(\mathbf{J})$. By fully constraining a redundant manipulator through heuristics or task space augmentation, a unique inverse kinematic solution may be generated. In RMPC a geometric solution of a redundant manipulator requires the adoption of joint displacement selection rules or heuristics. Another approach is to *augment* the forward solution with additional or secondary tasks that constrain the manipulator's unconstrained degrees of freedom. Redefining the task space as:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{f}_e(\mathbf{q}) \\ \mathbf{f}_c(\mathbf{q}) \end{bmatrix} \tag{3.59}$$

where $\mathbf{f}_c(\mathbf{q}) : \mathbf{q} \in \mathcal{Q}_m$ and $\mathbf{f}_c(\mathbf{q}) : \mathbf{q} \in \mathcal{Q}_r$ are the end effector and additional constraint forward solutions respectively. With an appropriately augmented task space an inverse solution can again become one-to-one. Despite this useful technique, geometric inversion remains a complicated process. Therefore, redundant systems are usually controlled through numerical inverse methods such as either resolved motion rate control (RMRC) or resolved motion acceleration control (RMAC) techniques.

Configuration space position setpoints are a double edged sword. Though the determination of $\mathbf{q}_d$ provides a clear specification of both the joint and end effector trajectories under stable control, the inversion process becomes increasingly complex with robot geometry. Thus RMPC provides an easy method of inversion for simple robot geometries combined with the security of predetermined trajectories. Conversely, RMPC becomes very complex for redundant manipulation – precisely when the robot exhibits the most versatility.

3.3.2 Resolved Motion Rate Control

An alternative to the geometric inverse kinematic solutions for position $\mathbf{q}$, is to solve an inverse kinematic solution for joint rates $\dot{\mathbf{q}}$.

$$\dot{\mathbf{q}}(t) = \mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{x}}(t) \tag{3.60}$$

First introduced by Whitney [Whitney, 1969], Resolved Motion Rate Control, RMRC, exploits the relationship between cartesian and joint space velocities in equation (2.6).

Since RMRC is a velocity controller, the joint controller is of the form:

$$\begin{aligned} u_k &= k_w \dot{q}_{e_k} && (3.61) \\ u_k &= k_w \mathbf{J}_k^{-1}(\mathbf{q})\dot{\mathbf{x}}_e(t) && (3.62) \end{aligned}$$

where $\mathbf{J}_k^{-1}$ is kth row the Jacobian inverse.

Unfortunately, this method is prone to numerical instability near Jacobian singularities. These usually occur near the extremities of a manipulator's work space or in underconstrained, redundant, regions of configuration space.

Redundancy Resolution

Whitney [Whitney, 1969] recognized these hazards and recommended the use of the pseudoinverse for redundant systems:

$$\dot{\mathbf{q}}(t) = \mathbf{J}^{\dagger}\dot{\mathbf{x}}(t) \tag{3.63}$$

to solve for joint angles. Defining the pseudoinverse as:

$$\mathbf{J}^{\dagger}(t) = \mathbf{J}(\mathbf{J}^{T}\mathbf{J})^{-1} \tag{3.64}$$

produces a least squares velocity solution. Additional constraints can then be inserted into $N(\mathbf{J})$ using the well established technique:

$$\dot{\mathbf{q}}(t) = \mathbf{J}^{\dagger}\dot{\mathbf{x}}(t) + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{h} \tag{3.65}$$

where $(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})$ selects the Jacobian null space and $\mathbf{h}$ is an arbitrary vector inserted into $N(\mathbf{J})$ based on some secondary optimization criteria (e.g. [Hollerbach, 1987, Nakamura, 1984, Nakamura, 1991, Kazerounian, 1988, Sung, 1996, Whitney, 1969]).

3.3.3 Resolved Motion Acceleration Control

Similarly Resolved Motion Acceleration Control, RMAC, exploits the relationship in equation (2.12):

$$\ddot{\mathbf{q}}(t) = \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t)\right] \tag{3.66}$$

With a prescribed twice differentiable task space trajectory, $\mathbf{x}_d(t)$, an acceleration control law may be devised [Luh, 1980b]:

$$\begin{aligned} \ddot{\mathbf{q}}_d(t) &= \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}_d(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v\dot{\mathbf{x}}_e + k_p\mathbf{x}_e\right] && (3.67) \\ &= -k_v\dot{\mathbf{q}} + \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v\dot{\mathbf{x}}_d + k_p\mathbf{x}_e\right] && (3.68) \end{aligned}$$

clearly the equation 3.68 is only possible if k_p and k_v are *scalars*.

From (2.34), the acceleration vector computed in equation (3.67) can be substituted into a feedforward dynamic model to provide the necessary control effort for each actuator:

$$\mathbf{u} = \hat{\mathbf{D}}(\mathbf{q})\ddot{\mathbf{q}}_d + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}) \tag{3.69}$$

Since RMAC relies on the Jacobian inverse, it, too, becomes numerically unstable near Jacobian singularities.

Redundancy Resolution

Differentiating equation (3.65) produces the general solution for joint accelerations in a redundant manipulator :

$$\ddot{\mathbf{q}}(t) = \mathbf{J}^{\dagger}\left[\ddot{\mathbf{x}}(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t)\right] + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})(\mathbf{J}^{\dagger}\dot{\mathbf{x}}(t) - \mathbf{J}^{\dagger}\mathbf{J}\mathbf{h} + \dot{\mathbf{h}}) \tag{3.70}$$

where the second term again inserts secondary tasks into the null space.Though setting $\mathbf{h} = \dot{\mathbf{h}} = \mathbf{0}$ in equation (3.65) results in a least squares joint velocity solution, clearly it does not necessarily produce a least squares acceleration solution in equation (3.70). Indeed [Kazerounian, 1988] shows that ignoring the second term of (3.70), locally minimizing joint *accelerations*, globally minimizes *joint velocities.*

Since the computation of both the dynamic model, equation (3.69), and the pseudoinverse [Press, 1988] can be a slow process, the computing resources required to exploit redundant manipulators in real time position control are significant.

3.3.4 Jacobian Transpose Control, JTC

Now, the setpoint controllers in equations (3.58),(3.62), and (3.67) ultimately specify a torque setpoint related through some map to task space position. So PD, RMRC, or RMAC controllers first invert the forward solution and then apply an error regulator to ultimately generate the necessary joint forces to move the end effector along the desired trajectory. Though effective, the control of end effector trajectories through inversion, joint space error regulation, and finally joint space force application is somewhat indirect. A more immediate method would be to regulate end effector forces directly through joint space forces and is exactly the mechanism behind Jacobian Transpose Control (JTC).

Unlike joint space methods, Jacobian Transpose Control applies torques to the manipulator by projecting a task-dependent *end effector* force trajectory onto each joint through the following equation

$$u_k = \mathbf{J}_k^T\, \mathbf{f}_d(\ddot{\mathbf{x}}_d(t), \mathbf{x}_e(t), \dot{\mathbf{x}}_e(t)) \tag{3.71}$$

where u_k and $\mathbf{f}(\cdot)$ are generalized forces in configuration and cartesian space respectively. Common in force control (e.g. [Raibert, 1981, Fisher, 1992]), JTC is less well known as an on-line position control scheme [Wu, 1982, Hogan, 1985c, Khatib, 1987, Seraji, 1987b] and off-line inverse kinematic solution [Asada, 1986, Slotine, 1988]schemes. In general, JTC is less compute intensive than " Newtons method" numerical solutions (i.e. RMRC and RMAC) but is slower to converge [Slotine, 1988]. Compute intensive disturbance compensation (i.e. through feedforward dynamics or adaptive control) must also be added to JTC to achieve precision position control [Khatib, 1985].

Redundancy Resolution

The Jacobian Transpose solution does not suffer numerical instability near Jacobian singularities as do solutions based on the Jacobian inverse. Since the required end effector force is projected onto each actuator, redundant manipulators are treated no differently than fully constrained manipulators. However, *kinematic* singularities, in which desired end effector forces have no projection on any of the actuators, produces no response in the manipulator. Such singularities occur near the extremities of the robot work space and can be avoided through simple techniques [Slotine, 1988].

3.4 Real Time Supervisory Control Architectures

Control represents one of four phases in the sense-model-plan-act (SMPA) cycle that persists as the context for 'intelligent' robotic systems. Of the four, the symbolic model-plan portion of the cycle has traditionally been considered the crux of the AI problem. The sensing and action phases of the cycle are generally considered services to this 'intelligent' symbolic system. As reviewed above, manipulation systems use symbolic engines that often include model driven inverse kinematics solutions (i.e. centralized RMPC, RMRC, RMAC, and JTC solutions) - the output of which is a deterministic trajectory in joint space. As a consequence, the traditional responsibility of manipulator control is to guarantee exact joint space trajectory tracking.

Contrary to early optimistic assessments of the problem, many now believe that autonomous robot performance is limited not by inadequate symbolic reasoning but by impoverished sensor fusion and control. Initially, investigators assumed that sensor fusion would eventually yield simple methods for detection and modelling of the workspace [4]. However, building and maintaining a sufficiently capable sensor fusion system for autonomous operation has proven to be complex, often unreliable and, in general, prohibitively expensive. Hence the interest in alternative control architectures that do not rely on extensive signal processing, modelling, and symbolic reasoning to achieve real time autonomous performance.

[4]A view traceable from early systems such PLANNER and Winograd's SHRDLU [Winograd, 1972] to existing task specification/assembly modelling languages such as RAPT [Ambler, 1975, Ambler, 1980, Thomas, 1988]

3.4.1 Manipulation

Real time manipulation has many aspects, such as real time navigation and force control, that often place multiple demands on manipulator performance in a changing workspace. Since it is possible that multiple, unpredictable, constraints may arise over the course of a single task, redundant robots are more attractive than traditional fully constrained manipulators in an unstructured environment. To leverage this flexibility, a growing number of redundant robotic systems incorporate additional sensing into RMRC, RMAC or JTC controllers. These sensors engage task specific auxiliary controllers through switching logic and/or provide data for environmental servos. In effect embedding limited modelling and planning portion into the control system. Some examples from manipulation and mobile robotics follow.

Hybrid Position/Force Control

Manipulation often requires not only simple end effector positioning but also force control, for example in machining tasks. Hybrid position/force control, in figure 3.4.1, [Raibert, 1981, Anderson, 1988, Fisher, 1992] employs a switching architecture to select appropriate control techniques for either position or force control. By incorporating end effector force sensors to control the manipulator's control effort, force controllers can regulate end effector forces in real time. Hogan formalized this concept further within impedance control [Hogan, 1985a]. From [Fisher, 1992] the equations for (differential) position and force control respectively:

$$\mathbf{q}_c = (\mathbf{SJ})^{\dagger}\mathbf{x}_e + [\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}]\mathbf{z}_{\mathbf{q}} \tag{3.72}$$

$$\tau = (\mathbf{S}^{\perp}\mathbf{J})^{T}\mathbf{f}_e + [\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}]\mathbf{z}_{\tau} \tag{3.73}$$

where the matrix $\mathbf{S}$ is the *selection* matrix and $\mathbf{S}^{\perp} = \mathbf{I} - \mathbf{S}$.

Visual Servoing

For target tracking in real time (e.g. grasping a moving object), even simple model-based vision can be too slow in unstructured environments. By embedding image processing into the controller a task specific vision servo can be devised. Weiss demonstrated visual servoing [Weiss, 1987] in which a desired *image* feature such as area or centroid was compared against actual image features and a mobile camera servoed to regulate the error between the features. This structure is depicted in figure 3.4.1.The system does not track an object in task space, but a feature in image space, a geometric object model does not exist.

A more ambitious system, MURPHY, by Mel [Mel, 1990] left feature extraction and inverse kinematics to a neural network to implement fast vision servoing and vision based obstacle avoidance. With a trained

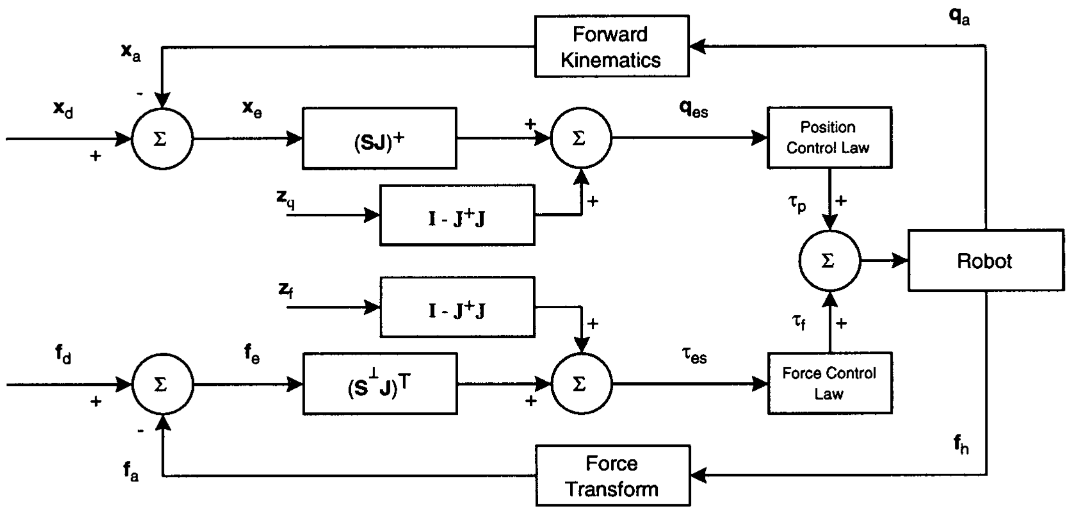


Figure 3.5: Fisher's corrected Hybrid Position Force Control. **S** is the selection matrix. $\mathbf{J}^+$ is the pseudoinverse of the Jacobian and $\mathbf{z_q}$ and $\mathbf{z}_\tau$ are arbitrary vectors.

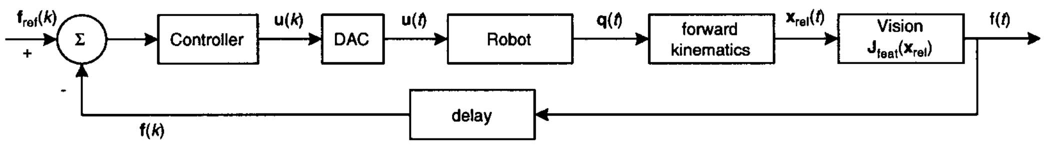


Figure 3.6: A block diagram of the Visual servoing approach to target tracking. Desired reference values are in feature space $\mathbf{f}_{\text{ref}}$. Image feature changes are computed through a feature Jacobian $\mathbf{J}_{\text{feat}}$.

the neural network, image, robot motion, and obstacle semantics became internal to the control system and unknowable to an external observer.

Obstacle Avoidance

SMPA based obstacle avoidance requires world models supplied with high speed, rich, range images over the robot's work space. At least two methods of real time manipulator obstacle avoidance [Colbaugh, 1989, Khatib, 1985] dispense both with the modelling of obstacle boundaries and planning of collision free trajectories. These algorithms simplify the obstacle avoidance process by relying on real time sensing and reactive control. By identifying points on the manipulator threatened by collision. "points subject to hazard" (PSH)[5] and applying an evasive force to each PSH, both schemes avoided collisions with workspace obstacles.

Khatib's Operational Space Formulation (OSF, see figure 3.4.1) [Khatib, 1987] employed a potential field obstacle model embedded into a simple control level obstacle avoidance scheme. By adopting a nonlinear potential field about an obstacle with a specified clearance range, manipulator motion was unconstrained while outside the clearance range of the obstacle. In OSF, Khatib circumscribed obstacles with artificial potential fields (APFs), Φ. By determining the negative gradient of these potential fields at the PSH, a repulsive force could be prescribed and applied to the PSH. The Jacobian Transpose of the PSH could then be used to project the repulsive forces onto the manipulator's actuators to generate an evasive maneuver.

Recognizing the computational burden in computing and assembling Φ, Colbaugh at JPL [Colbaugh, 1989] used range sensing, state *augmentation* and an adaptive controller to achieve similar results. In this implementation, range sensors on each link determine the range between the PSH and the nearest obstacle. By augmenting the state of the end effector with the state of the PSH to form an augmented task space vector, $\mathbf{x}_a$, and monitoring the range sensor values continuously, an *augmented* Jacobian Transpose could be applied to the augmented force vector.

Teleoperation

In recognition of practical real time system requirements, Albus et al. [Albus, 1990, Lumia, 1994] proposed a multilevel SMPA model of robot control that operates in parallel at multiple time scales, the NASA/NIST Standard Reference Model for telerobot control system architecture. In NASREM, the sense-model-plan-act cycle is spread over six time scale levels, successive scales differing by, approximately, an order of magnitude.

[5] A convention similar to Khatib's "points subject to potential" or PSP

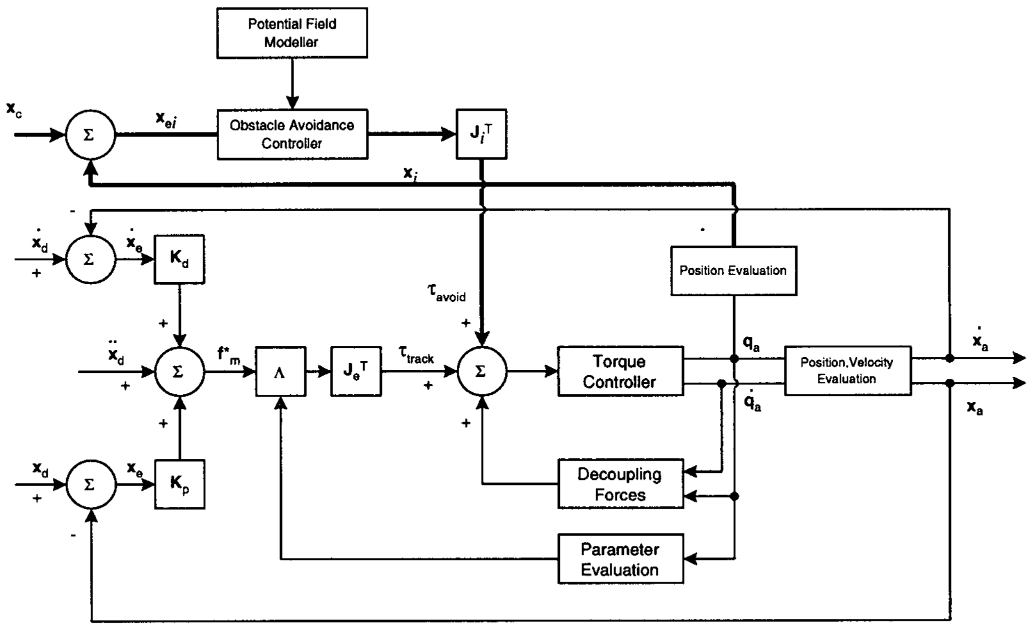


Figure 3.7: Khatib's operational space formulation. Feedforward cartesian dynamics and obstacle avoidance forces are summed to drive a manipulator along a stable collision free trajectory. Colbaugh's configuration control replaces the potential field model with sensing and the feedforward dynamics with adaptive control.

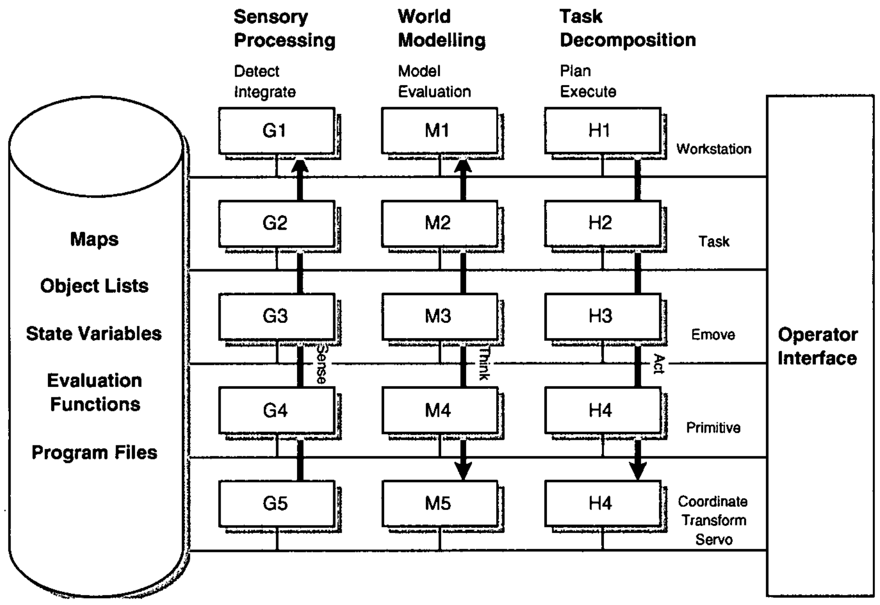


Figure 3.8: The NASA/NIST Standard Reference Model for telerobot control.

Level	Scope
servo	coordinate transforms
primitive	dynamics
elementary movement	path planning
task	actions on objects
service bay	assembly tasks
service mission	mission scheduling

Table 3.1: The temporal scope of the NASREM layers. Each layer represents an increasingly (approximately an order of magnitude) longer planning horizon .

All time scales access sensor data streams in the modelling and planning of tasks. Similarly all time scales contribute to the manipulators final motion. At any time, a human operator can interrupt the systems operation at any or all time scales. NASREM represents an important step in the formalized specification of robot system components and, like any specification, avoids detailed design issues. Only one published implementation conforms to this model [Lumia, 1994], though presumably NASREM is sufficiently flexible to accept many of the control systems reviewed here.

3.4.2 High Speed Motion

Real time robot control is naturally not limited to manipulation robotics, and much can be learned by reviewing high performance robots in other applications. The following is a brief overview of some particularly successful real time robotic systems.

Raibert's Hopping Robot

Raibert [Raibert, 1984] and others (e.g. [Pratt, 1995]) have designed robots to explore hopping and leaping dynamics. Raibert's was controlled through a sequencer, or finite state machine, driven by data streams from pressure, inclinometer, angle and position sensors. By observing incoming data streams the sequencer could coordinate height, velocity, and attitude controllers with the timing of the machine's support and flight phases.

Raibert's hopping robot employed a double acting pneumatic cylinder connected to a pair of pneumatic 'hip' actuator joints, and the entire leg/hip assembly to a large inertia balance beam. By tethering the balance beam to rigid aluminum boom, the robot was constrained to hop within a spherical surface. Two controllers cooperated in the motion of the robot.

Each phase of the hopping sequence was triggered by specific sensor thresholds determined through experimentation and analysis. The modelling portion of each phase was limited estimating the dynamics of the hopping frame, while the planning portion used the dynamic model to plan an angle/thrust response.

Raibert's robot is significant for its use of a combination of dynamic analysis, control, and finite state strategy sequencing rather than an SMPA like planning of the robots running stride. Quoting from [Raibert, 1984]:

> "The back and forth motions were not explicitly programmed, but resulted from interactions between the velocity controller that operated during flight and the attitude controller that operated during stance."

Andersson's Ping-Pong Player

Anderssons Ping Pong player [Andersson, 1989] used an 'expert controller' - an hybrid between an expert system and a controller. Andersson employed a 'figure of merit' system to trigger activity within the expert controller. Through an analysis of the ping pong ball dynamics, Andersson identified a set of 'free variables' upon which the ping pong task was dependent including: paddle orientation, ball velocity, manipulator settle time and others. In each control cycle these values would be run, in parallel, through a set of simple models, each generating a figure of merit. The model returning the highest figure of merit would be executed as the next task in the system. In effect, Andersson condensed the model-plan portion of the cycle into a set of parallel processes that simultaneously examined the free variable stream and developed correspondence measures.

3.4.3 Mobile Robot Navigation

Mobile robotics stands as one of the most challenging autonomous robotic tasks. Unlike manipulators, the robot is no longer holonomic and must rely on sensing and dead reckoning to establish its position (though differential global positioning systems make this task increasingly easier). The environment challenges sensors and actuators with inconsistent lighting, irregular surface properties and topographies, and complex obstacle morphologies. In an olympian contest, autonomous navigation systems pitch state of art hardware and software against changing environments only to reach a target location in one piece. Understandably, many of these systems are only partially successful, but all provide useful lessons in real time robot architectures.

Task Control Architecture (TCA)

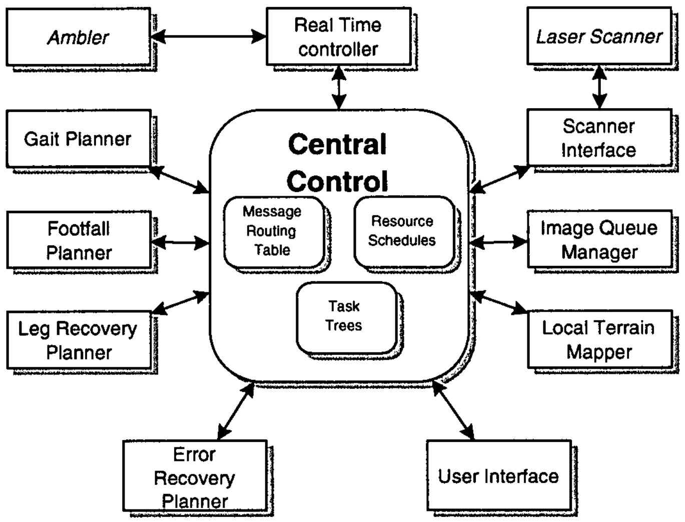


Figure 3.9: Carnegie Mellon's Task Control Architecture for the Ambler hexapod. Note the centralized planner and distributed reactive controller.

The Task Control Architecture, in figure 3.9, a product of CMU's Robotic Institute, sought to unite deliberative and reactive systems through an SMPA-like layer. TCA acts as a planner/oversear on top of task specific reactive systems. Perhaps the most well known implementation of TCA is on the Ambler hexapod. After initially using an SMPA cycle for Carnegie Mellon's six legged robot, 'Ambler's' Task Control Architecture (TCA) [Simmons, 1991, Simmons, 1994] was modified into an asynchronous reactive layer combined with traditional AI modelling and planning elements.

Distributed Architecture for Mobile Navigation (DAMN)

The DAMN architecture, depicted in figure 3.10, also developed at CMU, again sought to integrate deliberative planning with reactive control. In DAMN, a discrete set of control actions on a group of actuators (e.g. pan/tilt camera, vehicle steering motors, engine speed) forms a command space in which multiple modules concurrently share robot control. By voting for or against alternatives in the command space, each module contributes to the control commands for the robot. DAMN employs an arbiter for the resolution of the voting process on each device. In the case of the UGV project [Leahy Jr, 1994]: a turn arbiter, a speed arbiter, and a 'field of regard' arbiter. To explain the arbitration process the turn arbitration procedure will be described:

Each behaviour votes (ranging between -1 and $+1$) on every member of a discretized set of radii,R_{0i}. This means that each behaviour's vote is actually a distribution over all the possible steering radii. The

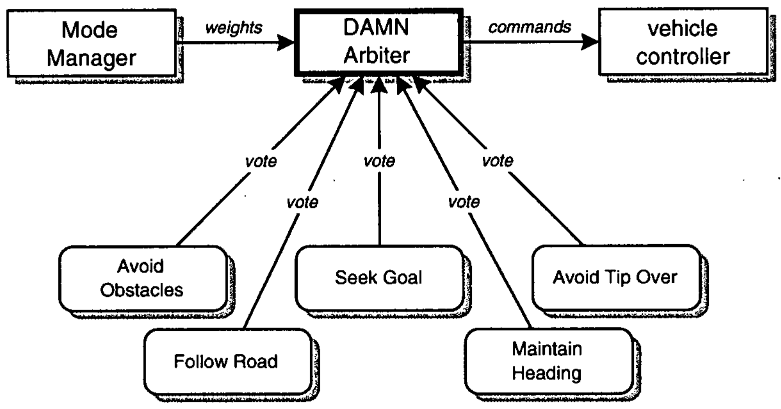


Figure 3.10: Carnegie Mellon's Distributed Architecture for Mobile Navigation for the NAVLAB series of robots. Each behaviour votes on every possible command (e.g. steering radii) and all votes are processed within the arbiter.

arbiter collects vote distributions from all participating behaviours, performs a gaussian smoothing on each, followed by a 'normalized weighted sum' for each of the i radii candidates:

$$v_i = \frac{\sum_j w_j v_i^j}{\sum_j w_j} \tag{3.74}$$

where w_j is a behaviour weight and v_j is the vote for the jth behaviour. The radius with the highest vote $v = \text{Max}(v_i)$, is sent to the controller. 'Field of regard' and velocity arbiters perform similar smoothing and selection operations. This approach allows for multiple modules operating at multiple frequencies to vote on various command spaces. DAMN runs on a number of platforms [Leahy Jr, 1994, Rosenblatt, 1995a, Rosenblatt, 1995b].

Motor Schemas

Motor schemas [Arkin, 1987, Arkin, 1991] are small processes that correspond to primitive behaviours that, when combined with other motor schemas, yield more complex behaviour. Arkin employed two kinds of schema, perceptual schema, that observed and represented the environment through sensing and potential field models respectively and motor schema that devised responses to classes of events. A central **move-to-goal** or **move-ahead** schema sums the responses and commands the robot motors.

Thus if a **find-obstacle** schema detected an obstacle, an **avoid-obstacle** schema was instantiated that produced a velocity vector based upon a repellent potential field around the obstacle (similar to Khatib [Khatib, 1985]). By summing the output velocity vectors from a collection of such schemas, a **move-robot** could navigate through the environment.

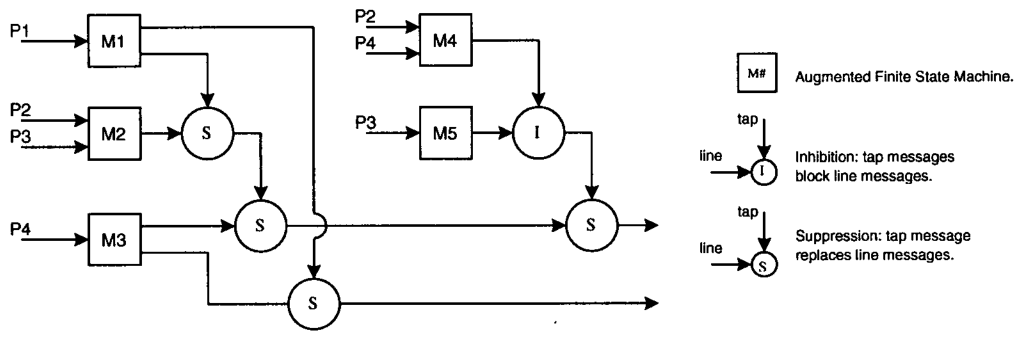

P1
P2
P3
P4
M1
M2
M3
M4
M5
P2
P4
P3
S
I
M#
Augmented Finite State Machine.
tap
line
Inhibition: tap messages block line messages.
Suppression: tap message replaces line messages.

Figure 3.11: A subsumption network. Perception (P) drives augmented finite state machines (M#) to output messages. Suppression nodes substitute horizontal line messages with vertical (tap) messages. Similarly Inhibition nodes disable line messages if a tap message is received.

Subsumption Architecture

Physically, subsumption is a hierarchical network of simple sensors, controllers, and actuators that can be embedded into relatively small robots. In Ferrell's 14 inch 3 kilogram hexapod, 19 degrees of freedom were controlled through 100 sensors, including leg mounted foil force sensors, joint angle and velocity sensors, foot contact sensors, and an inclinometer. Applications include aircraft flight and landing systems [Hartley, 1991], heterarchical subsumption (Connell's *Herbert*[Connell, 1990]), and hexapod motion (Ferrell's *Hannibal*[Ferrell, 1993]). Mataric [Mataric, 1994] *Nerds* showed that behaviour arbitration could be learned through repeated trials.

Each subsumption network node, an *augmented finite state machine* (AFSM), consists of registers, a combinatorial network, an alarm clock, a regular finite state machine, and an output. Sensors are connected to specific registers while actuators receive commands from the output of specific AFSMs. A message arriving at a register or an expired timer can trigger the AFSM into one of three states: wait, branch, or combine register contents. Results of combinatorial operations may be sent to an input register or an output port. Since each AFSM uses an internal clock, output messages can decay over time.

AFSMs can inhibit inputs and suppress outputs of other AFSMs through inhibition and suppression 'side taps' placed on input or output connections in the network. Inhibition side taps prevent transmission of original messages along an input connection if an inhibition message has been received from an AFSM. When a suppression message is sent to a suppression side tap from an AFSM, the original output message is substituted by messages from the AFSM.

Inhibition and suppression side taps encourage layered subsumption (as in figure 3.11) in which basic behaviours are embodied within a fundamental layer of AFSMs. Through judicious use of side taps, additional behaviours can be built over the basic set (e.g. 'leg down','walk', 'prowl'). Mataric developed a set qualitative criteria to aid in the selection of basic behaviours. Each behaviour should be:*Simple*, *Local* through local rules and sensors, *Correct* by provably attaining the desired objective, *Stable* through insensitivity to perturbations, *Repeatable*, *Robust* by tolerating bounded sensor or actuator error, and *Scalable* by scaling well with group size.

3.4.4 Computer Graphics

In order to produce complex, believable movement in computer animation, a number of novel strategies have appeared for motion control. While most ignore the details of real world dynamics, these systems apply a 'sensor' driven switching architecture to simplify the programming of flocks of birds [Reynolds, 1987], schools of fish [Tu, 1994], and planar running machines [van de Panne, 1992].

Boids

The realistic animation of large collections of entities e.g. crowds of people, schools of fish, etc. becomes time consuming and inflexible if the trajectories of each entity are specified a priori. In a novel solution, Reynolds [Reynolds, 1987] employed a set of three behaviours: collision avoidance, velocity matching, and flock centering to model formation flying within every bird-like graphical construct, bird-oids or boids.Each behaviour generated an *acceleration vector* and was placed in a priority list.

As each behaviour contributed a desired acceleration to the arbitrator, the arbitrator would accumulate both the acceleration vectors and the magnitudes of each output behaviour over time in order of priority. When the sum of the accumulated magnitudes exceeded a fixed acceleration value, the acceleration vector components would be apportioned to each behaviour in priority. Under normal flight conditions in which each behaviour is of approximately equal priority, this is equivalent to vector averaging. However, if one behaviour experiences an emergency and issues large magnitude vectors, this method effectively suppresses lower priority requests.

Similar strategies were used by Terzopoulos et al [Tu, 1994] to produce realistic behaviour within more sophisticated animated fish.

Sensor-Actuator Networks

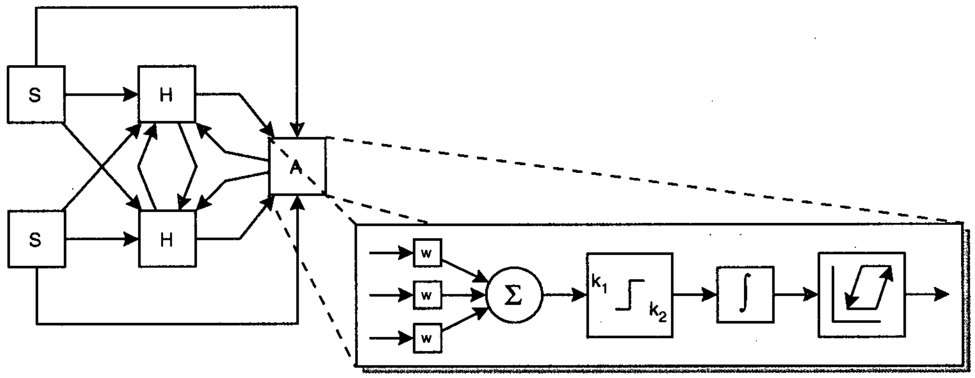


Figure 3.12: A SAN network example. Note the interconnections sensor (S), hidden (H) nodes and Actuator (A) nodes. Each node has the structure expanded at right. The hidden nodes act as an interactuator information mechanism.

The realistic animation of complex running or galloping entities, like the real world equivalent, is difficult to coordinate and control. Van De Panne and Fiume tackled this problem through the creation of Sensor Actuator Networks (SAN) [van de Panne, 1992]. Given a mechanical configuration and the location of binary sensors and PD actuators on the mechanism, a generate-and-test evolution method modulates weights connecting *sensor nodes*, *hidden nodes*, and *actuator nodes* of the network. In a complex information exchange, all sensor nodes are linked to all hidden and actuator nodes, all hidden nodes to one another and actuator nodes, and all actuators nodes to all sensors and hidden nodes. Each link is unidirectional and weighted. Within each node, a weighted sum is performed on all connections, thresholded, integrated, and filtered through a simple hysteresis function. The SAN structure is depicted in figure 3.12.

3.4.5 Common Denominators in Real Time Robot Control

Many of these architectures adopt common, pragmatic solutions to shared problems of computational complexity, limited time, and unstructured environments. While an SMPA controller might be feasible given sufficient computing and communication resources, in general the only means of implementing 'real world' autonomous robots (or real time animations) is through embedded, often multiprocess, control exploiting fast simple data streams. A summary of these shared characteristics follow.

Application Specific Sensing

While cost and precision requirements make task specific sensor suites more attractive than higher level general purpose sensors, time and computing constraints limit the amount of signal processing and modelling available for control. Together, these constraints drive the adoption of application specific signal processing and control.

Multiple Data Streams

Using multiple data streams, a single controller can perform limited sensor fusion based entirely on task specific criteria. Rather than performing broad based signal processing and recognition to develop a multi-purpose symbolic model, behavioural systems embed meaning into sensor data by responding through simple signal processing methods (e.g. thresholding) to specific stimuli. Thus each controller establishes a partial, self-interested, view of the available data streams.

Arbitration of Multiple Controllers

Using task specific sensing, a controller's output represents both an actuator command value and a confidence measure within a multicontroller environment. In effect, the output is a measure of the correspondence between the controller's task specific world model and the sensor data stream. In this way multiple behaviours classify the sensor data stream, *merging* the database-like world models and symbolic planners of the SMPA into a smaller, simpler action model. 'Correct' action selection is left to simple arbitration procedures (e.g. weighted averaging or switching).

Parallel Computation

A number of factors promote the distribution of control over multiple processors including sensor and controller complexity, disparate time scales, and robustness. Signal processing complexity can vary considerably within a single system e.g. from topographical LADAR mapping to inclinometers in the CMU Ambler project. Similarly controllers vary from simple PD-like control to sophisticated model based systems. *Concurrent* sensing and control removes I/O boundedness within the system during complex computation by allowing the system to respond over many scales. Furthermore, physically distributed computation makes the system less susceptible to catastrophic damage to the control system. In general, distributing computing cycles over multiple CPUs substantially reduces the cost of such systems, though some additional complexity may be encountered in interprocess communication.

In summary, practical constraints on cost, time, and performance are slowly driving robot control techniques that exploit inexpensive simple sensing, multiple control behaviours, and behaviour arbitration. These are the basic constituents of the modern robotic agent and to a significant extent, describe many existing real time manipulation systems. However, just as agent control methods may simplify the control of real time robots, teams of agents may simplify the control of complex, multiplant, systems.

3.5 Multiple Robot Control

Using multiple robots to achieve a single task is not new. Indeed, multiple manipulator control is a key issue for the International Space Station Special Purpose Dextrous Manipulator (SPDM) a twin armed manipulation system mounted on a remote manipulation system (RMS). However, the difficulties of real time manipulator performance become further magnified in such unstructured multirobot environments. Centralized methods must accommodate modelling and planning for multiple manipulators that carry common or separate payloads through a possibly cluttered work space. A task made more difficult by complex sensors and, possibly, remote computing installations. This section will review some examples of distributed robot team control and draw some conclusions on the significance of these architectures.

Reynolds' Boids revisited

Based on beliefs of actual bird behaviour, Reynolds reasoned that given similar, if simplified, sensing and flight dynamics, behaviours such as velocity matching, collision avoidance, and flock centering should be sufficient to 'hold together' a group of boids in a flock as well as produce realistic flock trajectories. In effect, Reynolds believed that desired team behaviour could be achieved through careful design of a team member.

After trial and error, this approach generated realistic flocking behaviour. Perhaps more interesting than the final flock performance are some of Reynold's observations on the design process:

- "Flock behaviour depends on a localized view". 'Realistic' behaviour did not require complete flock models within each boid.
- Linear collision avoidance rules (e.g. PD rules) produce oscillatory 'unrealistic' behaviour while nonlinear 'inverse square' rules promoted stability. In observing that linear flocking rules produced spring like behaviour, Reynolds demonstrated that agent teams are dynamic systems.
- Apparent flock complexity is due to environmental complexity (e.g. obstacles), an phenomenon observed by others (e.g. the *Intelligence* property identified by Brooks [Brooks, 1989a]).
- The control algorithm is O(N) in the number of boids, due primarily to the implementation of the behaviours. Autonomous agents in a multiagent systems should be constant time.

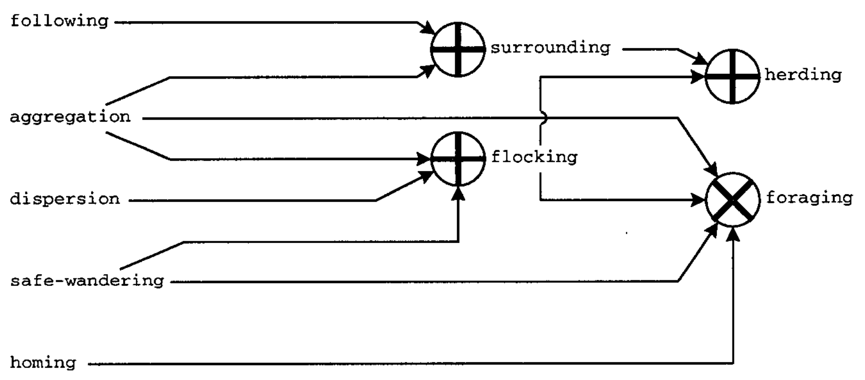


Figure 3.13: By applying both direct $\oplus$ and temporal $\otimes$ combination to the same basic behaviours higher-level behaviours can be generated. In this example, safe wandering is used to generate both flocking and foraging. Similarly aggregation is used in both foraging and in surrounding.

The central conclusion is that multiagent systems can exhibit desirable behaviour *without* explicit coordination from an external supervisor.

Ferrell's Hannibal: single or multiagent?

Though subsumption architecture has resisted classification, the approach falls within the definition of agency. In some subsumption networks it is debatable whether individual AFSMs are not themselves agents and that these hierarchies are not multiagent systems. Indeed, Ferrell refers to AFSMs in *Hannibal* as *agents*, each distributed over each leg of the hexapod coordinated loosely by a set of *global* agents. Each leg of the hexapod is controlled by a discrete AFSM network and the legs as a whole are coordinated by a central timing agent. In this instance coordination between agents is explicit, timing signals set the walking pace of the machine. However, the implementation of each step is unique to each leg. This demonstrated that while a global agent might explicitly synchronize agent activities, it need not explicitly specify the activity itself. Strict subsumption systems such as Hannibal show that multiagent systems can be hierarchical and yet autonomous within any given layer. Indeed, viewed as a multiagent system, Connell's Herbert shows that multiagent systems can be heterarchical.

Mataric's Nerd Herd

Mataric investigated the interaction of agents within a group of 20 mobile robots, 'the Nerd Herd'. As mentioned earlier, each robot was equipped with a set of *basic behaviours*: `Safe-Wandering`, `Following`, `Dispersion`, `Aggregation`, and `Homing`. These controllers used input from contact sensors, piezo-electric bump sensors, IR range sensors, and IR break beam sensors. Each shoe box sized robot could collect and stack pucks with a simple fork-lift gripper/actuator assembly. By expressing basic behaviours in terms relative to the agent or the environment, global group behaviour was distilled into each agent. For example `safe wandering` was expressed as:

$$\frac{dp_j}{dt} \neq 0 \quad \text{and} \quad d_{ij} > \delta_{\text{avoid}} \tag{3.75}$$

where p_j is the absolute position of the jth agent, $\frac{dp_j}{dt}$ is the velocity, δ_{avoid} is a range threshold, d_{ij} is the magnitude of the distance to the ith agent or obstacle. Similarly, `following` minimized the angle θ between the relative distance vector and the agent's velocity vector:

$$0 \leq \frac{dp_j}{dt} \cdot (p_i - p_j) \leq \left\| \frac{dp_j}{dt} \right\| \; \|(p_i - p_j)\| \; cos\theta \tag{3.76}$$

where i is the leading and j is the following agent. Similarly `dispersion` and `aggregation` employed attraction and repulsion controllers. By modulating the velocity vector through the use of these rules desirable global behaviours were generated.Figure 3.13 summarizes the behaviour hierarchy developed through behavioural and temporal sequencing in a single *Nerd.*

Troops

Parker [Parker, 1994] successfully explored the use of robot systems in the mediation of (simplified) toxic spills. The goal of the research was to establish a software architecture that enabled explicit cooperation between robots. Each robot possessed a set of low level subsumption behaviours. These behaviours are grouped into sets, each set representing a high level task (e.g. cleaning the floor). A set of prioritized *motivational behaviours* receives information through sensors, communication, and feedback from other behaviours and selection behaviour sets based on:

Behaviour Set Priority α_i. (integer).

Sensor Feedback $F_i(t)$. the applicability of the behaviour set to the current situation, (boolean).

Impatience $I_i(t)$. the period this agent will wait for another agent to accomplish the same behaviour (real).

Acquiescence $A_i(t)$ the period this agent will maintain the behaviour before yielding to another agent (boolean).

Suppression $S_i(t)$ whether another behaviour suppresses this behaviour set (boolean).

The motivation calculation:

$$m_i(0) = \alpha_i \cdot F_i(t) \tag{3.77}$$

$$m_i(t) = (m_i(t-1) + I(t)) \cdot A_i(t) \cdot S_i(t) \cdot F_i(t) \tag{3.78}$$

if $m_i(t) > \theta_i$, a threshold of activity, then the behaviour set is activated and, possibly, a message broadcast to other agents. Based upon observed broadcasts and the environment, an agent could either engage a task achieving behaviour set, wait while another agent performed the task, or, if an agent failed to perform, take over a task.

In this model of multiagent activity, agents inform one another through broadcast communication and do not know or assume the abilities of other agents present. Cooperation arises through opportunistic agent activity. Agents act only if they are ready and able to respond. Since there is no central coordinator or agent-to-agent information exchange, the system is robust to agent failure.

3.5.1 Common Denominators in Real Time Robot Team Control

Again, sharing similar problems to robot team control, multiagent systems possess common solutions worth summarizing. In particular: Global and Local goals, Interagent Communication, Robustness, Locality in Sensing, Linearity in Computation, and Interagent Dynamics.

Global Goals

Agent teams are often fulfill a team or global objective. The objective is either a specific goal (e.g. a trajectory) expressed by an external process to the team, or arises from interagent dynamics (e.g. Reynolds' boids) and the changing environment (e.g. Ferrell's Hannibal).

Local Goals

Local goals are often used to ensure safe operation. Global goals often coexist (and even arise from) the maintenance of local goal systems. Local goals are usually explicitly stated *relative* to the agent and not some global coordinate system (e.g. **move-forward**). Though a number of *similar* goal arbitration methods exist, there does not seem to be a consensus how to establish goal priorities or how local goal systems should be selected and/or designed.

Interagent Communication

Individual agents often communicate either through one-to-one (directed) transmission between agents, through broadcast, or through interagent sensing. Parker's Troops used directed messages between agents over RF bands while Mataric's Nerd Herd did not generally communicate. Similarly Ferrel's Hannibal did not use interleg communication to achieve stable gates. A common feature amongst most systems is that if a global objective is set, it is done so externally through some goal generation system and communicated either to all the agents or through some lead agent (e.g. boid's flock leader and Ferrel's synchronizing agent).

Global Robustness

The fulfilment of a global goal is often robust to individual agent failures. Mataric's system demonstrated considerable robustness to failures in positioning and communications, achieving the puck collecting objective in spite of these problems. Similarly Ferrel's Hannibal was robust to leg failures and environmental irregularities, while flocking in Reynold's boids was not influenced by flock fission or fusion.

Locality in Sensing

Individual agents sense only neighbouring team members and do not monitor the entire team. In general, sensing is kept local to each agent: short range sensing, limited boid fields of view, joint angles, forces, etc. This limits the impact of distant events on individual agents and, subsequently, lowers the computational burden. This means that global goals must be expressed either in terms of locally sensible events or communicated to each agent. There are exceptions to this rule, however. Hannibal adopted a single inclinometer

Constant Time Computation

Additional agents within a team do not significantly add to the computational burden on an individual agent. The short sensor horizon, limited modelling, and embedded computation within each agent means that (ideally) the number of agents should not affect the response time of a lone agent. In effect a multiagent system becomes a distributed, parallel computer.

Interagent Dynamics

Reynolds' Boid system demonstrated that multiagent teams are dynamic systems. Ferrel's Hannibal relies on dynamic interaction with the environment to achieve stability. Mataric's Nerd Herd, too, demonstrates characteristics of dynamic systems, though, surprisingly, she felt analysis of the team as a particle system had little merit.

3.6 Remarks

In this chapter we have reviewed the structure of supervisory manipulator control and shown that, in context, real time control of robots in an unstructured environment often requires novel robot architectures, most of which bear little resemblance to SMPA supervisory control.

Clearly, real time control of robots and robot teams requires integrated design of the four components of SMPA. It seems that rather than generating a monolithic model-plan system, real time systems classify and reduce environmental events into a set of controller processes. The response of each controller is often used as a measure of correspondence between the classification and the observed world. Though there is no consensus on action selection or combination from the controller set, voting schemes, simple combination, and switching (through AFSMs) seem the most common.

The next chapter will take these features and place them into a familiar, concise, control theoretic representation, laying the basic foundations for both Agent and Multiagent Control Theory.

Chapter 4

The Agent and Multiagent Model

4.1 Introduction

Previous chapters have discussed the SMPA architecture, its origins in AI, and the difficulties in extending it to real time environments. In particular, the last chapter reviewed architectures that break away from the SMPA model to a class of task specific architectures. Furthermore, a set of features common to real time robotics was identified, reinforcing the perception that a genuinely new class of controller, an *agent*, is emerging. Agent based control has spawned interest in multiagent systems, agents acting not only on the environment, but on and with other agents.

The objective of this chapter is to propose a control theoretic agent model based on observations of common features in existing agent and multiagent systems. In so doing, terminology will be defined, relating the characteristic features of agent control to traditional control theory. Within this framework, the reputed abilities of these systems will then be summarized within a set of *hypotheses.* Given these properties of agency, further definitions of *multiagency* will be offered and hypotheses on multiagent control proposed.

4.2 Behaviour Control

Though the basic philosophy of multithreaded behaviour arbitration that underlies agent control is widely acknowledged, the theoretical foundations remain weak due in large part to the wide variety of implementations, interpretations, and nomenclature of agent control. These applications range from production scheduling systems and autonomous mobile robot teams to software *avatars* (virtual personas) and combat simulators. Terminology is often the source of confusion in any growing field, agent and multiagent control is no exception.

Recognizing the breadth of*agent oriented* applications [Shoham, 1993], this thesis focuses exclusively on the control of robotic systems. Even within this narrow context, the definition and structure of an agent has yet to be formally fixed, though most agree that an agent contains some kind of 'behaviour arbitration mechanism' capable of perception and action [Mataric, 1994, Parker, 1992b, Kosecka, 1994]. Mataric

[Mataric, 1994] proposes the following qualitative definition "An agent is a process capable of perception, computation and action within its world...". Mataric goes on to develops a spectrum of agent control schemes based on modelling and state retention including *reactive* systems possessing no memory, *behavioural* systems with limited state retention, *hybrid* methods relying partially on symbolic systems, and purely symbolic *deliberative* systems.

MacKenzie, Cameron, and Arkin [MacKenzie, 1995] also proposed that an agent is "... a distinct entity capable of exhibiting a behavioural response to stimulus ", providing the symbolic expression:

$$Agent \equiv Behaviour(Stimulus) \tag{4.79}$$

defining an *agent* as a *behaviour*, itself a function of some *stimuli*.

To complicate matters, the term *behaviour* is rarely used consistently. From Steels [Steels, 1991] behaviour is "... a regularity in the interaction dynamics between the agent and the environment". While Mataric suggests behaviour is [Mataric, 1994]: "...a control law that clusters a set of constraints in order to achieve and maintain a goal". This latter definition seems to equate control and behaviour as does Parker [Parker, 1992a]. MacKenzie et al [MacKenzie, 1995] are more specific. Behaviour is y where:

$$y = f(v_1, v_2, \ldots, v_m) \tag{4.80}$$

where v_i are some input variables and $f(\cdot)$ is defined $\forall v_i$. Others are careful to discriminate between control and observed behaviour (e.g. [Colombetti, 1996]). Despite some confusion of terminology, *behaviour* is clearly related to the response of some system to control input.

Some have defined agents as a hybrid control problem, one in which discrete and continuous systems coexist, and have applied Discrete Event Systems (DES) theory[Kosecka, 1994] and a notable variant, Constraint Net (CN) theory [Zhang, 1995] to agent analysis. Discrete event systems experience asynchronous state changes at discrete points in time. Each *state* in the system corresponds to some continuous evolution of the system, while each *event* is a discontinuous transition between qualitative changes in the system's behaviour. In this model the definition of behaviour is *trace* [Zhang, 1995] or "an event trajectory"[Kosecka, 1994] and the agent a Supervisory Controller (or Constraint Solver in [Zhang, 1995]). By issuing a trace or event stream, these controllers can modulate the output of the plant, itself an event stream or trace, to follow a desired event trajectory. Since each symbol represents a control strategy, Pure DES controllers are a means of *sequencing* action towards a desired objective. Though powerful in uniting analog and switching controllers, neither represents new techniques in the design of individual controllers nor provide insights into the interaction between these controllers and the environment.

Agency and Control Theory

Though few investigators dispute Brooks' [Brooks, 1991a] assertion that 'the world should be its own model' and Steel's [Steels, 1991, Steels, 1994] observation that agents are dynamic systems, most continue to believe that furthering agent control depends on better computational and linguistic apparatus and not on the application of control theory. Some have concluded that agent based system's can only be designed through field trials and that control theoretical approaches have limited value. From Mataric [Mataric, 1994]:

> The exact behaviour of an agent situated in a nondeterministic world, subject to real error and noise, and using even the simplest of algorithms, is impossible to predict exactly...[1] Precise analysis and prediction of the behaviour of a single situated agent, specifically, a mobile robot in the physical world, is an unsolved problem in robotics and AI. [2]

While it is true the exact trajectory of any dynamic system is not *exactly* knowable, Mataric's statement underestimates the value of a dynamics and controls analysis of even simplified problems. Indeed, amongst subsumption researchers in particular there is an open prejudice against simulation [Parker, 1994].

Imprecise, confusing terminology has obscured the specification of agent designs and hindered the formalization of a plainly successful real time control technique. The result is a collection of systems that have many common structural attributes but, surprisingly, little theoretical common ground. To enable a theoretical treatment, this section will review the concepts behind agent control from the control theoretic standpoint, and establish formal definitions where possible. Through this process a set of hypotheses will be proposed that characterize agent design and performance objectives.

4.2.1 Goals

The term *goal* has many interpretations and is discussed in detail by Weir [Weir, 1984]. Colloquially understood to represent a desirable, possibly time varying, state of some system, the philosophical arguments reviewed in [Weir, 1984] center mostly on whether such states are external or internal to the construction of goal directed systems. For the purposes of machine control however, a goal is generally equivalent to the *desired behaviour* of a controlled mechanical plant. Such behaviours are sometimes expressed as the abstract "wander" [Brooks, 1989c] while others are more precisely expressed as a static setpoint or time varying trajectory through some relevant coordinate system (e.g. position, velocity, temperature, etc.) . A broadbased definition can then be proposed:

[1] page 27: paragraph 3.
[2] page 29: paragraph 4

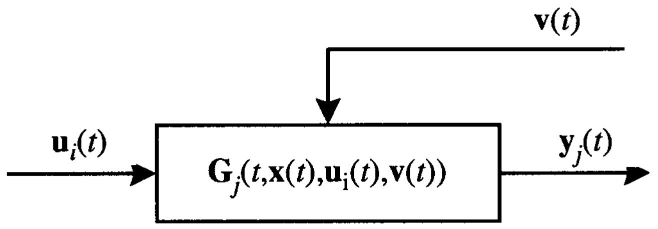

$\mathbf{v}(t)$
$\mathbf{u}_i(t)$
$\mathbf{G}_j(t,\mathbf{x}(t),\mathbf{u}_i(t),\mathbf{v}(t))$
$\mathbf{y}_j(t)$

Figure 4.14: A generic plant, $\mathbf{G}_j(\cdot)$, with control input $\mathbf{u}_i(t)$ and behaviour $\mathbf{y}_j(t)$.

Definition: D4.1 (Goal) *If a trajectory,* $\mathbf{r}_i(t)$, *is specified to traverse through some space,* $\mathcal{G}_i \in \mathbb{R}^g$, *then* $\mathbf{r}(t)$ *is a goal and* $\mathcal{G}_i$ *a* goal space.

There are no preconditions on the function $\mathbf{r}_i(t)$. However manipulator end effector goal functions $\mathbf{r}_i(t) = \mathbf{x}_d(t) \in \mathbf{R}^6$ are usually at least C^1 trajectories through task space. Since mobile systems often leave path planning to an on board reactive system goals are usually discretely changing planar positions or perhaps intervening waypoints or $\mathbf{r}_i(t) = \mathbf{x}_d(t) \in \mathbf{R}^3$ [Gat, 1994].

4.2.2 Behaviours

The artificial intelligence community uses the term *behaviour* to describe the observable evolution of an automaton's state. In robot control, behaviour is understood to mean the performance of the robot within its environment. The colloquial use of " behaviour" in the context of machine control generally refers to the *response* of some plant.

So to clarify the concept of behaviours, consider the plant in figure 4.14. In this figure a control effort $\mathbf{u}_i(t)$ and a disturbance, $\mathbf{v}(t)$ drives a plant, $\mathbf{G}_j(\cdot)$, to produce a response $\mathbf{y}_j(t)$. To an observer, the behaviour of a plant is simply the observable response, $\mathbf{y}_j(t)$.

Definition: D4.2 (Behaviour) *Consider an arbitrary plant,* $\mathbf{G}_j$, *with state,* $\mathbf{x}_j \in \mathbb{R}^{x_j}$ *that is perturbed by both control effort,* $\mathbf{u}_i(t) \in \mathbb{R}^u$, *and environmental disturbances* $\mathbf{v}(t) \in \mathbb{R}^v$. *Then the plant's* behaviour *is the* observed *output,* $\mathbf{y}_j(t) : \mathbb{R}^{x_j} \times \mathbb{R}^u \times \mathbb{R}^v \times \mathbb{R}^+ \to \mathbb{R}^b$*:*

$$\mathbf{y}_j(t) = \mathbf{G}_j(t, \mathbf{x}_j(t), \mathbf{u}_i(t), \mathbf{v}(t)) \tag{4.81}$$

and the state of the system evolves according to some relation:

$$\dot{\mathbf{x}}_j(t) = \mathbf{f}[t, \mathbf{x}_j(t), \mathbf{u}_i(t), \mathbf{v}(t)], \quad \forall t \geq 0 \tag{4.82}$$

For example, given a linear time invariant system, a familiar multivariable description of the system can be composed:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{D}\mathbf{v}(t) \tag{4.83}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{F}\mathbf{u}(t) \tag{4.84}$$

Through appropriate selection of $\mathbf{C}$ and $\mathbf{F}$, the system's behaviour, $\mathbf{y}(t)$, can be defined. For a linear time invariant manipulator, $\mathbf{u}(t)$ could be a vector of desired joint torques and, with $\mathbf{C} = \mathbf{I}$, the identity matrix, and $\mathbf{F} = 0$, the observed behaviour could be defined as the end effector state $\mathbf{y}(t) = [\mathbf{x} \;\; \dot{\mathbf{x}}]^T$. However, this expression could just as easily describe the ith time invariant link, in which case $\mathbf{u}_i(t) = \tau_i$, a scalar, and $\mathbf{y}_i(t) = [q_i \;\; \dot{q}_i]^T$.

D4.2 reflects the consensus that "behaviour" is the observed response of the plant. Since any realistic plant produces a bounded range of observable behaviours a *behaviour space* can also be defined, formally:

Definition: D4.3 (Behaviour Space) *If the set of all possible output trajectories,* $\mathbf{y}_j(t)$, *are bounded within a region,* $\mathcal{B}_j \subseteq \mathbb{R}^b$, *then* $\mathcal{B}_j$ *is the behaviour space of* $\mathbf{y}_j(t)$.

4.2.3 Behaviour Control

In agent and multiagent control literature, confusion arises when investigators use the term *behaviour.* Though differences in desired and actual response in a plant are minimized through control, "behaviour" is usually used indiscriminately, equating control and response. For example, in describing an obstacle avoidance behaviour in a mobile robot it is often unclear whether the robot's final response or driving control algorithm is described.In using such terminology, the process of controller design is ignored and, more significantly, the dynamics between control and environment overlooked. Nevertheless, the intent of this usage is clear: controllers can be designed to generate a specific predictable response from the system under specific stimuli. So to clarify this usage, a more precise concept, *behaviour control* will be examined in detail.

The purpose behind behaviour control is the generation of a predictable response to a specific, but narrow, set of stimuli. Suppose a controller can be designed to react to a specific stimulus. The controller must map the stimulus through some setpoint to a control effort that generates the desired response from the plant. The setpoint, a point or trajectory in a controller's input space, is clearly a goal description and the plant's response, as described earlier, a behaviour. Immediately, an elementary necessary condition of *reachability*

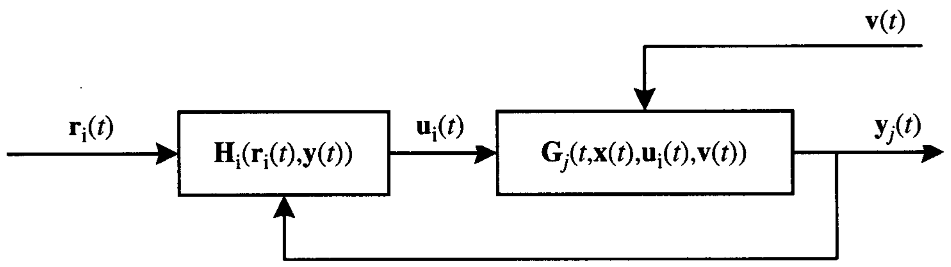


Figure 4.15: A goal $\mathbf{r}_i(t)$ input to the generic controller, $\mathbf{H}_i(\cdot)$, with output $\mathbf{u}_i(t)$ controls the plant, $\mathbf{G}_j(\cdot)$ making the behaviour, $\mathbf{y}_j(t)$.

can be applied to behaviour control. A goal is reachable by the plant if the goal resides within the range of all possible plant behaviours.

Definition: D4.4 (Reachable Goal) *Given a behaviour space, $\mathcal{B}_j$, and a goal space, $\mathcal{G}_i$, a goal $\mathbf{r}_i(t) \in \mathcal{G}_i$ is reachable if* $\mathbf{r}_i(t) \in \mathcal{G}_i \cap \mathcal{B}_j$

A controller, as depicted in figure 4.15, achieves a desired response by modulating a control input to a plant such that the desired and actual responses converge, specifically:

Definition: D4.5 (Control) *If a process $\mathbf{H}_i(\cdot, \mathbf{r}(t))$ producing a control effort $\mathbf{u}_i(t)$, can be designed such that given a reachable goal $\mathbf{r}_i(t) \in \mathcal{G}_i$, the output behaviour, $\mathbf{y}(t)$, is stable about $\mathbf{r}_i(t)$, then the function, $\mathbf{H}_i(\cdot, \mathbf{r}(t))$, is a* controller.

Thus control ensures that desired and actual responses are convergent. Clearly, *goal seeking* behaviour is synonymous with convergence:

Definition: D4.6 (Goal Seeking Behaviour) *If the behaviour $\mathbf{y}_j(t) \in \mathcal{B}_j$ is stable[Vidyasagar, 1993] about a reachable goal $\mathbf{r}_i(t) \in \mathcal{G}_i$, then the behaviour is* goal seeking.

This definition equates *goal seeking* with stability, including its progressively strict definitions such as local, global, and asymptotic, and exponential stability *stability* [Vidyasagar, 1993]. D4.5 establishes an artificial, causal relationship between a goal state and the observed behaviour and imposes an artificial dynamic equilibrium on the system through the application of control forces. In contrast, the more general D4.6 states that goal seeking behaviour can arise from any dynamic equilibrium – no causal relationship has been assumed (though one may exist nevertheless). In short, goal seeking behaviour does not imply control though control does imply goal seeking behaviour.

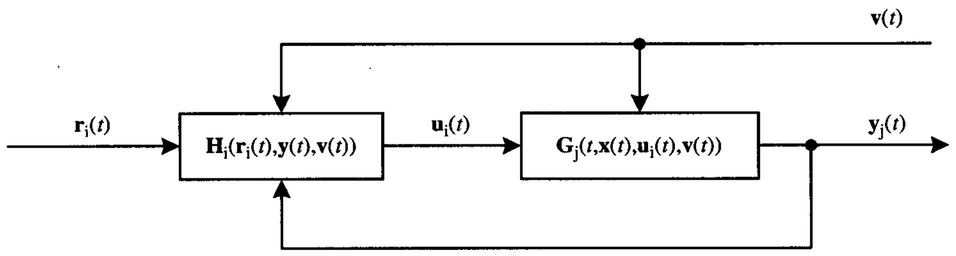


Figure 4.16: A goal $\mathbf{r}_i(t)$ input to the generic controller, $\mathbf{H}_i(\cdot)$, observes disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t)$ and control the plant, $\mathbf{G}(\cdot)$ making the behaviour, $\mathbf{y}_j(t)$.

For example: given stable `turn-right-to-avoid` and `turn left` controllers, a mobile robot will spontaneously follow a wall. Thus the apparent goal seeking `follow-wall` behaviour arises out of equilibrium with the environment and the two controllers. Unless an explicit `follow-wall` goal was issued, one cannot say the robot is controlled by a `follow-wall` controller, only that the robot exhibits `follow-wall` behaviour.

Ideally, *perfect* control is the result of complete knowledge of the plant's dynamics and disturbances. When condensed into a *model*, this knowledge can be used within the controller to cancel undesirable dynamics and disturbances.

In an unstructured environment, however, such complete knowledge is often too complex to maintain in real time. If only partial, task specific models are available, a task specific controller may be used to manage a particular disturbance or behaviour. In short, such behaviour controllers conforming to the structure in figure 4.16 respond to particular stimuli (e.g. wall collisions) using appropriate sensing (e.g. range finders), a suitable control algorithm, and a setpoint strategy to either minimize disturbances to the system or engage specific behaviours:

Definition: D4.7 (Behaviour Control) *If a process* $\mathbf{H}_i(\cdot, \mathbf{r}_i(t), \mathbf{v}(t))$ *can be designed such that, given a reachable goal* $\mathbf{r}_i(t)$, *the output of* $\mathbf{H}_i(\cdot)$, $\mathbf{u}_i(t)$, *minimizes the effect of a specific disturbance* $\mathbf{v}(t)$ *and if the behaviour* $\mathbf{y}_j(t)$ *is stable about* $\mathbf{r}_i(t)$, *then the function,* $\mathbf{H}_i(\cdot, \mathbf{r}_i(t), \mathbf{v}(t))$, *is a* behaviour controller.

Some good examples of behaviour control include:

- Hartley and Pipitone's [Hartley, 1991] aircraft landing gear deployment/retraction triggered at a fixed altitude.
- Arkin's [Arkin, 1987] obstacle avoidance triggered by obstacle range.
- Reynold's [Reynolds, 1987] boid velocity matching behaviour.

Given these definitions, behaviour controlled systems seem to be simple variations on traditional control albeit under new nomenclature. How does agent based control differ from traditional systems?

4.3 A General Model of Agency

Of course, agents *are* traditional control systems that exploit multiple goal systems to achieve desired behaviour. In highly structured environments, it is possible to devise model based traditional closed loop control system to detect and compensate for all likely disturbances. However, as the environment becomes more complex so too does the model dynamics, greatly challenging centralized closed loop architectures in performance, cost, and complexity.

One solution to the growing complexity of centralized model-based controllers is to arbitrate between a *set* of controllers, each coping with a specific contingency, to generate desirable plant behaviour. Though unproven, it is widely accepted that arbitrary behaviours can be decomposed into either simultaneous and/or sequential 'atomic' behaviours. Usually, atomic behaviours respond to a physically accountable stimulus, producing behaviour through either attractive or repulsive controllers.

All arbitrators seem to fall between two extremes: combination and switching. Combined arbitration mixes the outputs of behavioural controllers algebraicly to achieve the composite behaviour. Conversely, switching arbitrators switch from one controller to another based on some criteria. In either instance, arbitration between behaviours plays a critical role in the generation of final desired behaviour. Figure 4.17 documents this multicontroller structure while figure 4.18 depicts a condensed shorthand representation.

The set of controllers and the arbitration process together form a new control element known as an *agent*:

Definition: D4.8 (Agent) *Given a plant,* $\mathbf{G}_j(t, \mathbf{x}_j(t), \mathbf{u}_i(t))$, *and a set of behaviour controllers,* $\mathbf{H}_i(t, \cdot)$: $1 \leq i \leq b$, *an agent,* A, *combines* $\mathbf{u}_i : 1 \leq i \leq b$ *to form a composite control effort,* $\mathbf{u}_{cj}(t)$, *to generate the behaviour* $\mathbf{y}(t)$ *or:*

$$\mathbf{u}_{cj}(t) = \mathsf{A}_j(\mathbf{u}_1(t), \ldots, \mathbf{u}_b(t)) \tag{4.85}$$

$$\mathbf{y}_j(t) = G_j(t, \mathbf{x}(t), \mathbf{u}_{cj}(t), \mathbf{v}(t)) \tag{4.86}$$

By mapping the relatively informal definitions and structures from literature to more stringent terms within control theory, it is possible to specify an agent as a component composed of three fundamental parts:

- a *plant*: a controlled dynamic system acting within some environment.

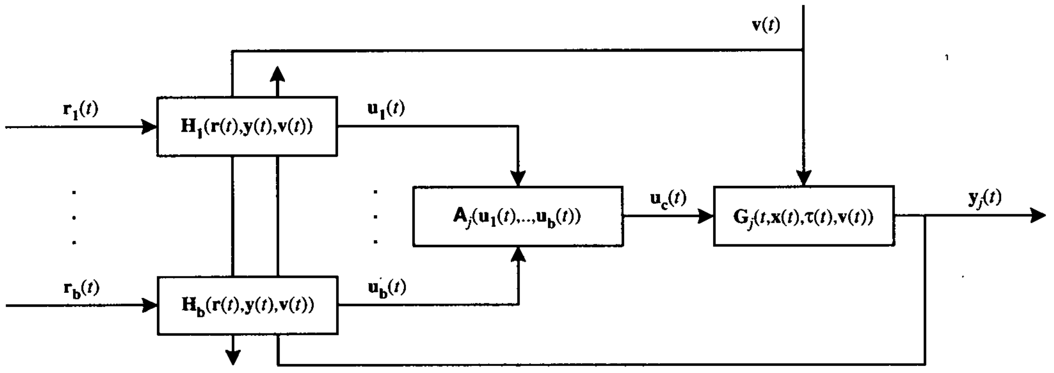


Figure 4.17: A set of goals $\mathbf{r}_i(t) : 1 \leq i \leq b$ input to generic controllers, $\mathbf{H}_i(\cdot) : 1 \leq i \leq b$, each observing select disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t) : 1 \leq i \leq b$ to the arbitrator $\mathsf{A}_j(\cdot)$. The composite output $\mathbf{u}_{cj}$ controls the plant, $\mathbf{G}_j(\cdot)$ to make the behaviour, $\mathbf{y}_j(t)$.

- a set of *behaviour controllers* that respond to specific environmental stimuli.
- an *arbitrator* that combines behaviour controllers to control the plant.

In effect, the combination process condenses and segments the model-plan portion of the SMPA cycle into smaller subproblems. From the earlier literature review, it is clear that many believe that this represents a fundamental control strategy, specifically:

Hypothesis: H1 (Agent Control) *Given a reachable goal,* $\mathbf{r}_{cj}(t)$*, and a set of behaviour controllers,* $\mathbf{u}_i(t) = \mathbf{H}_i(t, \cdot, \mathbf{r}_i(t)) : 1 \leq i \leq b$*, an* agent, A_j*, can be designed to combine* $\mathbf{u}_i(t)$ *into* $\mathbf{u}_{cj}(t)$ *such that the composite behaviour* $\mathbf{y}_j(t)$ *is stable about* $\mathbf{r}_{cj}(t)$*.*

In effect, A_j acts as composite controller with output, $\mathbf{u}_{cj}(t)$ or:

$$\mathbf{u}_{cj}(t) = \mathsf{A}_\mathsf{j}(\mathbf{r}_{cj}(t), \mathbf{y}(t), \mathbf{u}_i(t), \mathbf{v}(t)) \tag{4.87}$$

Based on control theoretic arguments, a general model of agency has been proposed. Given a fixed plant, this model highlights the two fundamental design problems of agent based systems: behaviour controller and arbitrator design.

Though apparently distinct in this model, behaviour controller and arbitrator design are not always distinct in practice and are rarely simple in structure. While some architectures draw clear distinctions between control and arbitration (e.g. the DAMN architecture), others appear to closely couple decision and control (e.g. subsumption or voting models). Furthermore, in some systems (e.g. subsumption and

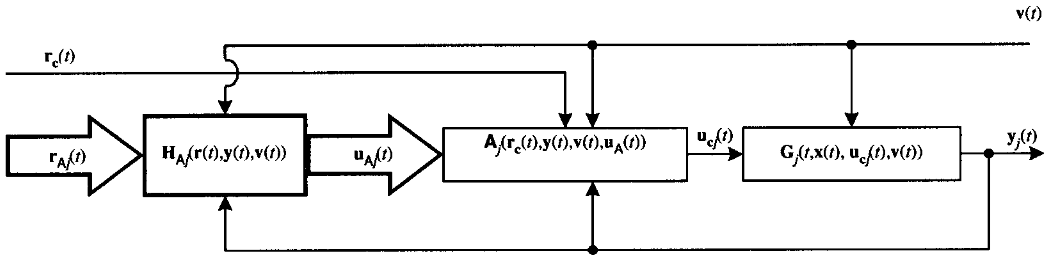


Figure 4.18: A set of behaviour controllers $\mathbf{r}_i(t)$ and $\mathbf{H}_i(\cdot)$, each observe select disturbances $\mathbf{v}(t)$ to output $\mathbf{u}_i(t)$ to the arbitrator $\mathsf{A_j}(\cdot)$. $\mathsf{A_j}(\cdot)$ arbitrates amongst these controllers to track a desired goal $\mathbf{r}_{cj}(t)$. The composite output $\mathbf{u}_{cj}(t)$ controls the plant, $\mathbf{G}_j(\cdot)$ to make the behaviour, $\mathbf{y}_j(t)$.

the SAN networks) arbitration is driven by individual controllers (a bottom up approach) while others rely on a centralized decision making process (e.g. Andersson's Ping Pong Player). Structurally arbitration mechanisms have been formed from linear (e.g. Motor Schemas) and nonlinear (subsumption), discrete (e.g AFSMs) and continuous functions of controller output, sensor data, and time.

A first step to understanding arbitrator design and the coupling between arbitration and control can be made by examining the information flow or communication between these controllers and arbitrator. The following section examines these data flow relationships to aide in the further classification of arbitration schemes.

4.3.1 Information Exchange

Information exchange can be classified according to the relationship between sender and receiver of a message. Fundamentally, a *transmission* has a source or *sender*, an audience or *receiver*, and possess' data content, a *message*. The act of communication reflects two assumptions:

1. a sender exists that can assemble a meaningful message.

2. a qualified receiver exists to disassemble the message.

Between a number of entities these assumptions are complicated by the potential of one-to-one or one-to-many transmission events. In one-to-one (or one-to-few) communication, the sender must be able to discriminate between potential receivers (e.g. through frequency selection, message tagging, encryption, etc.). To do so requires that the sender must maintain a *model* of the intended receiver (e.g the receivers frequency, identifier, or encryption key). In one-to-many communication, this need not be the case, the sender need only maintain a common communication standard (e.g. through a common frequency, message identifier, or encryption algorithm). From this basic description some definitions may be proposed that assist in the characterization of agent arbitration (and, later, multiagent coordination) strategies.

First, formalizing the concept of a message as a formal representation built out of some consistent language (here drawing briefly from elementary formal-language theory [Cormen, 1990]):

Definition: D4.9 (Message) *Given*

- *a set of symbols or* alphabet, Σ, *and*
- *a set of strings composed of the alphabet Σ or language, $L \subseteq \Sigma$,*

a message *is simply an element of* $\mathbf{m} \in L$.

Thus a message can range from a lone 8 bit character to an elaborate set of data structures, the meaning of which is determined by mutual agreement of sender and receiver.

Transmission of a message is simply the transfer of an interpretation of a representational structure from one entity to another or

Definition: D4.10 (Transmission) *Given a sender, S, a receiver, R, and a message, m, a* transmission *of m is the act of conveying m from S to R through some medium.*

The correct assembly of the message is the duty of the sender while interpretation of the same message is the duty of the receiver. Transmission could be as basic as variable sharing within a single program to interprocess communication between hosts (i.e. independent computers) over some network connection.

A sender may possess the ability to discriminate a subset of receivers from a field of candidates and transmit to them alone – an act of one-to-one or one-to-few communication, here termed directed transmission:

Definition: D4.11 (Directed Transmission) *Given a*

- *sender, S,*
- *a set of receivers, $\mathbf{R}_j : 1 \leq j \leq N_r$,*
- *a message, $m \in L$,*

then if S somehow identifies a unique receiver $d \in N_r$ and transmits m to $\mathbf{R}_d$, the transmission of m is a directed transmission *and the message m is* private.

In other words there exist messages for which the sender must know the identity of the receiver prior to transmission. Clearly, private messages exploit particular capabilities within particular receivers to interpret and/or act upon a message. So the act of direct transmission requires that the sender refers to an internal model of the receiver to both identify the receiver and compose a message that the receiver will understand. Within a multiprocess agent (i.e. an agent composed of many processes), the implications of such actions are obvious, the sending process must know the location of the receiving process (e.g. a UNIX socket or memory address) and must send a legible message (e.g. a byte format). In single process agents, direct transmission amounts to explicit variable sharing between functions or objects.

Conversely, it is possible the sender does not know the identity of the receiver(s) prior to transmission in which case the content of the message must adhere to some common or public standard, i.e. a *broadcast transmission*:

Definition: D4.12 (Broadcast Transmission) *Given a*

- *sender, S,*
- *a group of receivers, $R_j : 1 \leq j \leq N_r$,*
- *a message, $m \in L$,*

then if S does not identify a unique receiver and transmits m to all $R_j : 1 \leq j \leq N_r$, the transmission of m is a broadcast transmission *and the message m is* public.

This definition implies that a public message contains data that *may* be translated and/or acted upon by any or all receivers. These definitions can now be applied to the classification of agent arbitration and control strategies.

Mataric [Mataric, 1994] offers similar definitions of *directed communication* but does not distinguish between one-to-one and one-to-many. Interestingly, she defines *indirect communication* as act of one agent observing another's behaviour, known as *stigmergic* communication in biology.

4.3.2 Arbitration

The combination of behaviours to form a composite behaviour is commonly referred to as *arbitration*. Many strategies have been discussed in literature from subsumption [Ferrell, 1993] and task priority[Nakamura, 1984] to weighted summation[Rosenblatt, 1995a] and discrete event systems [Kosecka, 1994]. No single technique is definitively more applicable to autonomous operation than any other, though subsumption has enjoyed wider application than most arbitration techniques.

Behaviour arbitration usually falls between concurrent behavioural combination and serial temporal sequencing. Between these extremes, an arbitrator *combines* the output of *concurrent* behaviours, changing the mixture over time. However, most arbitrators occupy the extreme regions of this design spectrum, resulting in some common arbitration strategies. Mixtures of these systems generally employ combination in the basic controllers and build meta controllers through switching [Mataric, 1994]. Colombetti et al [Colombetti, 1996] supply a classification notation descriptive of many multiagent systems:

- *Independent Sum.* Two or more behaviours (e.g.α and β) act independently (i.e. on different actuators):

$$\alpha|\beta \tag{4.88}$$

- *Combination.* Two or more behaviours are combined into a new behaviour on a single actuator:

$$\alpha+\beta \tag{4.89}$$

- *Suppression.* One behaviour inhibits another, not necessarily on the same actuator:

$$\frac{\alpha}{\beta} \tag{4.90}$$

- *Sequence.* A behavioural pattern is built as a sequence, σ, of simpler behaviours, again not necessarily on the same actuator:

$$\sigma = \alpha \cdot \beta \cdot \ldots \tag{4.91}$$

  and a repeating pattern may be suffixed or $\sigma*$.

In the transformation of controller output, $\mathbf{u}_i$, into composite output, $\mathbf{u}_c$, all arbitration algorithms draw from one or more of these tools. As mentioned earlier, the integration of arbitration and control varies considerably in practice. Some arbitrators rely on a centralized model to select a specific controller. Others broadcast a single objective to the controller set and rely on a voting or summation strategy to determine the appropriate response. Finally, some arbitrators rely exclusively on environmental interaction to select the appropriate controller. Based on these observations the following classes of arbitration are proposed:

- explicit arbitration
- implicit arbitration
- emergent arbitration

Explicit arbitration transforms $\mathbf{r}_c$ into a set of subgoals,$\mathbf{r}_i$, executed by the controllers, $\mathbf{H}_i$, as directed by the arbitrator. *Implicit arbitration* occurs when each controller determines the appropriate response to the composite goal without direction from the arbitrator. Finally *emergent arbitration* occurs when the controllers drive the plant towards $\mathbf{r}_c$ based purely on environmental interaction and without the application of an externally defined goal system.

The communication definitions developed earlier can now be applied to form definitions that formally characterize these arbitration approaches:

Definition: D4.13 (Explicit Arbitration) *If an agent,* A, *with behaviour controllers* $\mathbf{H}_i(t, \cdot, \mathbf{r}_i(t), \mathbf{v}(t)) : 1 \leq i \leq b$

1. directly transmits $\mathbf{r}_i(t)$ *to controllers* $\mathbf{H}_i(t, \cdot, \mathbf{r}_i(t), \mathbf{v}(t))$
2. *combines the output* $\mathbf{u}_i(t)$ *to form a composite output* $\mathbf{u}_c(t)$
3. *achieves a stable trajectory* $\mathbf{y}_c(t)$ *about* $\mathbf{r}_c(t)$

A *exercises control through* explicit arbitration *or simply explicit control.*

In effect, the agent determines which controller can best achieve the desired behaviour. To do so requires that the arbitrator maintain an accurate model of the participating controllers and that the desired behaviour can be explicitly decomposed into controller setpoints. The direct transmission of setpoints $\mathbf{r}_i(t)$ is symptomatic of model maintenance since each setpoint is correlated with each controller. An example of explicit arbitration is common in manipulation control where a *supervisory controller* determines the necessary setpoints for each link controller. Since each setpoint is correlated with a particular link controller, setpoints must be transmitted directly to each controller (often through a simple `moveto` procedure call).

The less strict implicit arbitration permits each controller to determine suitable setpoints, autonomously:

Definition: D4.14 (Implicit Arbitration) *If an agent,* A, *with behaviour controllers* $\mathbf{H}_i(t, \cdot, \mathbf{r}_c(t), \mathbf{v}(t)) : 1 \leq i \leq b$

1. broadcasts *the composite goal* $\mathbf{r}_c(t)$ *to controllers* $\mathbf{H}_i(t, \cdot, \mathbf{r}_c(t), \mathbf{v}(t))$,
2. *combines of the output* $\mathbf{u}_i(t)$ *to form a composite output* $\mathbf{u}_c(t)$
3. *achieves a stable trajectory* $\mathbf{y}_c(t)$ *about* $\mathbf{r}_c(t)$

A *exercises control through* implicit arbitration *or simply implicit control.*

Again broadcast information transmission is symptomatic of a relatively model-free arbitrator. With a broadcast goal, there is no explicit one-to-one correspondence between goal and setpoint assignment by the agent. In effect, the agent leaves the applicability of a given control strategy to the controller, and arbitrates between the results using some selection criteria (e.g. votes, etc.). Implicit arbitration is rare in traditional control but common in daily activity. Consider the combat pilot's task of formation flying. The flight leader does not specify a trajectory for each wingman (in fact he may not know how many wingmen are present), but relies on the wingmen's ability to track the flight leader and maintain intervals autonomously.

Emergent arbitration permits each controller to respond purely to environmental stimuli -- the environment becomes the arbitrator:

Definition: D4.15 (Emergent Arbitration) *If an agent,* A, *with behaviour controllers* $\mathbf{H}_i(t,\cdot,\mathbf{r}_i(t),\mathbf{v}(t)) : 1 \leq i \leq b$

1. only *combines the output* $\mathbf{u}_i(t)$ *to form a composite output* $\mathbf{u}_c(t)$
2. *achieves a stable trajectory* $\mathbf{y}_c(t)$ *about* $\mathbf{r}_c(t)$

A *exercises control through* emergent arbitration *or simply emergent control.*

With no arbitrator to controller transmissions, an emergent arbitrator becomes a simple algebraic function or switching protocol of stimulus driven controller responses. Emergent controllers rely on a complex interaction of three dynamic systems to achieve the desired behaviour: the environment, the combination mechanism, and the controllers. Though this last form may seem somewhat contrived, in fact it represents a very common method of generating useful composite behaviours such as the wall following combination: **avoid-obstacle** and **move-ahead** [Arkin, 1987]. By fixing the arbitrator as a finite state machine switched between active behaviours, this stimulus driven controller combination will drive the system parallel to obstructing walls.

This classification strategy dissects predictive modelling and planning out of arbitration. The specification of setpoints based on a centralized model is a form of a priori arbitration, implying that a monolithic internal representation of the world is sufficiently accurate to plan future actions. By classifying setpoint transmissions as directed or broadcast, arbitration can, therefore, be categorized as explicit or implicit.

If no a priori setpoints are transmitted, then arbitration must be emergent or purely stimulus driven. Emergent arbitration does not preclude feedforward or predictive models within each controller, rather it limits their use to specific, narrowly defined problems. Similarly, models within an emergent arbitrator (i.e. those based solely on controller responses) are limited to action selection alone.

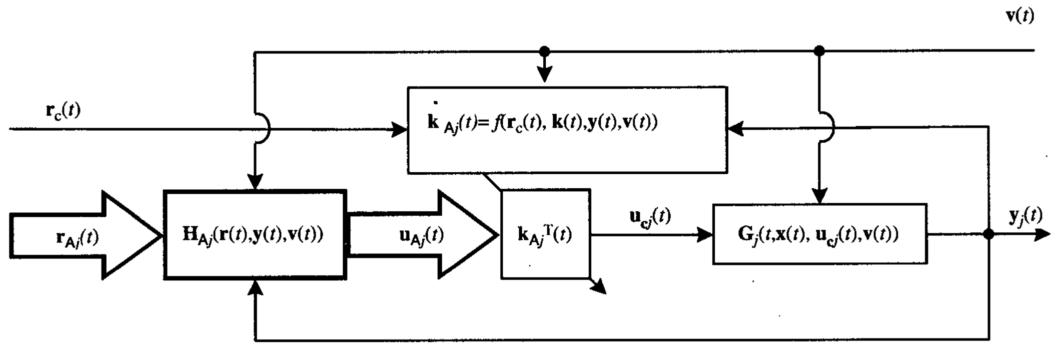


Figure 4.19: A simple linear agent arbitration model.

In theory, the spectrum of *explicit*, *implicit*, through *emergent* arbitration represents a migration from centralized to decentralized modelling; relatively fragile single threaded control to relatively robust parallelism; and reliance on computational determinism to dependence on environmental/controller interaction.

In practice, agent-like systems often exhibit one or more of these arbitration strategies. In other words, explicit, implicit and stimulus driven arbitrators may operate concurrently within a given agent.

4.3.3 Example: A Linear Arbitration Model

Despite the significance of the arbitrator in most real time systems, few choose to explicitly describe the arbitrator as part of the control system, preferring to classify the arbitrator as a computing and not a control element. To explore arbitration more deeply, a simple agent model can be presented that distinguishes the control from the arbitration while retaining a control theoretic representation.

Consider a model of A in which the output of m controllers are linearly combined to form a composite control action or:

$$\mathbf{u}_c(t) = \mathsf{A}(\mathbf{r}(t), \mathbf{y}(t), \mathbf{v}(t), \mathbf{u}(t)) = \mathbf{k}^T(t)\ \mathbf{u}(t, \mathbf{r}(t)) \tag{4.92}$$

where $\mathbf{k}(t) = [k_1(t), \ldots, k_b(t)]^T$ is a set of b gains and $\mathbf{u}(t) = [\mathbf{u}_1(t), \ldots, \mathbf{u}_b(t)]^T$.

Since the gain vector represents any class of arbitrating mechanism, from finite state machine to continuous combinatorial system, $\mathbf{k}$ should evolve as a nonlinear function of environmental events,$\mathbf{v}(t)$, time and possibly its current state:

$$\dot{\mathbf{k}}(t) = f(t, \mathbf{k}(t), \mathbf{r}(t), \mathbf{y}(t), \mathbf{v}(t)) \tag{4.93}$$

as depicted in fig. 4.19. Though $f(\cdot)$ has been chosen to map continuous time to a continuous domain $\mathbf{k}(t)$, plainly many arbitrators use discrete domains and discrete time measures [Zhang, 1995]. Indeed, the evolution of $\mathbf{k}(t)$ might be defined as a function of either a finite state machine (discrete time and domain), a difference equation (discrete time and continuous domain), an asynchronous circuit (continuous time and discrete domain), or a differential equation (continuous time and continuous domain). Recalling Colombetti's [Colombetti, 1996] composition operators, the outputs of controllers $\mathbf{H}_\alpha(\cdot)$ and $\mathbf{H}_\beta(\cdot)$, $\mathbf{u}_\alpha$ and $\mathbf{u}_\beta$ respectively might be combined through linear arbitration vectors as:

- *Independent Sum*:
$$\begin{bmatrix} \mathbf{u}_{c\alpha}(t) \\ \mathbf{u}_{c\beta}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u}_\alpha \\ \mathbf{u}_\beta \end{bmatrix} \tag{4.94}$$

- *Combination*:
$$\mathbf{u}_c(t) = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u}_\alpha \\ \mathbf{u}_\beta \end{bmatrix} \tag{4.95}$$

- *Suppression*:
$$\mathbf{u}_c(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_\alpha \\ \mathbf{u}_\beta \end{bmatrix} \tag{4.96}$$

- *Sequence*:
$$\mathbf{u}_c(t)(t) = \mathbf{k}^T(t) \begin{bmatrix} \mathbf{u}_\alpha \\ \mathbf{u}_\beta \\ \vdots \end{bmatrix} \tag{4.97}$$
$$\mathbf{k}^T(t) = \delta(\mathbf{k}(t), t) = \begin{cases} \begin{bmatrix} 1 & 0 & \ldots \end{bmatrix} & t < t_\beta \\ \begin{bmatrix} 0 & 1 & \ldots \end{bmatrix} & t \geq t_\beta \\ \vdots & \end{cases} \tag{4.98}$$

  where $\delta(\cdot)$ is the delta function indicating a discrete change in state. Therefore $\mathbf{k}$ is a temporal sequencer or finite state machine within which mutual suppression, combination and/or independent action can be expressed.

Now if $\mathbf{r}_i$ is based on a centralized model, it must be transmitted directly to each agent, symptomatic of explicit arbitration. If $\mathbf{r}(t)$ is unknown a priori and broadcast to all agents, then arbitration is implicit. Finally, if $\mathbf{r}_i(t)$ is internal to each controller, arbitration must be emergent.

Equivalency

Most of the models described here use elementary combination or sequencing. Indeed, Arkin's motor schemas, Steels' behaviours, Van de Panne's SAN, and the DAMN architecture are plainly behaviour summation strategies. In contrast, subsumption and DES are plainly discrete, using finite or augmented finite state machines to evolve **k** over time (e.g. [Mahadevan, 1992]).

4.3.4 From lone Agents to Agent Teams

During the design evolution of a complex single agent system, it is conceivable that individual controllers within an agent might themselves become sufficiently complex that they acquire the attributes of agents. Indeed, many multiagent systems are composed agent hierarchies, in which a set of agents operate within a larger agent (e.g. subsumption).

Alternatively, tasks may exist that are practically impossible to fulfill with a single agent (e.g. lacking time, energy, or degrees of freedom). In either case, close examination of the goal system and the available resources may suggest more than one agent is necessary to achieve the desired behaviour.

Not surprisingly, real time constraints makes centralized control of agent teams impractical. For this reason, the coordination and control of agent based robot teams has become a growing area of investigation and, as we shall see in the next section, shares many of the attributes and definitions of agent based control systems.

4.4 A General Model of Multiagency

In the previous section, definitions of behaviour, control, behaviour control, and agents were established and an Agent Control hypothesis proposed. Within a team these definitions remain largely unchanged, the team is composed of a set of plants each perturbed by disturbances and controlled by an agent controller. The difference and advantage that homogeneous teams of agents hold over lone agents is that the goal trajectory need not be reachable to all team members all the time. If members cooperate or compete to fulfill the goal trajectory, a degree of robustness and flexibility can be afforded.

The behaviour of a multiagent team, or *global behaviour*, is an *aggregation* of every agents' *local* behaviour in the team. *Global* goals are, of course, the desired behaviour of the agent team and are analogous to *composite goals* in single agent systems.

Achieving a global goal is more complex than a composite goal, since controller *arbitration* within a single agent is less complex than behaviour *coordination* amongst an agent team. As observed in Mataric [Mataric, 1994], the size of a discrete state space balloons from s states for a lone agent, to s^a for a team of a identical agents. Just as in the lone agent case, information exchange governs the classifications of explicit, implicit, and emergent coordination.

Historically, the ideal global behaviour arises from emergent coordination, in which a set of agents produces the desired behaviour through interagent dynamics and environmental interaction. Since both behaviours and the manner in which they are combined (e.g. temporally, algebraically, etc) greatly influence the output of such systems, the design of local behaviours becomes critically important. Without an explicit global goal definition, local behaviour has historically been an exercise in trial and error. Thus *emergent* behaviour is as potentially powerful is it is difficult to implement.

Therefore, this thesis attacks the multiagent control problem assuming an explicit a priori global goal statement. This approach is supported by two arguments. The first is that many tasks exist that must be executed with some predictable guarantees of stability and precision. The second is that through a top down *analysis* and *decomposition* of a global goal system, a clearer understanding of multiagent dynamics might be achieved. Given this investigation strategy,the following sections must again address structure and nomenclature in a manner consistent with foregoing definitions, in particular:

- What is global behaviour?
- What is a global goal?
- What is the structure of a multiagent system?
- How are global behaviours derived from local behaviours?
- How are local goals derived from global goals?

4.4.1 Local Behaviour

In a multiagent system, a group of agent controllers generate behaviours through their respective plants. Recalling that each agent arbitrates between b local controllers, $\mathbf{H}_i : 1 \leq i \leq b$ we can define *local* behaviour in a familiar form:

Definition: D4.16 (Local Behaviour) *Consider the jth subplant, $\mathbf{G}_j : 1 \leq j \leq N$, within a composite system perturbed by both control effort, $\mathbf{u}_{ij}(t) : 1 \leq i \leq b$, and environmental disturbance $\mathbf{v}_j(t)$. Then the plant's* local behaviour *is the* observed *output, $\mathbf{y}_j(t) \in \mathcal{B}_j$.*

In a flock of boids, for example, the trajectory of boid i is the local behaviour of the boid i. Flock behaviour, on the other hand, is clearly an aggregate characteristic or *global behaviour* of the collection of boids.

4.4.2 Global Behaviour

In [Mataric, 1994], global behaviour is termed *ensemble*, *collective*, or *group* behaviour and is defined loosely as:

"...an observer defined temporal pattern of interactions between multiple agents."

If the dynamics of a team of agents were enclosed within a 'black box', the system's behaviour could be defined through definition D4.2 as the behaviour of the black box and the structure would be indistinguishable from figure 4.14. Thus the definition of Global Behaviour is simply an extension of D4.2:

Definition: D4.17 (Global Behaviour) *Given plants,* $\mathbf{G}_j : 1 \leq j \leq N$*, perturbed by control efforts,* $\mathbf{u}_{ij}(t) : 1 \leq j \leq N$*, and disturbances* $\mathbf{v}_j(t) : 1 \leq j \leq N$*,* global behaviour *is the* observed *output,* $\mathbf{y}(t) : \mathrm{I\!R}^x \times \mathrm{I\!R}^u \times \mathrm{I\!R}^+ \rightarrow \mathrm{I\!R}^b$ *of the aggregate system:*

$$\mathbf{y}(t) = \mathbf{G}(t, \mathbf{x}(t), \mathbf{u}_c(t), \mathbf{v}(t)) \tag{4.99}$$

where $\mathbf{x}(t) = [\mathbf{x}_1(t), \ldots, \mathbf{x}_N(t)]^T$ *and* $\mathbf{u}_c(t) = [\mathbf{u}_{c1}(t), \ldots, \mathbf{u}_{cN}(t)]^T$ *and the state of the system evolves according to some relation:*

$$\dot{\mathbf{x}}(t) = \mathbf{f}[t, \mathbf{x}(t), \mathbf{u}_i(t), \mathbf{v}(t)], \quad \forall t \geq 0 \tag{4.100}$$

and a definition of the global behaviour space may be similarly defined as:

Definition: D4.18 (Global Behaviour Space) *If the set of all possible output trajectories,* $\mathbf{y}(t)$*, are bounded within a region,* $\mathcal{B}$*, then* $\mathcal{B}$ *is the behaviour space of* $\mathbf{y}(t)$*.*

Observing a large flock of birds, global behaviour is often observable as a flockwide change in direction and the behaviour space, possibly a complex mixture of gross motion and flock shape.

To the observer, a multiagent system may appear to behaves as a single 'black box' plant, $\mathbf{G}$ (i.e. at great distance a flock may appear as a coherent mass). On closer inspection the behaviour exhibited by the system arises from the combination of agent action and interaction (or birds in a flock). Thus the black box structure of definition D4.2 and depicted in figure 4.20 is more precisely described in the complex structure

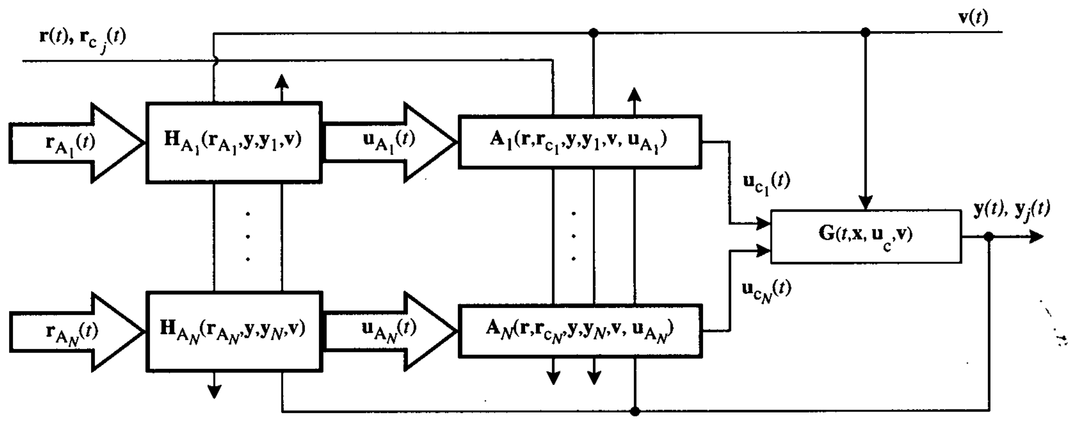

$\mathbf{r}(t), \mathbf{r}_{c_j}(t)$
$\mathbf{v}(t)$
$\mathbf{r}_{A_1}(t)$
$\mathbf{H}_{A_1}(\mathbf{r}_{A_1},\mathbf{y},\mathbf{y}_1,\mathbf{v})$
$\mathbf{u}_{A_1}(t)$
$\mathbf{A}_1(\mathbf{r},\mathbf{r}_{c_1},\mathbf{y},\mathbf{y}_1,\mathbf{v},\mathbf{u}_{A_1})$
$\mathbf{u}_{c_1}(t)$
$\mathbf{G}(t,\mathbf{x},\mathbf{u}_c,\mathbf{v})$
$\mathbf{y}(t), \mathbf{y}_j(t)$
$\mathbf{u}_{c_N}(t)$
$\mathbf{r}_{A_N}(t)$
$\mathbf{H}_{A_N}(\mathbf{r}_{A_N},\mathbf{y},\mathbf{y}_N,\mathbf{v})$
$\mathbf{u}_{A_N}(t)$
$\mathbf{A}_N(\mathbf{r},\mathbf{r}_{c_N},\mathbf{y},\mathbf{y}_N,\mathbf{v},\mathbf{u}_{A_N})$

Figure 4.20: A multiagent controller model. Note that each agent acts upon the global plant $\mathbf{G}$, though each agent may apply control effort to only a portion of the plant $\mathbf{G}_j$.

of figure 4.21 in which the global behaviour is the product of some binding *aggregate relation* applied to agent behaviours [3]. Drawing again on the flock of birds, the aggregate relation is some function that relates the properties of each individual bird to the behaviour of the flock (e.g. bird position and flock centroid).

Definition: D4.19 (Aggregate Relation) *Given the behaviour of N agents, an aggregate relation is a function that maps the set of behaviours $\mathbf{y}_j(t) : 1 \le j \le N$ of the combined system into an observable global behaviour $\mathbf{y}(t)$ in some global behaviour space $\mathcal{B}$ or:*

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{y}_1(t), \ldots, \mathbf{y}_N(t)) \tag{4.101}$$

Thus, aggregate relations express constraints on the behaviour of a multiagent system, relating the behaviour of individual plants to the behaviour of the global plant. Such constraints can be physical equality or inequality constraints (e.g. end effector position, obstacle boundaries, etc.) or performance constraints (e.g. minimum time or minimum energy). For example, *holonomic* constraints on a multiagent system of dimension N described in some coordinate system η can be expressed through the equality constraint:

$$\mathbf{y} - \mathbf{f}(\eta_1, \ldots, \eta_N) = 0 \tag{4.102}$$

Such expressions are greatly simplified through the adoption of a set of generalized coordinate systems. An aggregate relation describing the behaviour of the system, $\mathbf{y} \in \mathcal{B}$, and the *generalized coordinate* space of the

[3]Note that it is not apparent in this figure that as each subplant $\mathbf{G}_j(t)$ generates behaviour $\mathbf{y}_j(t)$, it may also disturb other subplants in the system. The term, $\mathbf{v}(t)$, however, embodies all such disturbances.

ith of N agents, $\mathbf{y}_j \in \mathcal{B}_j$, are often related by a nonlinear transformation $\mathbf{f} : \mathcal{B}_1 \times \ldots \times \mathcal{B}_N \to \mathcal{B}$ or:

$$\mathbf{y} = \mathbf{f}(\mathbf{y}_1, \ldots, \mathbf{y}_N) \tag{4.103}$$

and the coordinates $\mathbf{y}_j$ are independent of the constraints.

For example, the cartesian position of a manipulator end effector, $\mathbf{y}$ is clearly a function of the cartesian positions of the manipulator's links forming the constraint:

$$\mathbf{y} - \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_N) = 0 \tag{4.104}$$

Since the ith joint's cartesian position, $\mathbf{x}_i$, is a function of joint displacements, this constraint can be further simplified through configuration space generalized coordinates,$\eta_j = q_j$ or:

$$\mathbf{y} - \mathbf{f}(\mathbf{q}_1, \ldots, \mathbf{q}_N) = 0 \tag{4.105}$$

the forward kinematic solution. The role of the aggregate relation is depicted in figure 4.21. Many examples can be drawn such as:

- *mobile robot troops* local robot trajectories combines to form troop formations.
- *legged walkers* local leg trajectories combines to form gate patterns.
- *vehicle control* individual motor output combines to form vehicle thrust.

Of course, there exist systems for which establishing a desired behaviour is trivial in comparison to generating the aggregate relation[4] necessary to describe the aggregate behaviour.

4.4.3 Multiagent Coordination

From the definition of aggregate relation, any function of agents' behaviours is an acceptable aggregate relation. Thus, given the local behaviours of an agent team, aggregate relations describe a global behaviour. To achieve a *desired* global behaviour requires that local behaviours are organized into *coordinated* behaviour. More precisely:

Definition: D4.20 (Coordinated Behaviour) *If agents* $\mathsf{A}_j : 1 \leq j \leq N$ *demonstrate a global behaviour,* $\mathbf{y}(t)$*, stable about* $\mathbf{r}(t)$*, given a set of local composite goals* $\mathbf{r}_{cj} : 1 \leq j \leq N$ *then the agents exhibit* coordinated global behaviour *through some aggregate relation.*

[4]e.g. Stewart platforms [Fichter, 1996] a parallel mechanism in which a known platform position easily specifies leg lengths but known leg lengths do not easily specify platform position.

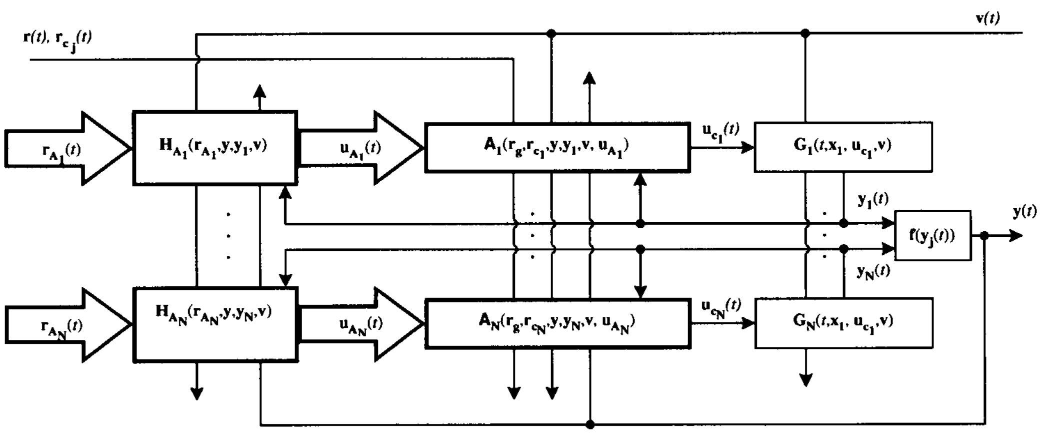


Figure 4.21: A decomposed model of a multiagent controller. Though each agent acts upon a local plant $\mathbf{G}_j$, disturbances dynamically bind plants to form a single global plant $\mathbf{G}$. The aggregate relation $\mathbf{f}(\cdot)$ represents the binding between the local and global behaviour states $\mathbf{y}_j$ and $\mathbf{y}$ respectively.

Just as an agent's arbitration strategy may be classified into either explicit, implicit, or emergent, so too, can multiagent coordination. The definitions of behaviour, D4.2, and global behaviour, D4.17 are virtually identical and the control of such system raises similar issues of centralized and decentralized arbitration. Indeed, it is this very duality that allows for single agent systems to be decomposed into multiagent systems.

In *explicit coordination*, a centralized process composes a unique instruction set for the jth $\mathbf{r}_{c_j}(t)$ of N agents in a multiagent system or:

Definition: D4.21 (Explicit Coordination) *If agents,* $\mathsf{A}_j : 1 \leq j \leq N$, *achieve coordinated behaviour through direct transmission (i.e. private message) of composite goals* $\mathbf{r}_{cj}(t)$ *then agents* $\mathsf{A}_j : 1 \leq j \leq N$ *exhibit control through* explicit coordination.

In order to collate goal messages and receivers prior to a directed transmission, the centralized sending process must possess a description or model of the receiving process. Explicit coordination identifies a particular agent receiver as capable of understanding and acting upon a goal message of a particular type.

For example, air traffic controllers assign unique instructions to each aircraft (eg. altitude, heading, take off and landing order, etc.) on approach or take off, but do not necessarily instruct each aircraft how to achieve these goals (e.g. stick/rudder/throttle positions). To achieve such explicit coordination requires an accurate centralized modelling system (e.g. radar imaging, transponders, air to ground communication, etc.).

Implicit coordination, however, disposes of a centralized model of the multiagent system while retaining a centralized goal generation process. Such coordination requires that each agent is able to interpret and act locally to pursue some form of global goal expression. Furthermore, implicit coordination suggests that the goal generation process does not maintain a model of each receiver, permitting the same message to be broadcast to any number of receivers.

Definition: D4.22 (Implicit Coordination) *If agents, $\mathsf{A}_j : 1 \leq j \leq N$, achieve coordinated behaviour through broadcast transmission (i.e. a public message) of a goal, $\mathbf{r}_c(t)$, then the agents A_j exhibit control through* implicit coordination.

Again drawing on air traffic control, air strikes in ground support are sufficiently uncertain to render explicit air traffic control impossible (however desirable). Though general directives (e.g. targeting, safe flight corridors, etc.) may be supplied by ground and air controllers, individual aircraft behaviour is determined by pilots as conditions warrant – an example of implicit coordination.

Given sufficient communication resources any single agent system could be decomposed into an explicitly coordinated multiagent system. In such systems a centralized process engages in bilateral information exchanges to achieve the global goal. Implicit systems presumably require lower bandwidth, unilateral communications between the goal generation and agent processes. In *emergent coordination*, the ideal system falls 'naturally' into the desired goal state through interagent dynamics:

Definition: D4.23 (Emergent Coordination) *If agents, $\mathsf{A}_j : 1 \leq j \leq N$, achieve coordinated behaviour, $\mathbf{y}(t)$, through* neither *directed* nor *broadcast transmissions of a goals $\mathbf{r}_{c_j}(t)$ or $\mathbf{r}_c(t)$ respectively, then the agents $\mathsf{A}_j : 1 \leq j \leq N$ exhibit control through* emergent coordination.

In a dogfight, air traffic is still less structured and pilots often must act independently based on local observation and experience (and rarely with external air traffic control) to achieve local air superiority. Since air traffic models are necessarily local to each pilot, this global objective is often difficult to observe, but is achievable with sufficient training and equipment.

In reality, multiagent systems often exhibit the full spectrum of coordination methods. Agents within multiagent systems have been known to communicate to negotiate explicit local plans and pursue broadcast objectives, while exploiting reactive behaviours (e.g. [Parker, 1992a]) – all of which support the global objective.

In Mataric's PhD thesis [Mataric, 1994], similar though not identical definitions are used to describe *cooperation* [5] that often requires directed communication to assign particular tasks to the participants. *Explicit* cooperation is defined as:

> "...a set of interactions which involve exchanging information or performing actions in order to benefit another agent"

Implicit cooperation is defined as

> "...consists of actions that are part of the agents own goal achieving repertoire, but have effects in the world that help other agents achieve their goals."

Implicit cooperation has more in common with emergent coordination in that global behaviour arises from local goal achieving behaviour.

4.4.4 Global Goals

Thus far the properties of global behaviour and multiagent coordination have been discussed, often with passing reference to a *global goal.* Global behaviour is, in general, the result of *team* behaviour. Global goals follow this definition by specifying a *desired* global behaviour. A necessary condition in the composition of a global goal is (as in the single agent case), *reachability*:

Definition: D4.24 (Globally Reachable Goal) *Given a global behaviour space, $\mathcal{B}$, and a global goal space, $\mathcal{G}$, a goal $\mathbf{r}(t) \in \mathcal{G}$ is globally reachable if $\mathbf{r}(t) \in \mathcal{G} \cap \mathcal{B}$*

Given that agents can only pursue *locally reachable goals* (D4.4), for a goal to be Globally reachable it must be *locally* reachable for at least one agent throughout the interval of the global goal. Therefore, one *possible* definition of global goal is:

Definition: D4.25 (Global Goal (basic)) *If a trajectory $\mathbf{r}(t) \in \mathcal{G}$ is locally reachable by two or more agents then the trajectory is a* global goal.

Now recall that agent behaviour spaces $\mathcal{B}_j : 1 \leq j \leq N$, are mapped into $\mathcal{B}$, the global behaviour space, through the aggregate relation, $\mathbf{f}(\cdot)$. Acting in isolation, the behaviour of the jth agent when projected through the aggregate relation, prescribes an *agent behaviour subspace*, $\mathcal{B}^j$, within $\mathcal{B}$. In other words, at any

[5] [Mataric, 1994]:p. 23, paragraphs 4 and 5

instant there exists a finite region of $\mathcal{B}$ reachable by the jth agent alone. The union of the N agent subspaces $\mathcal{B}^j$ is the global behaviour space or $\mathcal{B} = \bigcup_{j=1}^{N} \mathcal{B}^j$. Defining $\mathbf{y}_{\mathsf{nA}} = [\mathbf{y}_1 \cdots \mathbf{y}_N]^T$, the rate of change of global behaviour with respect to local behaviours may be expressed as:

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}(\mathbf{y}_{\mathsf{nA}})}{\partial \mathbf{y}_{\mathsf{nA}}} \dot{\mathbf{y}}_{\mathsf{nA}} \tag{4.106}$$

$$\dot{\mathbf{y}} = \mathbf{J}(\mathbf{y}_{\mathsf{nA}}) \dot{\mathbf{y}}_{\mathsf{nA}} \tag{4.107}$$

where $\mathbf{J}(\mathbf{y}_{\mathsf{nA}})$ is the Jacobian of the aggregate relation. Given bounds on the local behaviour 'velocity' the instantaneous extent of an agent's global 'velocity' subspace.

To achieve the desired global behaviour, the global goal must be decomposed into local goals through an *inverse* of the aggregate relation from the global space into the agent's local goal spaces $\mathbf{r}_{c\mathsf{nA}}(t) = [\mathbf{r}_{c1}(t) \ldots \mathbf{r}_{cN}(t)]^T$:

$$\mathbf{r}_{c\mathsf{nA}}(t) = \mathbf{g}(\mathbf{r}(t)) \tag{4.108}$$

and $\mathbf{g}(\cdot) : \mathcal{G} \to \mathcal{G}_1 \times \ldots \times \mathcal{G}_N$. Ideally, $\mathbf{g}(\cdot)$ would be a simple inverse of the aggregate relation or:

$$\mathbf{g}(\mathbf{r}(t)) = \mathbf{f}^{-1}(\mathbf{r}(t)) \tag{4.109}$$

For example, in resolved motion position control, a geometric inverse of the forward solution (an aggregate relation) is an example of an inverse aggregate relation. However, aggregate relations may not be easily invertible. There may be too few or too many agents available to exactly achieve the desired global goal.

Recalling Samson's task function approach, it is clear that, just as in manipulation end effector tasks, global goals must be instantaneously *feasible*. So a more stringent definition of global goal can be proposed:

Definition: D4.26 (Global Goal) *Given a trajectory* $\mathbf{r}(t) \in \mathcal{G}$, *local behaviours* $\mathbf{y}_j : \mathbf{y}_1 \leq j \leq \mathbf{y}_N$, *and an aggregate relation* $\mathbf{y} = \mathbf{f}(\mathbf{y}_1 \ldots \mathbf{y}_N)$, *if* $\mathbf{r}(t)$ *is feasible through* $\mathbf{f}(\cdot)$ *then the trajectory is a* global goal.

If the aggregate relation is invertible, explicit coordination can use a one-to-one mapping to decompose the global goal trajectory into a unique set of trajectories for each agent. In implicitly coordinated systems, the inverse aggregate is distributed amongst the agents, each agent determining autonomously the appropriate contribution to the global goal. Finally in emergent control, an inverse aggregate is unnecessary, since the system falls naturally into the desired global behaviour.

Regardless of the coordination method, one might argue that the perfect multiagent system contains exactly the right number and configuration of agents to produce the desired global behaviour (i.e. the aggregate is invertible). However, it is often beneficial for a multiagent system to contain agents in excess of the global goal's requirements. In this case the inverse aggregate is not invertible requiring the extension of an explicitly coordination system with additional rules to distribute subgoals between 'redundant' agents. So while redundant agent teams complicate explicit coordination, they provide flexibility and robustness to multiagent systems and encourage the adoption of implicit or emergent coordination methods. Indeed robustness through redundancy is a key attribute of many published multiagent systems (e.g. [Mataric, 1994, Parker, 1992a]).

4.4.5 Multiagent Control

It has been demonstrated by a number of investigators [Mataric, 1994, Reynolds, 1987, Parker, 1992a] that through the arbitration of multiple local control strategies, each agent within a population of such agents can contribute to useful global behaviour with little or no modelling of the entire system. The foregoing definitions provide the necessary components to construct a second hypothesis that characterizes the intent of *multiagent* control control systems:

Hypothesis: H2 (Multiagent Control) *Given a global goal, $\mathbf{r}(t)$, a set of Agents, $\mathsf{A}_j : 1 \leq j \leq N$, can be designed such that the global behaviour of the system, $\mathbf{y}(t)$, is stable about $\mathbf{r}(t)$.*

Encompassing both implicit and explicit varieties of coordinated agent behaviour, figure 4.20 and hypothesis H2 imply the existence of some form of goal statement $\mathbf{r}(t)$. However, D4.23 implies that *explicit* global goals and aggregate relations need not be specified in some systems, though desired global behaviours and aggregate relations may be observed nevertheless. [6] This is an example of emergent multiagent control, discussed in the following section.

4.4.6 Emergent Multiagent Control

Though often attributed with "something for nothing"[Mataric, 1994] qualities, emergent coordinated behaviour is attractive because it does not require a central controller to produce a global behaviour. Unfortunately, this reputation masks the considerable difficulty in designing systems that 'naturally' converge towards a desirable behaviour. Historically, the attributes of a given emergent behaviour has been subjective and rarely explicitly defined or measured. Hence the following broad hypothesis on emergent control:

[6] A good example is the Flock Centering behaviour in Reynolds Boids [Reynolds, 1987]. At no time was an explicit flock description provided, though the local goal 'attempt to stay close to nearby flockmates' produced the desired grouping behaviour.

Hypothesis: H3 (Emergent Multiagent Control) *Given a global goal,* $\mathbf{r}(t)$*, a set of Agents can be designed to exhibit emergent coordination such that the global behaviour of the system,* $\mathbf{y}(t)$*, is stable about a desired trajectory* $\mathbf{r}(t)$.

So important are the implications for redundant, fault tolerant control systems that emergent behaviour, originally an interesting side effect of multiagent interaction (e.g. [Walter, 1950b]), is now pursued as a performance objective in its own right ([Steels, 1991]). Unfortunately, the mechanisms of emergent behaviour remain largely unexplored, few investigators having modelled action and interaction of agent teams as dynamic systems.

Though condensing individual agent dynamics into a single team expression permits the exploration of multiagent dynamics, the number and variety of agent implementations, makes a generally descriptive model difficult. Nevertheless, one approach to this condensation is to adopt the linear agent model described earlier and assemble a linear *multiagent* model.

4.5 Linear Arbitration and Multiagent Control

Suppose that N agents , A_j, coexist in a multiagent system. From figure 4.20 $\mathbf{u}_c(t)$ may be defined as:

$$\mathbf{u}_{c\mathsf{nA}}(t) = [\mathbf{u}_{c_1}(t) \ldots \mathbf{u}_{c_N}(t)]^T \tag{4.110}$$

Substituting equation (4.92) for each $\mathbf{u}_{c_j}$:

$$\begin{aligned}
\mathbf{u}_{c\mathsf{nA}}(t) &= \begin{bmatrix} \mathbf{u}_{c_1}(t) \\ \vdots \\ \mathbf{u}_{c_N}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{k}_1^T(t)\mathbf{u}_{\mathsf{A}_1}(t) \\ \vdots \\ \mathbf{k}_N^T(t)\mathbf{u}_{\mathsf{A}_N}(t) \end{bmatrix} && (4.111) \\
&= \begin{bmatrix} \mathbf{k}_1^T & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{k}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\mathsf{A}_1}(t) \\ \vdots \\ \mathbf{u}_{\mathsf{A}_N}(t) \end{bmatrix} && (4.112) \\
&= \mathbf{K}_{\mathsf{nA}}(t)\mathbf{u}_{\mathsf{nA}}(t) && (4.113)
\end{aligned}$$

where $\mathbf{u}_{c\mathsf{nA}}(t)$ is the composite control vector, $\mathbf{K}_{\mathsf{nA}}$ is a *linear arbitration* matrix and $\mathbf{u}_{\mathsf{nA}}(t)$ is the agent control input vector. The condensation of a multiagent system into a single expression flattens the control hierarchy and conceals individual agent structures.

Theoretically, if an aggregate relation is defined for a multiagent system as in equation (4.103) and the agent plants, $\mathbf{G}_j$, (or global plant, $\mathbf{G}$) can be established and combined with the above linear arbitration model, the response of the system can be determined. In practice, however, this is rarely undertaken primarily due to the complexity of the (real world) plant dynamics. Thus further discussion of multiagent systems is difficult without greater understanding of the nature of the plant $\mathbf{G}(\cdot)$, the disturbances $\mathbf{v}(t)$ and aggregate relation $\mathbf{f}(\cdot)$. Clearly, the exploration of agent and multiagent systems would be facilitated by the adoption of a standard plant. The next section will do exactly this by adopting a serial manipulator as a reference plant and applying the foregoing definitions and structures to explore the multiagent control hypotheses in greater detail.

4.6 Multiagent Manipulation

A logical starting point in the development of a multiagent *manipulator* control system is to identify agents that together generate an aggregate behaviour. Fortunately, this is a problem with a simple solution. As reviewed earlier, a serial manipulator is the physical combination of a set of links - each driven by a single degree of freedom actuator. The selection of individual links as agents automatically relates a number of manipulator properties to multiagent system components.

4.6.1 Links as Agents

Recalling D4.2 and reiterating the time invariant linear link state space equation (2.50) for the jth link, we have:

$$\mathbf{G}_j : \dot{\mathbf{x}}_j = \mathbf{A}_j\mathbf{x}_j + \mathbf{b}_j\mathbf{u}_j + \mathbf{d}_j\mathbf{v}_j \tag{4.114}$$

$$\mathbf{x}_j = \begin{bmatrix} q_j \\ \dot{q}_j \end{bmatrix} \tag{4.115}$$

$$\mathbf{y}_j = \mathbf{C}_j\mathbf{x}_j \tag{4.116}$$

with control input $\mathbf{u}_j$ and disturbance $\mathbf{v}_j$. Selecting the output matrix, $\mathbf{C}_j$, as the identity matrix and assuming a direct drive robot (the gear transmission ratio in equation (2.50), $r_j = 1$), one can define the *behaviour* of the jth link as the output $[q_j \;\; \dot{q}_j]^T$. Given bounds on $\mathbf{y}_j$, the maximum and minimum joint displacements and velocities, the behaviour space $\mathcal{B}_j$ can also be prescribed. A reachable goal for the jth agent is then a trajectory that intersects this behaviour space - a joint position and velocity setpoint within the bounds of q_j and $\dot{q}_j$ respectively.

By selecting a stable controller (e.g. a PD controller with LHP characteristic roots) and providing a reachable goal trajectory, $\mathbf{r}_j(t)$, goal seeking behaviour can be observed. By providing multiple stable controllers that respond to local and global goal objectives, this link controller is transformed into a link agent.

4.6.2 Manipulators as Multiagent Systems

Collecting the link agents into a single model, the nonlinear manipulator plant can be recovered. Expressed (for example) in the linear time invariant Brunkowsy canonical form:

$$\mathrm{G}: \quad \dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{bu} + \mathbf{dv} \tag{4.117}$$

where

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} 0 & \mathbf{I} \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} \qquad \mathbf{u} = \mathbf{D}(\mathbf{q})^{-1}\tau$$

$$\mathbf{d} = \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} \qquad \mathbf{v} = -\mathbf{D}(\mathbf{q})^{-1}\left[\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \mathbf{J}^T\mathbf{f}_{\mathrm{ext}}\right] \tag{4.118}$$

$$\mathbf{y} = \mathbf{Cx} \tag{4.119}$$

Again, selecting the output matrix $\mathbf{C}$ as the identity matrix and the transmission coefficient $r_j = 1$, one can define the *behaviour* of manipulator as the output $[\mathbf{x} \ \ \dot{\mathbf{x}}]^T$. Given bounds on $\mathbf{y}$, the maximum and minimum end effector displacements and velocities, the behaviour space $\mathcal{B}$ can also be prescribed.

The definition of the aggregate relation is dependent upon the desired global behaviour. Though not directly controlled by most manipulator controllers, the end effector ultimately performs tasks expressed in cartesian coordinates. Adopting generalized coordinates for each link agent, the aggregate relation maps the agent states to the end effector coordinate, the forward kinematic solution. Reiterating equation(2.4):

$$\mathbf{y} = \mathbf{f}(\mathbf{q}) = \Pi_{j=1}^{N}\mathbf{A}_j^{j-1}(q_j) \tag{4.120}$$

If, for example, the global behaviour was the manipulator *shape*, the aggregate relation could be stated as:

$$\mathbf{y} = \mathbf{f}(\mathbf{q}) = \mathbf{I}_N\mathbf{q} \tag{4.121}$$

and the global behaviour: $\mathbf{y} = \mathbf{q}$.

4.6.3 Global Goal Distribution

Once a feasible global goal trajectory has been established, the goal expression must somehow be transformed from $\mathcal{G}$ to the $\mathcal{G}_i : 1 \leq j \leq N$ generalized spaces of the contributing agents through some inverse of the aggregate mapping as in equation (4.108).

End effector trajectory inverse solution methods, discussed earlier, are commonly implemented as explicit coordination strategies. In these methods joint positions, velocities, accelerations or forces are uniquely correlated to each joint of the manipulator. These methods may be characterized as monolithic centralized inverse solutions often requiring substantial dynamic modelling and are, therefore, a poor foundation for multiagent control. However, a closer examination of these methods reveals subtle differences that ultimately provides insight into an effective decentralized multiagent manipulator control strategy.

Inverse Aggregate Maps

Recall that RMPC,RMRC and RMAC establish a one-to-one mapping of end effector position to joint displacement, velocity, and acceleration respectively. RMPC performs this through a geometric inverse kinematic solution, transforming end effector position from task space to configuration space. Usually nonlinear functions, independent of the manipulator's configuration space history, RMPC inverse solutions tend to be monolithic solutions in which model errors are not easily corrected on line. Therefore, it is imperative that this solution exactly model both manipulator and environment.

Like RMPC, RMRC and RMAC are centralized inverse solutions that correlate unique setpoints with each joint. These integrable methods rely on the Jacobian inverse or pseudoinverse to solve for joint velocities or accelerations respectively. Unlike RMPC, however, the Jacobian inversion process uses the manipulator's configuration space history to correct for modelling errors. As reviewed in chapter 3, RMAC and RMRC use this model to implement an outer, task space control loop, moving centralized end effector tracking from joint space to task space. In redundant systems, this outer loop frees inner loop joint controllers to pursue local goals in the end effector Jacobian null space as described in detail in the next chapter.

Finally, JTC methods apply a similar outer loop control strategy. Provably identical to RMAC given an exact feedforward dynamic model in task space, the Operational Space Formulation (OSF) appears to be little more than a transformation of integrable methods into task space.

This summary suggests that local goals and link controller are correlated, explicit coordination methods. Indeed, RMPC is restrictive even for explicit multiagent coordination, since *all* decision making is bound to a centralized geometric arbitration strategy. Conversely, RMRC and RMAC seem reasonable candidates for explicitly coordinated multiagent manipulator control of redundant systems. Through the use of outer task space control loops, both permit local activity with little global interference.

And yet, the requirement of centralized models for both RMRC and RMAC seems to defeat the purpose of a multiagent system. If the state of each agent must be collected, a centralized aggregate relation assembled and inverted, prior to forming a set of local setpoints, what advantages are there to decomposing a monolithic RMAC controller into a multiagent system? Computationally and architecturally, there is no advantage. Indeed, the overhead of interprocess communications would add to the computational burden.

To reduce or remove the burden of information exchange would require a substantial reduction in reliance upon a centralized model. However, if a model free agent-decentralized global objective could be formulated based on a task space control strategy, the computational disadvantages would vanish, replaced by significant gains in robustness, extensibility, and, possibly, real time response.

Does a model-free decentralized global goal system exist for manipulation robotics? Given the complex nonlinearity of manipulators one would not think so. However, with some minor compromises a decentralized global goal system is possible.

Consider the problem from the standpoint of the lone link agent. What action must this agent take to perform an end effector task if acting alone? Referring to figure 4.22 the required action becomes apparent. Suppose the desired end effector motion can be expressed as a differential distance and rotation:

$$\begin{aligned}\delta\mathbf{r}(t) &= [\delta\mathbf{r}_{\text{trans}}(t)\ \delta\mathbf{r}_{\text{rot}}(t)]^T\\ \delta\mathbf{r}_{\text{trans}}(t) &= [dx\ dy\ dz]^T\\ \delta\mathbf{r}_{\text{rot}}(t) &= [d\theta_x\ d\theta_y\ d\theta_z]^T\end{aligned}$$

and that a lone actuator is to provide this motion. A prismatic actuator, oriented arbitrarily along vector $\mathbf{k}_{i-1}$ must translate:

$$\delta q = \mathbf{k}_{i-1}\cdot\delta\mathbf{r}_{\text{trans}}(t) + 0\cdot\delta\mathbf{r}_{\text{rot}}(t) \tag{4.122}$$

but cannot rotate (hence the second term on the right hand side). For a revolute actuator at $\mathbf{p}_{i-1}$

$$\delta q = \mathbf{k}_{i-1}\times(\mathbf{p}_n - \mathbf{p}_{i-1})\cdot\delta\mathbf{r}_{\text{trans}}(t) + \mathbf{k}_{i-1}\cdot\delta\mathbf{r}(t)_{\text{rot}} \tag{4.123}$$

represents the differential rotation of the ith agent's actuator. Using this line of reasoning the following global differential motion projection operator can be developed:

$$\delta q \quad = \quad g_i(\mathbf{p}_n, \mathbf{p}_{i-1}, \mathbf{k}_{i-1})\cdot\delta\mathbf{r}(t) \tag{4.124}$$

$$g_i(\mathbf{p}_n, \mathbf{p}_{i-1}, \mathbf{k}_{i-1}) = \begin{cases} [\,\mathbf{k}_{i-1} \times (\mathbf{p}_n - \mathbf{p}_{i-1}) \quad \mathbf{k}_{i-1}\,] & \text{if revolute} \\ [\,\mathbf{k}_{i-1} \quad \mathbf{0}\,] & \text{if prismatic} \end{cases} \tag{4.125}$$

Comparison of this equation with equation (2.11) reveals that $g_i(\cdot)$ is, in fact, the column vector of a revolute or prismatic Jacobian respectively. Thus $g_i(\cdot)$ is the row of the Jacobian Transpose, $\mathbf{J}_i^T$. Apparently, the product of the end effector force trajectory and the ith row of the Jacobian transpose prescribes the required generalized force required from the ith actuator. Furthermore, the components of the row elements are simply the local and end effector frames in task space.

Jacobian Transpose Control

Jacobian Transpose control also employs equation (4.120) as the aggregate behaviour and adopts a variation on the inverse solution by determining required forces and torques rather than position, velocity, or acceleration:

$$\tau = \mathbf{J}^T(\mathbf{q})\mathbf{f}_d \tag{4.126}$$

where $\mathbf{f}_d$ is a desired force profile. On the surface, this seems little different than RMPC, RMRC, or RMAC, since the joint torque vector is computed through a central process that models the complete manipulator kinematics in the Jacobian, $\mathbf{J}$. As we have seen, however, the transpose has a unique rowwise structure equivalent to a differential projection of the end effector's motion onto each actuator or:

$$\tau_i = \mathbf{J}_i^T(\mathbf{p}_{i-1}, \mathbf{p}_n) \;\; \mathbf{f} \tag{4.127}$$

Encapsulated within a joint controller, $\mathbf{J}_i^T(\cdot)$ can be applied to a common global goal $\mathbf{f}$, acting as a *proxy* of on behalf of the global goal.

4.6.4 The Global Goal Proxy

Jacobian Transpose Control offers an avenue for *implicitly* coordinated multiagent manipulator control. By assuming that $\mathbf{p}_n$, $\mathbf{p}_{i-1}$ and $\mathbf{f}$ are communicated to or sensed by the agent, the Jacobian Transpose may be decomposed row-wise into n separate, parallel computations. Unlike decentralized controllers in which setpoints and controllers are uniquely correlated, distributed Jacobian Transpose Control enables all the controllers to apply the same global goal, $\mathbf{f}_d$, to $\mathbf{p}_n$. In effect, $\mathbf{J}_i^T(\cdot)$ is a projection operator that identifies that portion of the global goal that the ith joint can perform. Since it is decentralized [7] , the row can be

[7] Strictly speaking *decentralized control* refers to the independent control of generalized coordinates. Clearly task space link states, functions of geometric constraints, are not generalized coordinates. However, from the data flow (or parallel

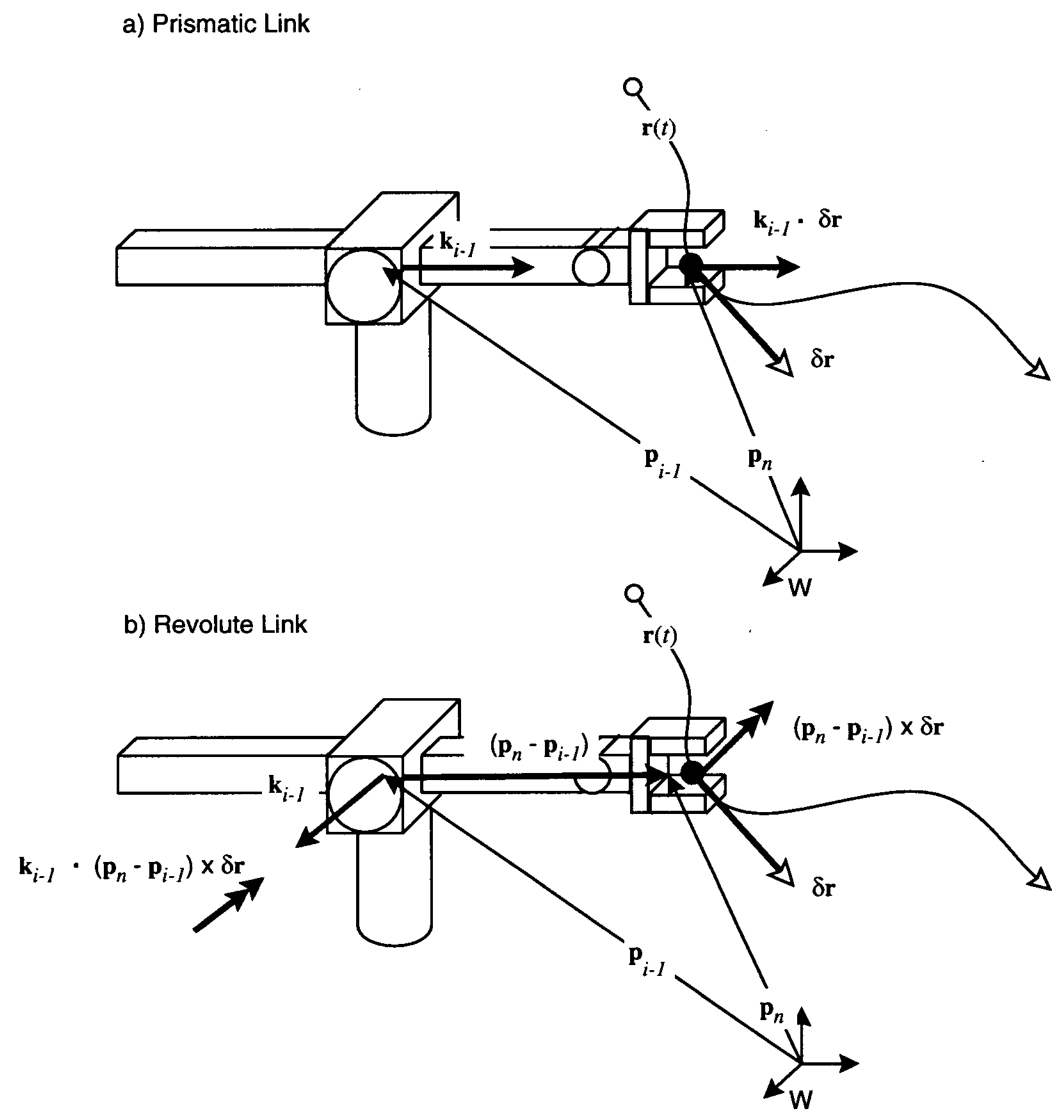


Figure 4.22: The rows of the Jacobian transpose are a projection operator from the global goal (a force vector) to the local goal space (an actuator moment or thrust vector).

incorporated into each link agent as a *proxy* controller, a local controller acting on behalf of a global goal:

$$\mathbf{J}_i^T(\mathbf{p}_{i-1}, \mathbf{p}_{\text{app}}) = \begin{cases} [\,\mathbf{k}_{i-1} \times (\mathbf{p}_{\text{app}} - \mathbf{p}_{i-1}) \quad \mathbf{k}_{i-1}\,] & \text{if revolute} \\ [\,\mathbf{k}_{i-1} \quad \mathbf{0}\,] & \text{if prismatic} \end{cases} \tag{4.128}$$

where $\mathbf{p}_{\text{app}}$ is the *position of application* - in the case of trajectory tracking, $\mathbf{p}_n$. Rather than decentralizing the controller into functions of local *joint state*, this technique decentralizes the controller into functions of local *cartesian state*, $\mathbf{p}_i$ and $\mathbf{k}_i$. To acquire this state information, each link process must either sense or, receive through communication, the necessary information.

If the jth agent link stores a communications directory (arguable a rudimentary model) of agents $j-1$ and $j+1$, directed transmission can be used to pass kinematic *packets* containing cartesian state data between neighbours[8] These messages are equivalent to shared sensory data upon which the agent may act at its own discretion.

To support the proxy controller, *message* packets (composed from the *language* of real numbers) describing the distal coordinate frame of link agent A_{j-1}:

$$\text{Kinematic Packet, } \Xi_{j-1} : \{\mathbf{p}_j, \mathbf{R}_j, \ldots\} \tag{4.129}$$

must be assembled and directly transmitted by A_{j-1} to agent A_j. Interpreted by agent A_j as the jth proximal coordinate frame, A_j must transform the data to describe the distal frame of j and transmit a new packet to A_{j+1}. Through this recursive communication, the communication bus distributes the forward solution (the aggregate relation) over the manipulator.

With the proxy controller and the distributed forward solution, an implicitly coordinated multiagent system is possible. The global goal process derives a and broadcasts a message composed of a *point of application* and a desired applied force, together forming a global goal *couplet*

$$\mathcal{G} \equiv \{\mathbf{p}_{\text{app}}, \mathbf{f}_d\} \tag{4.130}$$

for end effector tracking, $\mathbf{p}_{\text{app}} \in \mathbf{R}^6$ is end effector position vector and $\mathbf{f}_d \in \mathbf{R}^6$ the force goal, both in global coordinates. Once assembled, $\mathcal{G}$ may be *broadcast* to the agent team. The global goal generator performs no arbitration and, as detailed later in this study, need not maintain a model of the manipulation system.

computing) standpoint, global proxies are solely dependent on the task space objective and link frame positions and, therefore, are decentralized inverse aggregate relations. In contrast, traditional decentralized manipulator controllers (e.g. [Stokić, 1984]) require explicit centralized inverse kinematic solutions.

[8]Alternatively a unilateral communications bus structure could be implemented to transport packets from manipulator base to end effector, 'hard wiring' a communications model into the agent.

Trajectory tracking under Jacobian transpose control is not without difficulties, however. It is trivial to construct conditions under which the transpose becomes 'singular', the row product collapsing to zero:

$$\tau_i = \begin{cases} \left[\mathbf{k}_{i-1} \times (\mathbf{p}_{\text{app}} - \mathbf{p}_{i-1}) \quad \mathbf{k}_{i-1}\right]^T \mathbf{f}_d & \text{if revolute} \\ \left[\mathbf{k}_{i-1} \quad \mathbf{0}\right]^T \mathbf{f}_d & \text{if prismatic} \end{cases} \tag{4.131}$$

Though rare, such conditions arise in revolute links through the colinearity of

1. $\mathbf{f}_d$ and $\mathbf{k}_{i-1}$.

2. $\mathbf{f}_d$ and $(\mathbf{p}_{\text{app}} - \mathbf{p}_{i-1})$.

3. $(\mathbf{p}_{\text{app}} - \mathbf{p}_{i-1})$ and $\mathbf{k}_{i-1}$.

and in prismatic links if $\mathbf{f}_d$ and $\mathbf{k}_{i-1}$ are perpendicular. Rarely completely freezing link motion, these conditions can produce a *stalled* or wallowing response as $\mathbf{p}_{\text{app}}$ leaves the collapsed region. With these caveats in mind, Jacobian transpose techniques permit the specification of a variety of global goals, automatically fulfilling the implicit coordination definition.

4.7 A Link Agent Specification

Having identified a distributed trajectory tracking global goal, the specification of an implicitly coordinated link agent can now be formed based on the previously defined structure of generic agent and multiagent systems.

The jth agent acts on a subplant, $\mathbf{G}_j$, first defined in chapter 2.

$$\dot{\mathbf{x}}_j(t) = \mathbf{A}_j\mathbf{x}_j(t) + \mathbf{b}\mathbf{u}_j(t) + \mathbf{d}\mathbf{v}(t) \tag{4.132}$$

where:

$$\mathbf{x}_j = \begin{bmatrix} q_{m_j} \\ \dot{q}_{m_j} \end{bmatrix} \quad \mathbf{A}_j = \begin{bmatrix} 0 & I \\ 0 & -J_{\text{eff}_j}^{-1}(B_{\text{eff}_j} + KK_b) \end{bmatrix} \tag{4.133}$$

$$\mathbf{b}_j = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{u}_j = J_{\text{eff}_j}^{-1} K \mathbf{u}_{cj} \quad \mathbf{d}_j = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{v}_j = -J_{\text{eff}_j}^{-1} d_j \tag{4.134}$$

note that since r_j, the motor transmission coefficient is set to 1.0, $q_j = q_{m_j}$.

The behaviour of this subplant is simply the current link's joint:

$$\mathbf{y}_j(t) = \begin{bmatrix} q_j(t) \\ \dot{q}_j(t) \end{bmatrix} \tag{4.135}$$

For end effector trajectory tracking, the behaviour of the multiagent system is simply the end effector position$\mathbf{p}_n(t)$, velocity $\dot{\mathbf{p}}_n(t)$, and force,$\mathbf{f}_n(t)$:

$$\mathbf{y}(t) = \begin{bmatrix} \mathbf{p}_n(t) \\ \dot{\mathbf{p}}_n(t) \\ \mathbf{f}_n(t) \end{bmatrix} \tag{4.136}$$

The instantaneous local behaviour space $\mathcal{B}_j$, is bounded by the joint position and velocity limits.

$$\mathcal{B}_j = \begin{cases} q_{jlow} \leq q \leq q_{jhigh} & \text{Joint position} \\ \dot{q}_{jmin} \leq \dot{q} \leq \dot{q}_{jmax} & \text{Joint velocity} \end{cases} \tag{4.137}$$

Despite the vector notation, the control effort in this case applied to the jth link is a *scalar* generalized force or torque, $\mathbf{u}_{cj}$. The combination and/or selection of behaviour controllers will be performed through the linear arbitration model defined earlier

$$\mathbf{u}_{cj}(t) = \mathsf{A}(\mathbf{k}(t), \mathbf{u}_{ij}(t) : 1 \leq i \leq b) = \mathbf{k}^T(t)\ \mathbf{u}_j(t) \tag{4.138}$$

Though the details of global and local goal design are yet to be discussed, the form of the global goal is known. The global goal *proxy* discussed earlier provides the foundation for decentralized global goal seeking. A global goal trajectory tracking goal is a force trajectory applied to the end effector. The global goal $\mathbf{r}(t)$:

$$\mathbf{r}(t) = \begin{bmatrix} \mathbf{f}_d(\mathbf{p}_n(t), \dot{\mathbf{p}}_n) \\ \mathbf{p}_n(t) \end{bmatrix} \tag{4.139}$$

where $\mathbf{p}_n$ is the location of the end effector in world coordinates. Therefore, the multiagent system's behaviour space, $\mathcal{B}$, is ultimately limited to the bounds of end effector position and velocity performance as well as the maximum applied force. The latter is a product of the saturation properties of each link motor, $\mathbf{u}_{\text{sat}}$, and the manipulator dynamics:

$$\mathcal{B} = \begin{cases} \mathbf{p}_n \subset \text{Work Volume}_n & \text{Cartesian position} \\ \dot{\mathbf{p}}_n \in R(\mathbf{J}_n) & \text{Cartesian velocity} \\ |\mathbf{f}_n| < |\mathbf{J}^{-T}(D\ddot{\mathbf{q}} + C\dot{\mathbf{q}} + g - \mathbf{u}_{\text{sat}})| & \end{cases} \tag{4.140}$$

The bounds on $\mathbf{f}_d$ simply indicate that the bounds on the available applied force are dependent on the manipulator's instantaneous joint state and joint actuator saturation limits.

The global proxy becomes:

$$\begin{aligned} \mathbf{u}_i &= \mathbf{H}_i(\mathbf{r}_c(t), \mathbf{p}_{j-1}) & (4.141) \\ &= \mathbf{J}_j^T(\mathbf{p}_{j-1}(t), \mathbf{p}_n(t))\mathbf{f}_d(t) & (4.142) \end{aligned}$$

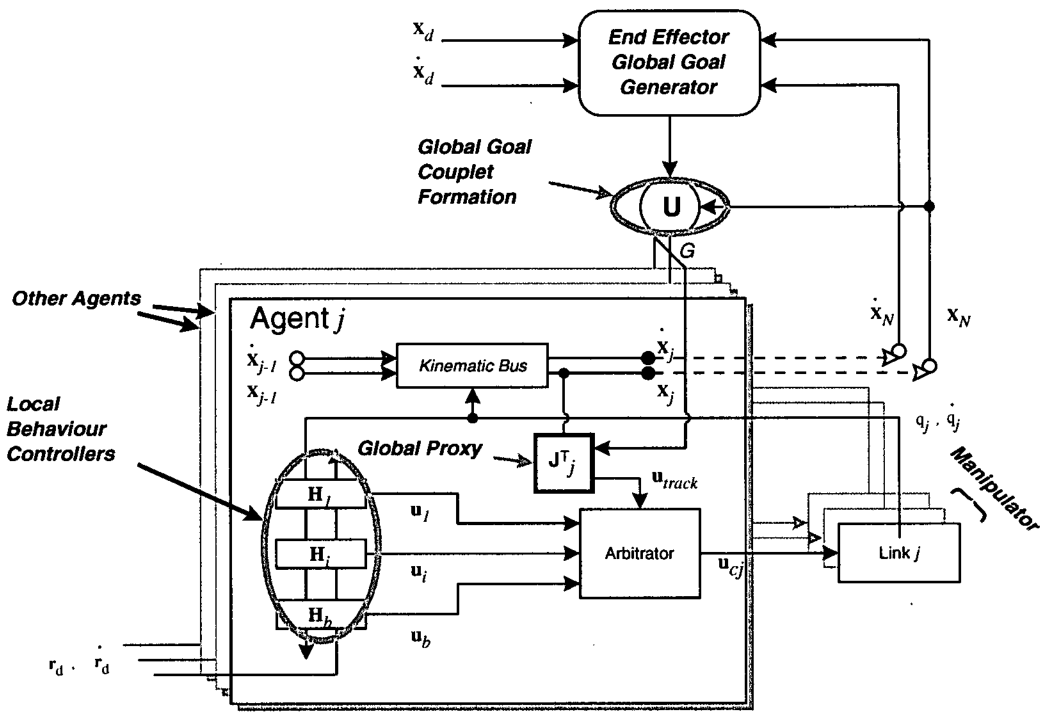


Figure 4.23: With a global goal $\mathbf{f}_d$ and the global goal proxy, $\mathbf{J}_i^T(\mathbf{p}_{i-1}, \mathbf{p}_N)$, manipulator end effector control can be distributed amongst N *agents*, each with local and global goals $\mathbf{r}_d$ and $\mathbf{x}_d$ respectively.

where $\mathbf{J}_j^T(\mathbf{p}_{j-1}(t), \mathbf{p}_n(t))$ is defined in equation (4.128). This prototype link agent structure is depicted in figure 4.23.

4.8 Summary

In this chapter, a number of common terms in both agent and multiagent control have been discussed and formal definitions proposed. In particular, *behaviours* have been defined as the response of some plant; a *behaviour controller* seeks to produce a particular behaviour in response to a specific environmental condition; and an *agent* combines behaviour controllers through some arbitration mechanism. Definitions of public and private messaging clarified three forms of agent control: explicit, implicit, and emergent.

These definitions have enabled the clear statement of three hypotheses that drive interest in agent and multiagent control systems:

An *Agent Control Hypothesis* posits that an agent can be devised to achieve a desired composite behaviour, the product of selection and/or combination of control action to achieve a desired net behaviour.

The *Global behaviour* of a multiagent team was defined as the product of some *aggregate relation*, a function of some set of constraints imposed on the system. Just as agent control was classified according to explicit, implicit and emergent *arbitration* strategies, multiagent control was categorized according to explicit, implicit and emergent *coordination.* From these definitions, multiagent hypotheses were formulated.

A *Multiagent Control Hypothesis* proposed that a *set* of agent controllers may be designed such that the global behaviour of the system achieves a desired global behaviour. By using a global goal generation process, explicit and implicit coordination methods were identified as a mechanism for the production of desired global behaviour of the team. Emergent multiagent coordination was discussed as the product of a multiagent system *without* such central processes, the *Emergent Multiagent Control Hypothesis* proposing that such control was possible.

With these tools in hand, the manipulator control problem was reexamined to discover a mechanism that decomposes a monolithic manipulation process into a set of link agents. After discussing the significance of aggregate and inverse aggregate relations and manipulation, a global goal distribution operator was derived and shown to be exactly equivalent to the Jacobian Transpose.

Given these results, it seems that multiagent manipulator control is indeed possible, though many questions remain unanswered. While multiple control strategies have been combined within Khatib's OSF, Seraji's Configuration Control, and numerous RMAC based redundant resolution controllers, these combinations have been strictly controlled through a centralized arbitration mechanism such as task prioritization and/or null space selection. Implicitly coordinated agents do not have the luxury of a centralized null space selection technique. Naturally, this raises questions on the stability and practicality of multiagent manipulator control.

Decentralized manipulator control is not new. However, decentralization of manipulator control through the Jacobian Transpose has, to this authors knowledge, neither been identified nor demonstrated previously. The next step is to demonstrate that multiagent trajectory tracking is possible and to explore the effects of additional behaviours within each agent on global behaviour and stability. Manipulator simulation, like manipulation in general, is typically performed through monolithic simulators. The next chapter will discuss the structure of a multiprocess manipulator simulation system assembled specifically for this research.

Chapter 5

The Multiprocess Manipulator Simulator

5.1 Introduction

In the previous chapter the structure of agent and multiagent systems was examined from a control theoretic standpoint. It was shown that multiagent systems are composed of multiple processes each capable of achieving independent local goals but cooperating towards a collective, global objective. Exploring these issues further, a manipulator was chosen as a benchmark dynamic system upon which these concepts could be applied. With the identification of an implicit coordination architecture, the dynamics of a multiagent system can now be explored through simulation.

Traditional control simulations are usually monolithic, centralized processes. Though such simulations are often implemented in a high level language such as Mathematica^{TM} or Matlab^{TM}, the real world controller is implemented in a lower level language such as C or assembler. Of course high level languages speed the investigation of controller-plant dynamics by ignoring details such as hardware architecture, data flow and communications limitations. However, these details are crucial to the successful control of a real world plant. For this reason, the software system was designed with a view to the simulation of a real world *multicontroller* environment in the belief that some of these architectural concerns might be clarified. The objective of this chapter is to explain the structure of the Multiprocess Manipulator Simulator (MMS) and its relevance to multiagent control.

Though any simulator remains only an approximation of real world conditions, this multiprocess simulator represented a significantly more realistic and demanding software development environment than monolithic simulators, bearing a striking resemblance to distributed, embedded controller development platforms [RTI, 1996].

5.1.1 Requirements Overview

Functional Requirements

The functional requirements of any manipulator control simulation fall into two areas: environmental simulation and controller logic. The environmental simulation must manage real world events such as temporal flow, manipulator dynamics, and other environmental features such as obstacle kinematics. The control portion of the simulator represents some form of controller that, ultimately, applies a force through each actuator in the simulator.

Fundamentally, a manipulator simulator must faithfully generate the dynamic response of a robot given the application of an *arbitrary* number of linear or revolute actuator forces. In short, the simulator must be capable of modelling any serial configuration of links that drive an arbitrary constant payload along any feasible end effector trajectory. Of course manipulation is not limited to end effector trajectory tracking. A versatile simulator must allow for controllers based on environmental sensing such as force, range, and machine vision sensors, all operating within disparate time scales. Furthermore, the system must accommodate the possibility of world modelling and path planning extensions. In reality a system of this complexity would likely be distributed over multiple computers, becoming a *multiprocess* controller.

To simulate parallel multicontrollers, each controller and dynamic model must become a distinct process, ideally each executing on separate CPUs. This multiprocess architecture places additional requirements on the simulation system in the design of both process and communication structures. In reality, control processes communicate with the environment through sensors and actuators. In monolithic simulators this 'data flow' is scarcely visible, embedded within a common set of symbols shared between simulator and controller. In multiprocess controllers, however, interprocess data flows must be explicitly implemented through interprocess communication (IPC) or shared memory structures.

An inescapable reality of all control processes, and one that presents a significant barrier to SMPA architectures, is that sensing and actuation usually occur at different rates, often by orders of magnitude. Similarly, data transfer between domains (e.g. from controller to motor or from controller to controller) may not occur synchronously or at identical rates, and may not always be reliable. Simulated data flow implementations should reflect these realities.

5.1.2 Design Specification Overview

In examining a generic parallel multiprocess control problem, a number of key processes can be identified as fundamental to a simulated manipulator control system:

- a process manager and synthetic clock
- a generic manipulator and obstacle modelling process
- a generic link process.
- a global goal process

Clearly, the launch and synchronization of the simulator requires a central simulation management module or process manager within which a clock can simulated. For speed, the manipulator model that mimicks the response of a real manipulator and its environment must be contained within a single process. Obviously each link agent is, ideally, an independent process. Global goals are, themselves, independent processes performing tasks such as trajectory generation, visual servoing, and obstacle avoidance.

Despite the multiprocess architecture of the system, each process type shares a number of common elements, such as interprocess communications, vector and matrix algebra, and cartesian state representation to name a few. Coherent design and implementation of a class library is crucial to the simplification of the simulator's design and implementation.

With basic software design specifications in place, a number of nonfunctional requirements impose constraints on the implementation of the multiprocess manipulator simulator.

Nonfunctional Requirements

To take advantage of available computing platforms (Sun SPARCstations, NextStations, and SGI Indigo workstations), the simulator should be designed for extensibility, robustness and portability. Extensibility is greatly enhanced through the adoption of object oriented computing languages such as C++ or objective C and the adherence to both code libraries or class hierarchies. Robustness should be guaranteed through modular testing and industrial debugging tools. Portability can be guaranteed through the use of ANSI standard languages, particularly ANSI 2.0 C++. The selection of an object oriented language such as C++ is based on portability and the ability to build on tested, proven code with little difficulty.

To ease portability between imaging platforms (e.g. Display PostScript or X windows) and between simulation and hardware implementation, animation, rendering, and analysis should be separated from simulation. By using an ASCII data log format any third party data analysis tool may be used for performance assessment.

Another portability issue is the form of IPC or interprocess communication. On the grounds of performance and simplicity, shared memory appears to be a logical choice for interprocess data transfer. Unfortunately, standard shared memory is based on local machine memory protocols for which there is no standardized network support, the most likely hardware option. This forces the selection of widely supported, though slower, UNIX datagram socket techniques.

With these basic requirements for a multiprocess manipulator simulator, this chapter will overview the design of the simulation system and discuss the implications of this architecture on both simulator and controller design.

5.2 Foundations

Written in C++, any manipulation simulator will be composed of numerous classes that, if well designed, fall into a class hierarchy. The Multiprocess Manipulator simulator is no exception, composed of approximately 20 object classes. This section will *briefly* overview the technical details around the most significant members of the MMS class hierarchy. In particular, attention will be focussed on the most complex abstractions including: quaternions, cartesian state and trajectory, LinkModel and ManipulatorModel classes. Where possible every class possessed a dedicated testing module and passed through the industrial-strength bug filter, Purify$^{\mathrm{TM}}$.

5.2.1 Quaternions

Crucial to the implementation of a cartesian controller is the representation of orientation and, in particular, orientation *error*. Orientation, unlike translation, presents special problems not the least of which is that elements of the familiar 3×3 rotation matrix are not a generalized coordinates. A number of alternatives exist including a number of Euler angle representations [Goldstein, 1981]. In representing orientation error, Paul [Paul, 1981] employed a differential orientation system only useful over small displacements (from Appendix

B equation(B.399)):

$$\mathbf{x}_e = \begin{bmatrix} dx \\ dy \\ dz \\ \delta x \\ \delta y \\ \delta z \end{bmatrix} = \begin{bmatrix} \mathbf{n}_a \cdot (\mathbf{p}_d - \mathbf{p}_a) \\ \mathbf{o}_a \cdot (\mathbf{p}_d - \mathbf{p}_a) \\ \mathbf{a}_a \cdot (\mathbf{p}_d - \mathbf{p}_a) \\ \frac{1}{2}(\mathbf{a}_a \cdot \mathbf{o}_d - \mathbf{a}_d \cdot \mathbf{o}_a) \\ \frac{1}{2}(\mathbf{n}_a \cdot \mathbf{a}_d - \mathbf{n}_d \cdot \mathbf{a}_a) \\ \frac{1}{2}(\mathbf{o}_a \cdot \mathbf{n}_d - \mathbf{o}_d \cdot \mathbf{n}_a) \end{bmatrix}$$

where **n**, **o**, and **a** are the column vectors of the end effector orientation matrix and the subscripts d and a refer to desired and actual respectively.

Alternatively, Yuan [Yuan, 1988] developed a Quaternion expression for orientation error $\mathbf{Q}_e = -\mathbf{Q}_1\mathbf{Q}_2$ where the quaternion Q is a couplet $\eta, \mathbf{q}$:

$$\delta\eta = \eta_1\eta_2 + \vec{\mathbf{q}}_1^T\vec{\mathbf{q}}_2 \quad (5.143)$$

$$\delta\vec{\mathbf{q}} = \eta_1\vec{\mathbf{q}}_2 - \eta_2\vec{\mathbf{q}}_1 - \vec{\mathbf{q}}_1 \times \vec{\mathbf{q}}_2 \quad (5.144)$$

and determines that two coordinate systems coincide if and only if $\delta\mathbf{q} = 0$. Thus cartesian error can be represented as:

$$\mathbf{x}_e = \begin{bmatrix} dx \\ dy \\ dz \\ \delta\mathbf{q}_x \\ \delta\mathbf{q}_y \\ \delta\mathbf{q}_z \end{bmatrix} \quad (5.145)$$

(see Appendix A for more detail on quaternion algebra and Appendix B for details on orientation error). Since this representation is compact and accurate over large orientation errors, MMS regulates cartesian orientation error through quaternion representations.

By providing a quaternion class with the necessary addition, subtraction and inversion operators, a uniform interface to both vectors and quaternions was established. This interface greatly simplifies the implementation of the Cartesian State and Trajectory classes to follow.

5.2.2 Cartesian State

The Cartesian State is universally understood to be a position and velocity vector in $\mathbb{R}^6$, though, as stated above, orientation is usually represented as a 3×3 matrix. In building a Cartesian State, quaternions were

Class	Object	Representation
vector	`P`	position
vector	`V`	velocity
vector	`A`	acceleration
Quaternion	`Q`	orientation
vector	`W`	angular velocity
vector	`Wd`	angular acceleration

Table 5.2: Structure of the `CartesianState`.

incorporated into the object definition summarized in table 5.2. With equality and subtraction operators, arithmetic manipulation of cartesian states, (e.g. cartesian error vectors, trajectory generation, etc.) is straightforward.

5.2.3 Cartesian Trajectories and Paths

Cartesian Trajectories used four simpler trajectory objects to build both step and cubic linear trajectories in both position and orientation. A `CartesianTrajectory` could be completely described through a time interval, starting, and endpoint cartesian states. Once instantiated, `CartesianTrajectory` data members could be updated through an `update(int TimeStamp)` method. The class has been designed to be easily extensible to other trajectory types.

Each element of the cartesian position vector was described by a trajectory object.Since the difference between two quaternion orientations is, itself a quaternion, an orientation trajectory can be described as a time varying *angle* of rotation, ϕ, about the quaternion *error* axis $\delta\mathbf{q}$ was also specified as a trajectory object. Hence a trajectory in $\mathbb{R}^6$ requires the *maintenance* of four time varying trajectory objects $x_{e_x}(t), x_{e_y}(t), x_{e_z}(t), \phi_e(t)$ and the preservation of an orientation error axis, $\delta\mathbf{q}$, and start (or end) point positions.

Since realistic manipulator trajectories are usually chains of smooth trajectories sometimes joined by discrete changes, a `CartesianPath` list class was defined that automatically managed the trajectory over changing segments. New trajectory segments could be appended to the trajectory list at any time.

5.2.4 Parsers

The system relies heavily on a flat file database format that describes the physical and architectural characteristics of each link in the manipulator. Though flat files are often adopted for legibility, large streams of numbers remain extraordinarily difficult to read without classification. A flat file format based on a simple BNF syntax implemented through Lex and Yacc[1] enabled the assignment of numbers, strings, vectors, and matrices to named standard variables such as `mass` and `inertia` or 'user defined parameters' for unique variables (e.g. in a particular controller implementation). This data file system also allowed the definition of more abstract structures such as `globalgoal`, `knotpoint`,`controller` and `link`, the details of which are discussed later.

In general, manipulator control experiments test the response of systems given variations in robot type, payloads, trajectories, control algorithms, and, sometimes, obstacle trajectories. Within the Multiprocess Manipulator Simulator, these four entities have been formed into an abstraction called a *scenario.*

Experiments as Scenarios

A *scenario* is formed by bundling five database files:

- a robot database: manipulator `link` and `controller` specifications
- a payload database: payload mass and inertia.
- an obstacle database: obstacle geometries and `knotpoint` lists.
- a goal trajectory database: end effector `knotpoint` list.
- a global goal database: a `globalgoal` list.

together within a single directory structure. By maintaining libraries of robot types, payloads, trajectories, end effector goal descriptions, and obstacle trajectories, scenarios can be the rapidly assembled for experimental trials. Furthermore, automated scripting enables rapid construction of experimental *series* in which a single database type may be varied over a number of scenarios (e.g. varying payload mass). So significant is the scenario concept that each process in the MMS is launched with an internal representation of this file bundle, a `Scenario`, retrieving initialization data as required and writing execution and data logs to the `Scenario` directory.

5.2.5 LinkModel

The abstraction of a 'link', though straightforward, is fundamental, appearing as a parent class to both the link agent and link rendering concepts. The `LinkModel` class contains fundamental link descriptive types

[1]Lex and YACC (Yet Another Compiler Compiler), both standard Unix utilities

Class	**Object**	**Representation**
char	`Name`	link label
matrix	`I`	inertia
vector	`C`	centroid
vector	`X`	state vector
char	`Type`	revolute or prismatic
double	`Mass`	link mass
double	`theta`	revolute link rotation
double	`offset`	offset displacement
double	`length`	link length
double	`alpha`	link twist

Table 5.3: Structure of the `LinkModel`.

and scenario based self initialization methods. Table 5.3 details the root structure of the `LinkModel` family tree.

5.2.6 Interprocess Communications

Interprocess communications play a central role in the multiprocess manipulator simulator. IPC acts in three capacities: the application of forces computed by link processes to the link joint motors in the dynamic model, the simulation of sensor data flow in link agents, and communication between link and global goal processes.

In Berkeley UNIX, the main mechanism for interprocess communication is *sockets* a named file-like device with a well deserved reputation as both difficult to implement and debug. To simplify and encapsulate the communications protocol a `Socket` object was developed as the base of a communications hierarchy that is ultimately equivalent to a distributed shared memory system.

The first layer of this hierarchy is the abstraction of data sources and sinks or *Measurands* and *Sensors* into segments of shared data `SensorVector` and `MeasurandVectors`.

Sensor and Measurand Vectors

`MeasurandVectors` are the means through which an MMS process *sends* data streams to other processes (thus acting as a *measurand*). Each `MeasurandVector` is composed of a linked list of data types (double, vector, or matrix) and a transmission buffer composed of an array of doubles. Double, vectors, or matrices of specified dimensions allocated through a MeasurandVector with a specific update rate are mirrored in the buffer.Allocated data elements may then be transmitted through an `update(int TimeStamp)` message to the

`MeasurandVector`. With a specific update rate, *stale* buffer elements can be updated prior to transmission.

Similarly, `SensorVectors` are the means through which a process *receives* data streams from other processes (thus acting as a *sensor*). Just as in the `MeasurandVector`, doubles, vectors, or matrices of specific dimensions can be allocated with a specific update rate and are mirrored in a receive buffer. Again sensed data can be updated through an `update(int TimeStamp)` message in which only *stale* variables are updated.

To establish the equivalent of a distributed shared memory, a transmitting processes must establish a `MeasurandVector` and receiving process a `SensorVector`. Once both processes have allocated identical data sets in identical orders, continuous data updates can be facilitated through update messages on each side of the communication connection.

Sensing and Actuation

By default, each link has joint position and velocity sensors. These are implemented through `SensorVector` connections to the dynamic model and are updated at a specified rate. Other sensors can also be implemented: end effector force data for example. Again this is provided through a `SensorVector` connection to the dynamic model. Conversely, joint actuator forces are exerted on the link through a `MeasurandVector` connection to the dynamic model.

The foregoing overview provides enough background to simplify the description of the individual processes in the Multiagent Manipulator Simulator. In the following sections, each MMS process structure will be briefly reviewed and the execution life cycle explained.

5.3 The Process Manager and Synthetic Clock, `SI`

The Process Manager launches, safely initializes, and synchronizes each process in the system. Given a scenario, `SI` launches and initializes the necessary link processes, the dynamic modelling process, and the goal process. The process manager also establishes communication links to each process used to synchronize processes to the synthetic clock. In the MMS, an artificial clock was adopted to maintain synchronization and to ensure that all the time dependent processes receive a common time stamp.

5.3.1 Execution

The process manager and synthetic clock are contained within the Simulation Initialization process, `SI`. Launched with the command:

```
SI <aScenarioName>
```

, `SI` reads the robot database file and *spawns* a link process, `LP`, for every `link` in the manipulator description. A goal process, `GP`, and dynamic model, `DM`, are then spawned successively. Therefore at any time in the simulation cycle at least $N+2$ processes are active an N link manipulator.

Once initialized, `SI` creates a set of synchronization sockets through which `TimeStamps` can be transmitted. The link and goal processes then await for the time stamps from `SI`, the number of milliseconds (a long integer) from simulation T_0 and broadcast in 1 ms. intervals.

Execution commences with a call to the method,`execute(StartTime,EndTime,TimeStep)`. Only `SI` uses the experiments duration from `StartTime` to `EndTime`, all other processes cycle *ad infinitum.* During execution, the process manager transmits time stamps to both goal and link processes, synchronizing them to a common, synthetic clock. Before sending the next timestamp, the process manager awaits an 'echo' of the timestamp from the dynamic modeler.

If an error occurs, due to spawn failures or a late 'end-of-cycle' message, or if the simulation is concluded, the system invokes a clean shutdown by transmitting a `kill` notice – a negative timestamp. During error based termination the system simply completes writing to execution logs, terminates socket connections, to exit cleanly. During normal termination, the system awaits an 'end-of-cycle' message from `DM` before shutting down.

The overall structure of the multiprocess manipulator simulator is illustrated in figure 5.24.

5.4 The Dynamic Modeler, `DM`

The dynamic modeler represents the core of the *simulation* portion of the MMS. The purpose of the Dynamic Modeler is to accurately compute the response of a specific robot given physical characteristics such as the mass, inertia, relative geometry, and state of each link in the manipulator and the link motor's applied forces. The dynamic modeler must also manage the motion of obstacles in the environment.

5.4.1 `ManipulatorModel`

An abstraction of a manipulator model is important for both the dynamic modelling aspect of the MMS and also for controllers that rely on models for dynamic cancellation. Unlike other C++ modules in the class hierarchy that compromise speed for ease of use, the `ManipulatorModel` class has been optimized exclusively for speed. Object oriented methods are powerful programming tools and greatly simplify the design and

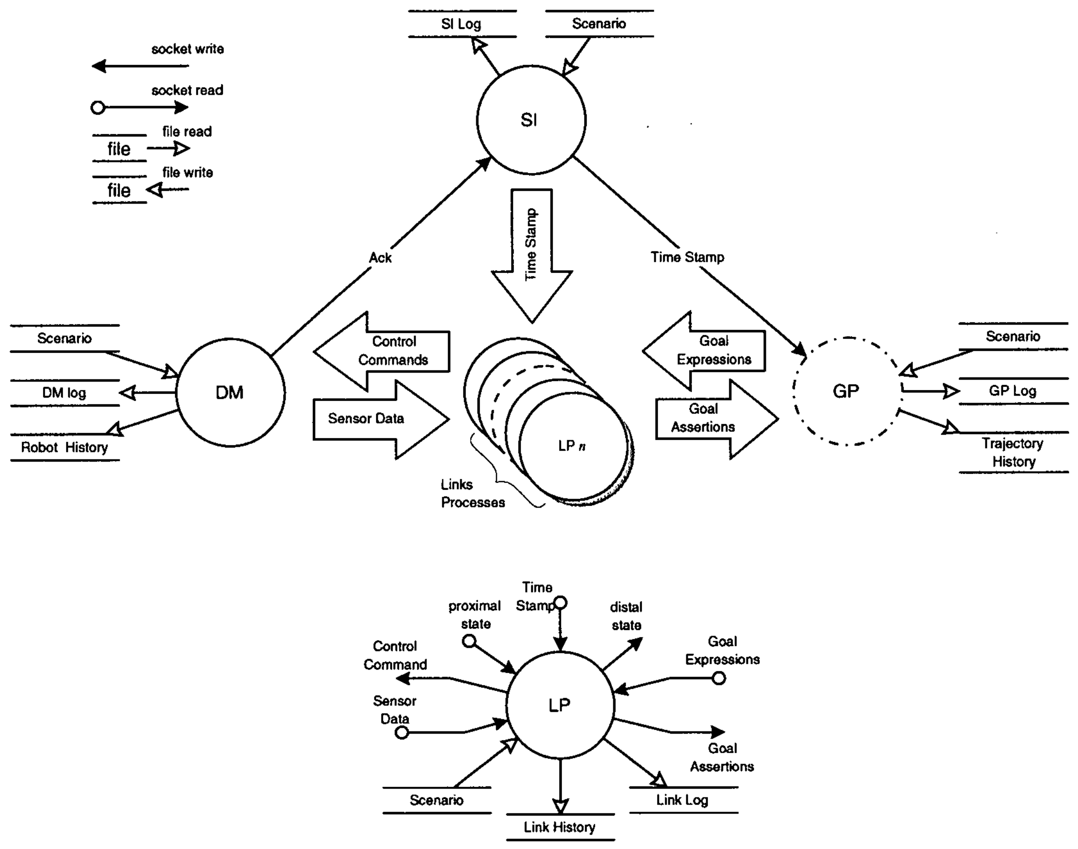


Figure 5.24: The interprocess data flow of the Multiprocess Manipulator Simulator. As GP is not a mandatory process, it appears as a dashed circle.

testing process. However, OOP methods are often slower than traditional C procedural methods. Therefore, a compute intensive dynamic model [Luh, 1980a, Walker, 1982] was implemented based on proven 'C' tools [Press, 1988].

The manipulator was represented by an array of `link` structures, similar to the `LinkModel` object above. This approach permitted fast access to a given link's parameters through a single structure rather than simpler methods that aggregate manipulator parameters into monolithic arrays indexed by link number.

Kinematics and Dynamics

Kinematics and dynamics were computed using established Newton-Euler methods [Luh, 1980b, Walker, 1982]. Given link state values, $[q_i\ \dot{q}_i\ \ddot{q}_i]^T : 1 \leq i \leq N$, and the link `Type`, link cartesian position, velocity, and acceleration, $[\mathbf{x}_i\ \dot{\mathbf{x}}_i\ \ddot{\mathbf{x}}_i]^T : 1 \leq i \leq N$ were developed through a recursive outward computation from the manipulator base to the end effector.

$$\mathbf{R}_i = \mathbf{R}_{i-1}\,\mathbf{R}_i^{i-1}(q_i) \tag{5.146}$$

$$\mathbf{p}_i = \mathbf{R}_i\mathbf{p}_i^{i-1} + \mathbf{p}_{i-1} \tag{5.147}$$

$$\omega_i = \begin{cases} \omega_{i-1} + \mathbf{R}_{i-1}\dot{q}_i\mathbf{k}_i & \text{if revolute} \\ \omega_{i-1} & \text{if prismatic} \end{cases} \tag{5.148}$$

$$\dot{\mathbf{p}}_i = \begin{cases} \dot{\mathbf{p}}_{i-1} + \mathbf{R}_{i-1}\dot{q}_i\mathbf{k} \times (\mathbf{p}_i - \mathbf{p}_{i-1}) & \text{if revolute} \\ \dot{\mathbf{p}}_{i-1} + \mathbf{R}_{i-1}\dot{q}_i\mathbf{k} & \text{if prismatic} \end{cases} \tag{5.149}$$

$$\dot{\omega}_i = \begin{cases} \dot{\omega}_{i-1} + \mathbf{R}_{i-1}\ddot{q}_i\mathbf{k} + \mathbf{R}_{i-1}\dot{q}_i\mathbf{k} \times (\mathbf{p}_i - \mathbf{p}_{i-1}) & \text{if revolute} \\ \dot{\omega}_{i-1} & \text{if prismatic} \end{cases} \tag{5.150}$$

$$\ddot{\mathbf{p}}_i = \begin{cases} \ddot{\mathbf{p}}_{i-1} + \dot{\omega}_i \times \mathbf{p}_i + \omega_i \times \omega_i \times (\mathbf{p}_i - \mathbf{p}_{i-1}) & \text{if revolute} \\ \ddot{\mathbf{p}}_{i-1} + \mathbf{R}_{i-1}\ddot{q}_i\mathbf{k} + \dot{\omega}_i \times (\mathbf{p}_i - \mathbf{p}_{i-1}) + 2\dot{\omega}_i \times \mathbf{R}_{i-1}\dot{q}_i\mathbf{k} + \omega_i \times \omega_i \times \mathbf{p}_i & \text{if prismatic} \end{cases} \tag{5.151}$$

Recall that the ith joint displacement (and derivatives) is expressed as a *vector* with a direction along link a prismatic or revolute axis of frame $i-1$ (hence the frequent terms such as: $\mathbf{R}_{i-1}\dot{q}_i\mathbf{k}$).

Link forces were developed in a recursive inward computation from end effector to manipulator base. The inertial forces in each link:

$$\mathbf{f}_{\text{total}_i} = m_i\ddot{\mathbf{p}}_i \tag{5.152}$$

$$\mathbf{m}_{\text{total}_i} = \mathbf{J}_i\dot{\omega}_i + \omega_i \times (\mathbf{J}_i\omega_i) \tag{5.153}$$

where $\mathbf{J}_i$ is the link inertia in *world* coordinates. Applying Newtons Second Law to the ith link:

$$\mathbf{f}_{\text{total}_i} = \mathbf{f}_i - \mathbf{f}_{i+1} \tag{5.154}$$

$$\mathbf{m}_{\text{total}_i} = \mathbf{m}_i - m_{i+1} + (\mathbf{p}_{i-1} - \mathbf{p}_{ci}) \times \mathbf{f}_i - (\mathbf{p}_i - \mathbf{p}_{ci}) \times \mathbf{f}_{i+1} \tag{5.155}$$

where $\mathbf{f}_{i+1}$ is the force applied to the ith link by link $i+1$ and $\mathbf{p}_{ci}$ is the position of the center of mass in world coordinates. Solving for $\mathbf{f}_i$ and $\mathbf{m}_i$:

$$\mathbf{f}_i = \mathbf{f}_{i+1} + \mathbf{f}_{\text{total}_i} \tag{5.156}$$

$$\mathbf{m}_i = \mathbf{m}_{i+1} + \mathbf{p}_i^* \times \mathbf{f}_{i+1} - (\mathbf{p}_i^* - \mathbf{p}_{ci}^*) \times \mathbf{F}_i \tag{5.157}$$

where $\mathbf{p}_i^* = \mathbf{p}_{i+1} - \mathbf{p}_i$ and $\mathbf{p}_{ci}^*$ is the location of the ith link's center of mass in link i coordinates (i.e. a constant). The forces at the joint may then be extracted from $\mathbf{f}_i$ and $\mathbf{m}_i$ through:

$$\tau_i = \begin{cases} (\mathbf{R}_{i-1}\mathbf{k})^T\mathbf{m} & \text{if rotational} \\ (\mathbf{R}_{i-1}\mathbf{k})^T\mathbf{f} & \text{if prismatic} \end{cases} \tag{5.158}$$

The origin of the world coordinate system is fixed to the base of the manipulator and assigned a null velocity (i.e. $\dot{\mathbf{p}}_0 = \mathbf{0}\ \ \mathbf{w}_0 = \mathbf{0}$). Gravity is simulated by applying an acceleration of magnitude g to the world coordinate system (e.g. $\ddot{\mathbf{p}}_0 = g\mathbf{k}\ \ \dot{\mathbf{w}}_0 = \mathbf{0}$).

Dynamics are initialized with either known end effector forces (e.g. $\mathbf{f}_{n+1} = \mathbf{f}_{ee}\ \ \mathbf{m}_{n+1} = \mathbf{m}_{ee}$) or inertial payload forces (e.g. assuming a fixed end effector payload: $\mathbf{f}_{n+1} = m_{\text{payload}}\ddot{\mathbf{p}}_n\ \ \mathbf{m}_{n+1} = \mathbf{J}_{\text{payload}}\dot{\omega}_n + \omega_n \times (\mathbf{J}_{\text{payload}}\omega_n)$). For clarity, the manipulator equations of motion have been presented in world coordinates. However, the actual dynamic model adopts a faster, though less intuitive form of these equations composed in link coordinates. For a complete discussion of this well established technique consult [Luh, 1980a].

Model Integration

The objective of a dynamic simulator is to compute the accelerations of the joints given the applied torques, displacements, and velocities, integrating the results to develop the next time step's position and velocity. At each timestep (every 0.001 seconds of simulation time) the MMS manipulator model was integrated through a fixed timestep fourth order Runge Kutta integrator [Press, 1988]. Joint accelerations are determined from the robot equation:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}^T(\mathbf{f}_e) = \tau \tag{5.159}$$

The robot equation can be rewritten:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} = \tau - \mathbf{b} \tag{5.160}$$

$$\mathbf{b} = \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}^T(\mathbf{f}_e) \tag{5.161}$$

where $\mathbf{b}$ is a *bias vector*, the output of the recursive Newton Euler dynamics algorithm when $\ddot{\mathbf{q}}$ have been set to zero. The accelerations may then be computed simply through:

$$\ddot{\mathbf{q}} = \mathbf{D}(\mathbf{q})^{-1}[\tau - \mathbf{b}] \tag{5.162}$$

The final problem is the determination of the mass matrix $\mathbf{D}(\mathbf{q})$. For a manipulator of known geometry an analytical solution may be determined manually a priori. However, for arbitrary manipulators a numerical method is required.

Determining the Manipulator Mass Matrix

The mass matrix algorithm (from [Walker, 1982]) employs an interesting technique to determine the manipulator mass matrix, the most CPU intensive task of a generic manipulator modelling system. Basically the technique uses the relation that each element, ij, of the mass matrix is equivalent to the internal forces present in link i during a unit acceleration of link j, all other links being motionless. For a given accelerated link j, the inferior links $> j$ are treated as a rigid body, requiring the computation of an effective mass center and moment of inertia for this rigid body. The forces exerted on the rigid body above link j are:

$$\mathbf{F}_j = M_j \mathbf{a}_j \tag{5.163}$$

where M_j and $\mathbf{a}_j$ are the mass and acceleration of the composite rigid body in the jth coordinate system. Since only the jth link is accelerating and all others are motionless, this may be rewritten as:

$$\mathbf{F}_j = M_j \ddot{q}_j \mathbf{z}_{j-1} \times \mathbf{c}_j \tag{5.164}$$

where $\mathbf{c}_j$ is the rigid body's center of mass. Rewritten for a unit acceleration:

$$\left.\begin{array}{l} \mathbf{F}_j = \mathbf{z}_{j-1} \times M_j \mathbf{c}_j \\ \mathbf{F}_j = M_j \mathbf{z}_{j-1} \end{array}\right\} \begin{array}{l} \text{revolute joint } j \\ \text{prismatic joint } j \end{array} \tag{5.165}$$

Similarly the applied moment becomes:

$$\left.\begin{array}{l} \mathbf{M}_j = \mathbf{z}_{j-1} \mathbf{E}_j \\ \mathbf{M}_j = 0 \end{array}\right\} \begin{array}{l} \text{revolute joint } j \\ \text{prismatic joint } j \end{array} \tag{5.166}$$

where $\mathbf{E}_j$ is the effective moment of inertia of the composite rigid body. These forces,$\mathbf{f}_i$, and moments,$\mathbf{m}_i$, are propagated down to the base of the manipulator employing the recursive relations begun at joint j:

$$\mathbf{f}_j = \mathbf{F}_j \tag{5.167}$$

$$\mathbf{m}_j = \mathbf{M}_j + \mathbf{c}_j \times \mathbf{F}_j \tag{5.168}$$

and propagated down to the base of the manipulator through all the links $i < j$:

$$\mathbf{f}_i = \mathbf{f}_{i+1} \tag{5.169}$$

$$\mathbf{m}_i = \mathbf{m}_{i+1} + \mathbf{p}_i \times \mathbf{f}_{i+1} \tag{5.170}$$

where $\mathbf{p}_i$ is the distance from joint $i-1$ to i and remembering that all the links $i < j$ are motionless. With the values of $\mathbf{f}_i$ and $\mathbf{m}_i$ known for each link $i = 1 \ldots j$, the value of the Mass Matrix Elements for the jth column may be determined by:

$$\mathbf{D}_{ij} = \left\{ \begin{array}{ll} \mathbf{z}_{i-1} \cdot \mathbf{m}_i & \text{revolute joint } i \\ \mathbf{z}_{i-1} \cdot \mathbf{f}_i & \text{prismatic joint } i \end{array} \right. \tag{5.171}$$

The key in this process is designing this algorithm to be completely recursive, thus conserving CPU resources. Walker and Orin [Walker, 1982] describes the necessary recursive technique in detail, finally recommending a "method 3", used in DM, that has the least computations per degree of freedom. The total minimum computations to compute N joint accelerations using "method 3" becomes [Walker, 1982]:

$$\begin{aligned} \text{multiplications :} \quad & (\frac{1}{6})N^3 + (13\frac{1}{2})N^2 + (192\frac{1}{3})N - 49 \\ \text{additions :} \quad & (\frac{1}{6})N^3 + 8N^2 + (166\frac{5}{6})N - 64 \end{aligned}$$

Thus, at best, arbitrary manipulator dynamic model computations are $O(N^3)$.

5.4.2 Obstacle Modelling

In this simulation, obstacles were represented as rigid physical objects having both geometry and movement specified in the obstacle database. An `ObstacleManager` acted as a single interface between the dynamic modelling process and a set of obstacle objects. Updates or queries posted to the manager were broadcast to all obstacles.

`Obstacle` objects were specified in database entries standardized about the obstacle type, current position and path of the obstacle frame to form a trajectory script. Specific characteristics, unique to obstacle subclasses, were specified through nonstandard 'user defined variables'. Though only spheres were implemented (`SphereObstacles`), the obstacle interface was designed to be generic and could support arbitrary obstacle geometry.

In addition to motion, obstacles must also respond to sensor probes of the environment. Fortunately, such probing produces data of limited dimension, either scalar, linear arrays, or matrices of values. For a given sensor type, the magnitude of these values depends entirely on the relevant obstacle property (e.g. shape, temperature, etc). Thus sensor implementations reside not in a `Sensor` class but within the obstacle itself. In this way a `Ping` method applied to a particular obstacle returns the vector to the surface nearest to the sensor source – much like an ultrasound, lidar, or radar sensor. It is up to the obstacle implementation to develop the correct return vector for a particular obstacle morphology.

5.4.3 Physical Realizability

Though every attempt has been made to ensure that the manipulator and motor dynamic models are realistic, one important real world characteristic has been omitted from DM, interference checking. Interferences occur when two objects simultaneously occupy identical regions of space. Of course, in reality colliding objects do not intersect, but rebound according to the kinematic and material properties of both bodies. In manipulation simulation, interference is frequent as revolute manipulators rotate *ad infinitum* about their axes and links intersect both work space obstacles and other links. The solution is to model and maintain the boundaries of each object in the system and, if interference is detected, apply collision forces to interfering bodies. So for every N simulated bodies $N-1$ interference checks must be performed or $N(N-1)$ checks per timestep and, if necessary, collision forces applied. Collision modelling, too, is nontrivial, requiring accurate surface geometric and material descriptions to compute incident angles and return forces.

Neither the link nor obstacle modelling systems adopt interference checking (a common practice in many manipulator simulations) nor is a collision model provided. The computational cost of such checking, the subsequent difficulty of developing a realistic collision model, and the limited value of such extensions to an investigation of multiagent control are sufficient cause to leave these features to future manipulator simulation systems.

So while manipulator dynamics have been developed according to standard well established simulation techniques, DM is unable to accurately portray manipulators collision dynamics during interference events. Though theoretically valuable, the manipulator's response during such events is, nevertheless, physically unrealizable.

5.4.4 Execution

When DM is spawned by SI:

```
    DM <aScenarioName>
```

the robot database is used to build the ManipulatorModel object discussed above and to construct two socket connections to each link process, a measurand connection to provide each link with joint state data and a dynamic stream connection to receive motor control commands. DM also inspects the global goal database for global goal sensor requirements (e.g. force sensing in RMFC). Obstacles and their trajectories are created based on the contents of the obstacle database. DM then creates both an execution log and a result history within the scenario directory.

Once initialized DM initializes sensor data to the Link Processes. The process then enters the main event loop by awaiting a controller command, a timestamp and force value pair. If either a kill notice is received or a discrepancy in the timestamps is observed, an error condition has occurred and DM begins shutdown and cleanup. Otherwise, the model proceeds with the integration process, a call to the RungeKutta4(.) method.

The result of a successful integration, a new manipulator state, is immediately transmitted to the link processes as position sensor data. Similarly, force data computed through the Newton Euler dynamic model methods may also be transmitted, if required. Integration fails only if the inversion of the mass matrix, $\mathbf{D}(\mathbf{q})$, in equation 5.162 (via LU decomposition) fails, a sure sign of lurking algorithmic errors or controller instability. Such failures trigger `kill` notice transmissions and subsequent process shutdowns.

If the state transmissions are successful, an end of cycle message is passed to `SI` and the system returns to the start of the main event loop.

The dynamic modeler executes the main event loop *ad infinitum.* However, any `kill` notice frees `DM` from this loop and invokes a clean shutdown of the process, including the transmission of the final timestamp to `SI`, the storage to file of historical data, the deallocation of memory, and the closure socket connections.

Thus far the simulator portion of MMS has been described, the basic infrastructure of the system. The following sections detail the control portion and represent a significant divergence from traditional monolithic control architectures.

5.5 The Global Goal Process, `GP`

The global goal process serves a key function in MMS: to compute and distribute goals to link processes. The design of the global goal process accommodates multiple global goal processes through the adoption of a `GoalArray`, a simple array of goal generation objects. As indicated in the definition D4.1, a goal is a trajectory through some goal space. The goal space varies between control strategies. Traditional RMPC systems translate desired end effector locations into local goals for each link process, position setpoints in joint space. RMAC systems, however, translate these same locations into force setpoints for each link process. JTC, as discussed earlier, uses a goal couplet containing both end effector position and desired force (equation(4.130)),

5.5.1 The `GlobalAgent` Object

The `GlobalAgent` forms the core of the global process. This object has two important data structures: the `Goal` array, and the `AgentIOStream` array. The `Goal` array contains a list of active global goals, and the `AgentIOStream` is an array composed of a `SensorVectors` and a `MeasurandVectors`, each of which comprises an input and output data flow between global and link processes.

The purpose of the `GlobalAgent` is to generate and distribute global goals to all of the link processes.

There are two sources of goal generation. The first is through local goal generators such as :

- Resolved Motion Force Control
- Resolved Motion Decentralized Adaptive Control
- Resolved Motion Acceleration Control.

and reviewed in the next chapter. The second mechanism is through remote goal generation in which link processes *assert* global goals to the agent team, demonstrated in Resolved Motion Obstacle Avoidance in the next chapter. In this case, the global goal process acts as a rebroadcast medium propagating a single goal expression to the entire agent group.

To support *multiple* goals, the `GlobalAgent` employs a `Goal` array. Given goals specified within the Global Goal database file, the `GlobalAgent` instantiates objects inheriting from the `Controller` class into the `Goal` array.

The `Controller` Abstract Superclass

The `Controller` class encapsulates any object possessing time dependent response within an ***abstract superclass***, such as goals and controllers. By adopting an abstract superclass for various goals, goal subclasses can be placed into a single array of `Controllers`. Practically speaking, this means that regardless of the controller subclass, all `Goal` array members will respond to the `update()` method regardless of their subclass. For a further description of the abstract superclass concept consult [Lippman, 1991].

`IOStreams`

To communicate global goals, receive goal assertions, and kinematic bus data, a general purpose interprocess input/output data structure or `IOStream` implements a bilateral communication connection. The `GlobalAgent` object can pass these connections to members of the `Goal` array, allowing each to establish input or output data streams with the agent team at will. In this way an end effector trajectory tracking task can simultaneously receive end effector state data and obstacle avoidance goal assertions without interference from the `GlobalAgent` object. The only requirements for successful mating of goal and agent communication is that the agent proxies and the global goals employ identical data formats and that space is allocated in `SensorVector` and `MeasurandVector` structures for both link as the global goal processes. Fortunately, the latter is easily ensured by adopting identical sequences in both link `controller` and `globalgoal` lists in the robot and global goal database files respectively.

5.5.2 Execution

At launch GP parses the robot in the Scenario and creates the necessary number of IOStreams. Based on their order of appearance in the global goal database, GP instantiates the necessary goal generators into the Goal array, passing the IOStreams to each generator. Within each goal generator, data is allocated from the SensorVector and a MeasurandVector components of the IOStreams. This allocation step, when mirrored in the link processes, establishes the equivalent of a shared memory connection between the two processes. Though not all goal generators allocate SensorVector memory, all allocate MeasurandVector space through which the global goal is transmitted.

Once initialized GP enters an event loop by awaiting the arrival of a timestamp from SI. On receipt of a timestamp, the input IOStream is updated. If successful, the GlobalAgent successively calls the update() method for each Controller subclass in the Goal array. This method either updates or copies goal assertions into the appropriate allocated variables in the output IOstream. The event loop is concluded by updateing the output IOstream.

Just as the receipt of a kill notice invokes shutdown and cleanup, so too will an IOStream transmission or receipt failure. Like the other processes, termination does not occur until the execution and historical data logs are completed.

5.6 The Link Process,LP

The link process draws on the scenario database to construct an agent that receives sensor data streams from the dynamic model, acts on an arbitrary number of both local and global goals, and returns a some force output back to the dynamic model. The core of LP is the Agent object.

5.6.1 The Agent Object

The Agent Object implements the concept of an agent as defined in chapter 4.8. Since the purpose of an agent is to arbitrate between behaviours employing some decision or combination strategy, a means of storing and invoking generic behaviours must be provided as well as a mechanism for the collection and supply of goal and sensor data to and from external processes.

Structurally, an agent object is composed of three fundamental entities:

- the Kinematic Bus
- the IOStream

- a set of Controllers
- an arbitrator

exploiting six fundamental data flows:

- kinematic bus streams (input/output)
- Sensor data streams (input)
- Global goal `IOStreams` (input/output)
- Control effort (output)

5.6.2 The Kinematic Bus

LP decentralizes control by distributing *cartesian state* computations over each link of the manipulator. This is accomplished by supplying each link process with locally sensed displacement data, a local geometric model, and through distributed Newton Euler equations coupled to an interagent communication infrastructure or *Kinematic Bus.* This infrastructure has a flow through recursive architecture in which the ith link successively receives and transforms cartesian state information *packets* from link $i-1$ and transmits the transformed data in a packet to link $i+1$. The *Kinematic Packet* sent from i to $i+1$ is a *message* with syntax:

$$\text{Kinematic Packet, } \Xi_i : \{\mathbf{p}_i, \mathbf{R}_i, \dot{\mathbf{p}}_i, \omega_i\} \tag{5.172}$$

Since values are in double precision (8 bytes) the kinematic packet is 144 bytes:

- $\mathbf{R}_i$, an orientation matrix (72 bytes).
- $\mathbf{p}_i$, a position vector in base coordinates (24 bytes).
- $\dot{\mathbf{p}}_i$, a velocity vector in base coordinates (24 bytes).
- ω_i,an angular velocity vector in base coordinates (24 bytes).

The Kinematic Bus encapsulates the reception, transformation, and transmission of cartesian state data as well as sensed data collection from the dynamic model, `DM` within a single `KinematicBus` class. Communications are through `SensorVector` and `MeasurandVector` objects updated at every timestep.

On receipt, a kinematic packet is transformed from the cartesian state of A_{i-1} to A_i through the following set of recursive Newton Euler equations:

$$\mathbf{R}_i = \mathbf{R}_{i-1}\,\mathbf{R}_i^{i-1}(q_i) \tag{5.173}$$

$$\mathbf{p}_i = \mathbf{R}_i\mathbf{p}_i^{i-1} + \mathbf{p}_{i-1} \tag{5.174}$$

$$\omega_i = \begin{cases} \omega_{i-1} + \mathbf{R}_i\dot{q}_i\mathbf{k}_i & \text{if revolute} \\ \omega_{i-1} & \text{if prismatic} \end{cases} \tag{5.175}$$

$$\dot{\mathbf{p}}_i = \begin{cases} \dot{\mathbf{p}}_{i-1} + \mathbf{R}_i\dot{q}_i\mathbf{k} \times (\mathbf{p}_i - \mathbf{p}_{i-1}) & \text{if revolute} \\ \dot{\mathbf{p}}_{i-1} + \mathbf{R}_i\dot{q}_i\mathbf{k} & \text{if prismatic} \end{cases} \tag{5.176}$$

where $\mathbf{k}$ is the link's revolute axis of rotation or the prismatic axis of translation (according to the Denavit Hartenberg convention [Denavit, 1955]) in link coordinates and $(q_i, \dot{q}_i)$ are the sensed joint positions and velocities respectively [2]. The local geometric model is limited to Denavit Hartenberg parameters.

5.6.3 The Global Goal `IOStream`

As mentioned in the description of `GP`, the global goal and link processes communicate through `IOStream` connections – pairs of `SensorVector` and `MeasurandVector` structures. Controller proxies receive global goal directions (such as global setpoints) through the `SensorVector`, while data can be transmitted to a global goal process through the `MeasurandVector` structure (such as end effector locations, obstacle avoidance assertions, etc.).

5.6.4 Controller Objects

A key feature of the MMS in general and `LP` in particular is controller modularity. This was greatly simplified through the adoption of the `Controller` abstract superclass for the description of local controllers. All controllers employed within an agent were subclasses of `Controller`. Though most local controllers used fixed setpoints specified within the robot database file, another variety of controller acted on behalf of the global goal process to influence the agents actions.

When a *global* objective is being pursued, `SI` launches the global control process, `GP` that generates and transmits instructions to the link processes, `LP`. This requires that each link has a mechanism for receiving and interpreting externally generated goal process instructions. Controllers that implement these reception protocols are *proxies*, acting on behalf of the global goal specifying a control effort to the agent. Controller

[2] As a second derivative of position, cartesian accelerations are considered too volatile to incorporate into the kinematic packet.

proxies object allows external goal processes to act as a local controller to a link object. These proxies are indistinguishable from any other controller object from the Link object's standpoint. However, a controller proxy differs from local controllers by establishing a communication link between an independent controller process and the link object.

The goal process (described earlier) continuously broadcasts goal objectives to the *client* controller proxies. Semantically one can interpret such transmissions as the arrival of global sensory data to a local controller. Depending on the control rate of the sensor the controller proxy may or may not act upon the information received from the global goal.

The `Controller` Array

Active controllers (and controller proxies) are stored within an array of M `Controller` objects, $\mathbf{u}_{\mathsf{A}}$:

$$\mathbf{u}_{\mathsf{A}} = [\mathbf{u}_1 \;\; \mathbf{u}_2 \;\; \dots \;\; \mathbf{u}_M]^T \tag{5.177}$$

. After the link process parses the link database file, legal `controller` database definitions are instantiated as subclasses of the `Controller` class and added to the controller array.

Once instantiated, the link process has no mechanism discriminating between controller objects treating each controller identically by invoking an `update()` message in all members of the controller array at each timestep.

5.6.5 Arbitration

In this agent implementation a simple linear arbitrator of the form described in equation (4.92) is used to weight controller array responses. The jth controller definition in the link database file can assign a constant arbitration coefficient k_j (by default $k_j = 1.0$). These coefficients are, in turn, stored as an arbitration vector **k**:

$$\mathbf{k}_{\mathsf{A}} = [\mathbf{k}_1 \;\; \mathbf{k}_2 \;\; \dots \;\; \mathbf{k}_M]^T \tag{5.178}$$

The agent output, $\mathbf{u}_c$, becomes:

$$\mathbf{u}_{\mathbf{C}} = \sum_{j=1}^{M} \mathbf{k}_j \mathbf{u}_j \tag{5.179}$$

As discussed in the following chapters, even this simple strategy can produce interesting 'self arbitrating' behaviour between rival goal systems.

5.6.6 Execution

At launch, link processes establish bus connections through the `SensorVector` and `MeasurandVector` objects to neighbouring links, and the terminal link (the end effector) establishes a similar connection through an `IOStream` connection to the global goal process.

The link cartesian state is updated upon confirmation of incoming data from the Kinematic State Bus describing the cartesian kinematic state of the lower link frame.

Once this data has been retrieved the Newton Euler transformations (either revolute or prismatic) are applied to the cartesian state and transmitted immediately to the next link process via the KSB.

If global goals have been specified, each link process awaits any global goal broadcasts. Each controller is then anonymously invoked and the responses combined within the linear arbitrator to form $\mathbf{u}_c$. Finally, $\mathbf{u}_c$ is transmitted to the Dynamic model as a motor torque.

As in other processes, the receipt of a `kill` notice invokes shutdown and cleanup as does the failure of interprocess communications. Again, termination occurs only after execution and historical data logs are completed.

5.7 Operational Overview

Figure 5.25 provides an overview of the operation of the Multiprocess Manipulator Simulator. At the beginning of each cycle a timestamp is transmitted to the `GP` and all `LP` processes. The timestamp triggers the base `LP` process to transmit the kinematic packet Ξ_1 to $\texttt{LP}_2$, in turn triggering a similar recursive transmissions down the kinematic chain to $\texttt{LP}_N$ and, if necessary, `GP`. With updated end effector data, `GP` is free to compute and broadcast the global goal couplet to all $\texttt{LP}_i$. On receipt of global goal couplet(s) each $\texttt{LP}_i$ computes and transmits a response to the dynamic modelling process, `DM`. `DM` incorporates these control efforts into the dynamic model and integrates the system to the next time step. Finally, the model transmits sensed data back to each link process.

5.8 Data Logs and Viewing

Only `GP`, `DM`, and `LP` processes generate experimental data, all stored in historical data logs. Rather than archiving results from every 1 ms. time step, all results data streams are sampled every five timesteps to reduce potentially enormous data files to manageable sizes. Nevertheless, running a 10 degree of freedom manipulator over a 40 second simulation run consumes 12 Mbytes.

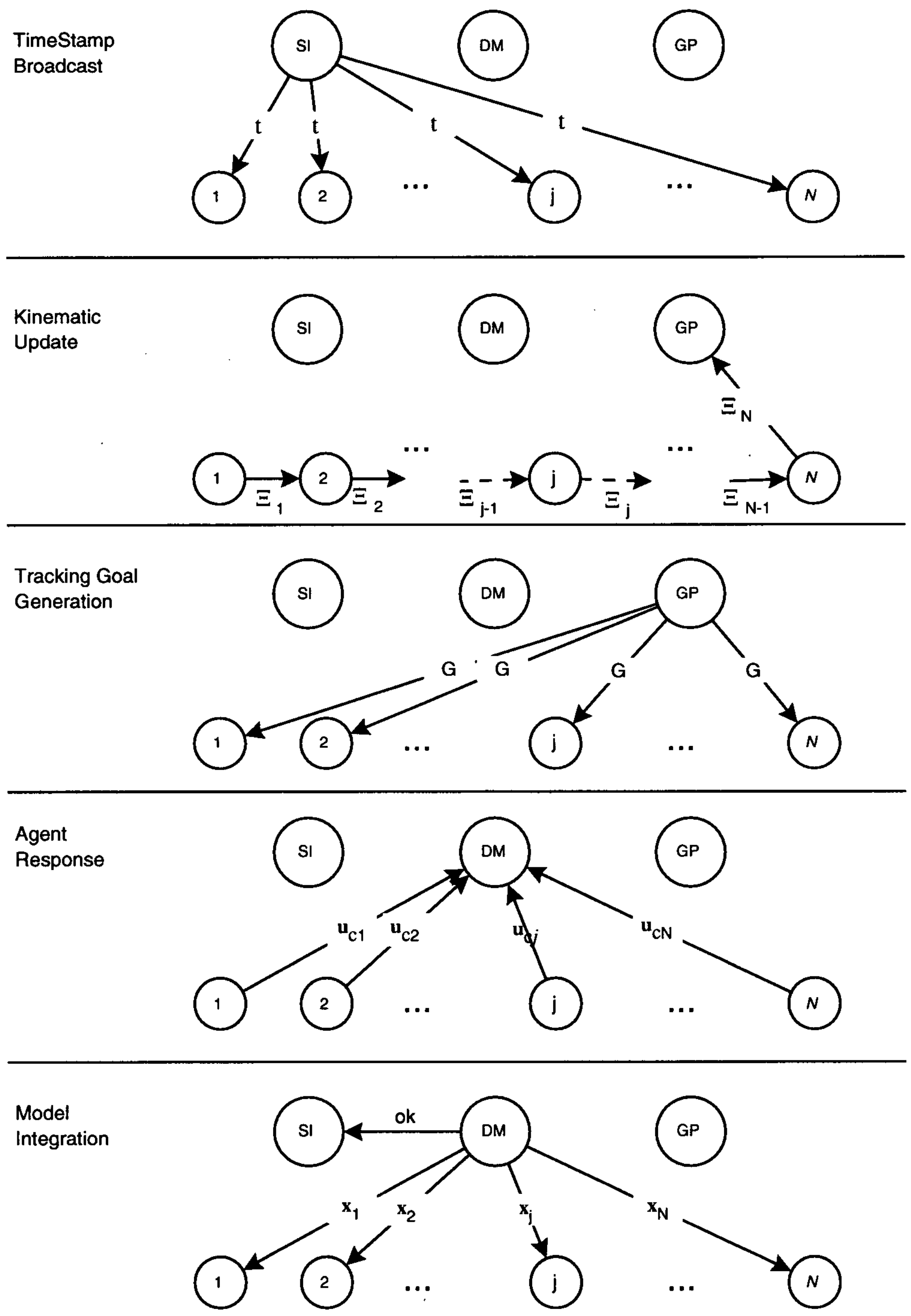


Figure 5.25: Interprocess data flow between the link processes $1 \ldots N$, global goal process GP, process manager SI, and dynamic model DM. Note that the link state $\mathbf{x}_i = [q_i \; \dot{q}_i]^T$.

Digesting and analyzing these data files was, itself, a significant undertaking. The MMS class hierarchy in conjunction with Pixar's excellent QuickRenderMan^{TM} image rendering library [Upstill, 1990] and superb NEXTSTEP^{TM} [NeXT, 1993] 'AppKit' libraries, greatly facilitated the rapid recovery, representation, manipulation, and archiving of robot images from these data files. Data plots were generated through the powerful MATLAB^{TM} environment.

5.9 Verification

Verification of such complex software systems must be a cumulative process, each software layer tested independently. By ensuring that vector mathematics modules (for example) are functioning to specification, only the algorithmic correctness of later modules (such as the dynamic modeler) need be verified. Extensive cumulative testing played a crucial role in establishing confidence in the simulator software.

While cumulative tests are necessary and sufficient for service utilities, they are ineffective in testing the algorithmic correctness of physical simulations. Therefore, the final verification step in the simulator development process is to validate the kinematic, dynamic, and mass matrix computations by comparing the simulator against a standard, verified, reference model.

To speed the verification process and ensure accurate implementation of a reference model, the dynamic modelling facilities of the simulator were verified against an established symbolic dynamics generator, `NEDYN` provided by Toogood [EL-Rayyes, 1990]. This remarkable software generates *symbolic* dynamic models for either rigid or flexible manipulators models in FORTRAN, further optimizing these models into the minimum number of operations. At once, this software provides arbitrary dynamic models, optimized and verified through previous testing. After minor conversion to C++, `NEDYN`'s output was used to verify the arbitrary dynamics generator, `DM` through comparative testing.

Though the specific verification reference model must be sufficiently rich to exercise all aspects of the modelling system, it must not be overly complex. Indeed as an error detection tool, it is important that discrepancies between the reference and tested models be easily resolved. For this reason a relatively simple planar manipulator model is sufficient to exercise the dynamic modeler and mass matrix computation system and yet is simple enough to provide a useful debugging tool. Based on previous matrix algebra verifications, planar manipulators provided sufficient complexity to demonstrate algorithmic correctness.

Parameter	Symbol	Value
Name		`link` i
Mass	m_i	1.000e+01 kg.
Pr. Inertias	$\mathbf{I}_{jj}$	[+0.0500 + 0.4938 + 0.4938] kg-m^2
centroid	$\mathbf{c}$	[−0.3750 + 0.0000 + 0.0000] m.
Type		R (revolute)
Init. Displacement	θ_{0_i} or d_{0_i}	$+0.000e+00°$ (or m)
Init. Velocity	v_{0_i}	$+0.000e+00°$ /s (or m/s)
Twist	α_i	$+0.000e+00°$
Length	d_i	$7.500e-01$ m.
Back EMF	K_{emf}	1.0
Saturation		$+1.000e+06$ (unsaturated) N-m

Table 5.4: Reference Link, Joint, and Motor parameters. Centroid is relative to link frame.

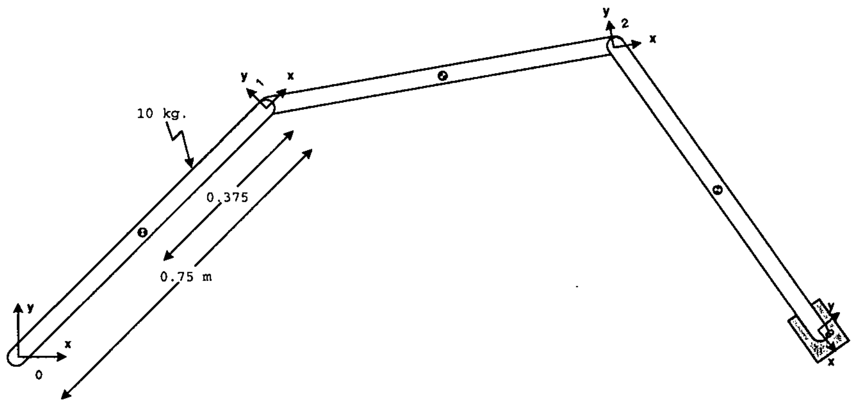


Figure 5.26: The 3R planar reference model used to verify `DM` against `NEDYN`.

5.9.1 A 3R Reference Model

One such example is the planar three degree of freedom robot depicted in figure 5.26 and composed of three identical links.

Within the robot database file, each link entry clearly identifies all the relevant parameters for a given link. A link name distinguishes each entry and provides a source of unique socket names. Link parameters such as the link mass, inertial and centroidal data (in local coordinates), and the denavit Hartenberg parameters, θ, d, α, ℓ, provide critical geometric data the assembly of both kinematic and dynamic models. Saturation is included to provide real world constraints on torque/force output from each actuator. These values are summarized in table 5.4.

These physical parameters are followed by controller specifications. For the verification runs, a simple PD

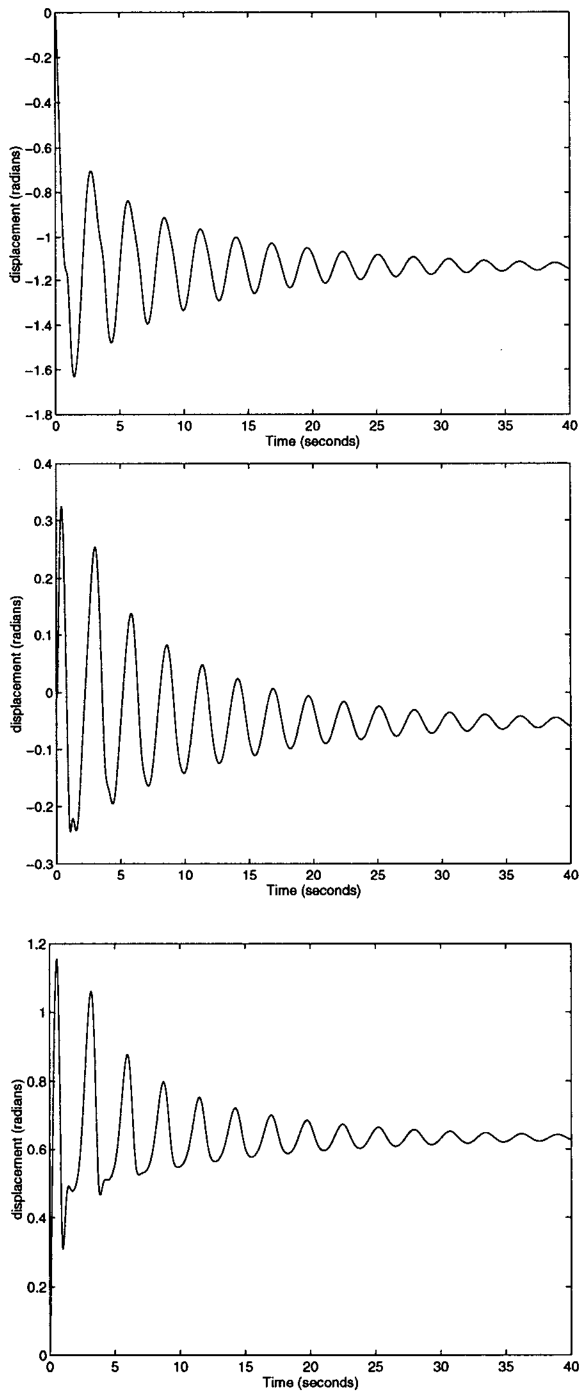


Figure 5.27: Here, the superimposed displacement responses of both the symbolic NEDYN and numerical 3R planar reference manipulator (link 1, top; link 2, middle; link 3, bottom) indicate excellent agreement between the two methods.

controller, `PID_joint_center`, described in detail later, with setpoint $\theta_i = +1.57$rad, position and velocity gains set to 100 and 20 respectively, operating at 120Hz. The payload data file specifies a 10 kilogram payload centered on the end effector. Since `PID_joint_center` controllers pursue the setpoint specified in the database entry, they do not require a central global goal generator, and `GP` is never launched.

The experimental results depicted in figure 5.27 demonstrate excellent agreement between the symbolically optimized NEDYN and the equivalent numerical methods described by Walker and Orin. Since NEDYN and the numerical modeler are based on identical Newton Euler methods and use identical manipulator models, identical output from the two systems is to be expected.

5.10 Summary

In this chapter we have reviewed the design and implementation of the Multiprocess Manipulator Simulator and verified its performance against a published numerical simulation system. In the following chapters, this simulator will be applied to the multiagent manipulator control problem and the goal and link processes expanded to implement multiple local and global goals. The next chapter, however, will examine the performance of the global goal system for trajectory tracking, through comparison of goal generation methods and demonstration of the global goal proxy concept.

Chapter 6

Global Goal Generation

6.1 Introduction

In chapter 4 resolved motion position control, resolved motion rate and acceleration control, and Jacobian transpose control were evaluated as global goal distribution mechanisms. Based on an examination of the structure of these inverse aggregate methods, RMRC and RMAC were shown to be less desirable than Jacobian transpose methods. Further, it was suggested that if each link's cartesian state could be sensed or computed, Jacobian transpose control becomes effectively decentralized. By projecting a global goal, an end effector force onto the ith link through the ith row of the Jacobian transpose, a global goal *couplet* was defined in equation (4.130):

$$\mathcal{G}_g \equiv \langle \mathbf{p}_{\mathrm{n}}, \mathbf{f}_d \rangle \tag{6.180}$$

where $\mathbf{p}_{\mathrm{n}}$ is the end effector position.

This chapter will explore a multiagent manipulation system based on these precepts. Having identified a *distribution* mechanism for $\mathbf{f}_d$ in the global goal proxy, a method of *generating* $\mathbf{f}_d$ must be selected from the following possible techniques: Resolved Motion Force Control, the Operational Space Formulation, and Configuration Control. Each of which develop joint *forces* based on an end effector *force vector*, $\mathbf{f}_d$, a function of a desired task space trajectory $\mathbf{x}_d(t)$:

$$\mathbf{f}_d = f_d(\ddot{\mathbf{x}}_d(t), \mathbf{x}_e(t), \dot{\mathbf{x}}_e(t)) \tag{6.181}$$

where $\mathbf{x}_e(t) = \mathbf{x}_d(t) - \mathbf{x}_a(t)$ and to varying degrees, employ a centralized kinematic model of the manipulation system. In the Operational Space Formulation cartesian trajectory tracking forces are generated through a task space feedforward manipulator model. In a simpler technique, Resolved Motion Force Control develops a force trajectory based on a feedforward payload model, compensating for 'unmodelled' dynamics with a force feedback stochastic estimator. Direct Adaptive Control (for clarity renamed here Resolved Motion Decentralized Adaptive Control or RMDAC) employs model reference adaptive controllers to compensate for both payload and manipulator dynamics. Appendix E briefly reviews the pros and cons of OSF and

RMFC as goal generator candidates, while the details of Resolved Motion Decentralized Adaptive Control, the preferred goal generator, are described below.

6.2 Global Goal Generation

A number of force trajectory generation methods have been proposed for Jacobian Transpose Control, including Khatib's *Operational Space Formulation* (OSF), Wu and Paul's *Resolved Motion Force Control*(RMFC), and Seraji's *Direct Adaptive Control* (DAC). Intended as centralized end effector control architectures that maintain a single, coherent, manipulator model, these techniques must be reevaluated as global goal generation systems. In short, what constitutes an appropriate global goal generator for multiagent manipulator control?

Recall that three forms of coordination span multiagent system control: explicit, implicit, and emergent, each representing a compromise between deterministic behaviour and robustness. Explicit coordination strategies regulate each agent's behaviour through a centralized model based process. Implicit coordination strategies control *global* behaviour without any team model. Emergent coordination methods produce *global* behaviour purely through agent interaction, without any global goal specification or modelling.

Local and global Behaviour are deterministic for systems under explicit coordination though they tend to be fragile in the face of unmodelled changes. By sacrificing some local determinism, implicitly coordinated systems maintain globally deterministic behaviour and gain robustness to unmodelled events. Local behaviour under emergent coordination is often unpredictable while global behaviour often cannot be analytically guaranteed. Nevertheless, these systems routinely fulfill global objectives while demonstrating resilience to environmental change.

For multiagent manipulator control, a balance between determinism and robustness must be maintained. With less modelling to achieve a desired behaviour comes less computation, lower expense, and the robustness to change. Yet, to be useful, global behaviour must also be deterministic. Clearly a compromise exploiting some form of implicit coordination is necessary.

In decentralizing Jacobian transpose control, a centralized kinematic model has been removed from the goal distribution mechanism, a significant step towards this compromise. However, the global goal generator, $\mathbf{f}_d$, too, must limit model dependency. Therefore, establishing model independence within a global goal generator is an important objective in implicit multiagent controller design.

Retaining centralized feedforward dynamic models of the manipulator, both OSF and RMFC are less attractive implicit coordination strategies (see appendix E) than Seraji's *Direct Adaptive Control.* This latter controller replaces the last vestige of a feedforward dynamic model with an adaptive task space controller that models an ideal generic trajectory tracking process rather than a specific manipulator geometry. In so doing this global goal generator becomes manipulator independent or *model-free*[1]. In the following section DAC will be explored in detail and its performance within a multiagent controller demonstrated.

6.2.1 Resolved Motion Decentralized Adaptive Control

Though the internal operation of any global goal generator, such as RMDAC, is a separate issue from multiagent control, the global performance of the multiagent team is dependent on the performance of this generator. Furthermore, the less manipulator within the generator, the more robust the global goal system is to environmental changes. A close examination of this global goal generator is, therefore, worthwhile. In 1987 Seraji [Seraji, 1987b] introduced *Direct Adaptive Control* and later *Configuration Control* of manipulators in cartesian space. To differentiate this technique from a further variation *Improved Configuration Control* for joint space controllers, these methods are henceforth referred to as *Resolved Motion Decentralized Adaptive Control* (RMDAC) in recognition of its cartesian adaptive control heritage.This technique differs from OSF and RMFC, in that a model reference adaptive controller (MRAC) generates the desired end effector forces based on an idealized trajectory tracking process, devoid of any manipulator model.

The controller may be derived from a linearized manipulator model about an initial condition P. At P, the manipulator is at joint positions $\mathbf{q}_0$, end effector position $\mathbf{x}_0$, and joint torque τ_0. By perturbing τ_0 slightly by $\delta\tau(t)$ to form $\tau(t) = \tau_0 + \delta\tau(t)$ the joint and end effector positions change by $\delta\mathbf{q}(t)$ and $\delta\mathbf{x}(t)$ to form $\mathbf{q}(t) = \mathbf{q}_0 + \delta\mathbf{q}(t)$ and $\mathbf{x}(t) = \mathbf{x}_0 + \delta\mathbf{x}(t)$. In [Seraji, 1987a], Seraji shows that by linearizing 2.34 about P:

$$\hat{\mathbf{A}}\delta\ddot{\mathbf{q}}(t) + \hat{\mathbf{B}}\delta\dot{\mathbf{q}}(t) + \hat{\mathbf{C}}\delta\mathbf{q}(t) = \delta\tau(t) \tag{6.182}$$

where $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$, and $\hat{\mathbf{C}}$ are constant $n \times n$ matrices dependent on P. By using equation (2.33) and (2.7), this expression can be transformed into cartesian space [2].

$$\mathbf{A}\delta\ddot{\mathbf{x}}(t) + \mathbf{B}\delta\dot{\mathbf{x}}(t) + \mathbf{C}\delta\mathbf{x}(t) = \delta\mathbf{f}(t) \tag{6.183}$$

where Seraji then defines $\delta\mathbf{r}(t)$ as an "incremental reference trajectory" in cartesian space. To track this trajectory, feedback and feedforward controllers are developed. The feedback controller provides a stable

[1] Model-free in the sense that the global goal neither estimates nor guarantees convergence of any manipulator parameters.

[2] Significantly, Seraji points out the system could be derived entirely in cartesian space and, therefore, irrespective of any manipulator model

closed loop control, ensuring asymptotic convergence of any error to zero:

$$\delta\mathbf{f}_1(t) = \mathbf{K}_p\delta\mathbf{x}_e(t) + \mathbf{K}_v\delta\dot{\mathbf{x}}_e(t) \tag{6.184}$$

where $\delta\mathbf{x}_e(t) = \delta\mathbf{r}(t) - \delta\mathbf{x}(t)$. The feedforward controller ensures that $\delta\mathbf{x}(t)$ tracks the reference trajectory $\delta\mathbf{r}(t)$ and is simply the inverse of the end effector model in equation (6.183):

$$\delta\mathbf{f}_2(t) = \mathbf{A}\delta\ddot{\mathbf{r}}(t) + \mathbf{B}\delta\dot{\mathbf{r}}(t) + \mathbf{C}\delta\mathbf{r}(t) \tag{6.185}$$

Substituting $\delta\mathbf{f}_1(t)$ and $\delta\mathbf{f}_2(t)$ into $\delta\mathbf{f}(t)$ and rewriting equation (6.183)

$$\mathbf{A}\delta\ddot{\mathbf{x}}_e(t) + (\mathbf{B} + \mathbf{K}_v)(t)\delta\dot{\mathbf{x}}_e + (\mathbf{C} + \mathbf{K}_p)\delta\mathbf{x}_e(t) = 0 \tag{6.186}$$

Now this control scheme works well for regulating small errors about the nominal operating point, P. However, the control of the end effector over a general trajectory, $\mathbf{r}(t)$, is the sum of the incremental control forces required to maintain the control at $\delta\mathbf{r}(t)$ and the nominal force $\mathbf{f}_o(t)$ required to drive the end effector along the trajectory, $\mathbf{r}(t)$:

$$\begin{aligned} \mathbf{f} &= \mathbf{f}_o + \delta\mathbf{f} && (6.187)\\ &= \mathbf{f}_o + \mathbf{K}_p\delta\mathbf{x}_e(t) + \mathbf{K}_v\delta\dot{\mathbf{x}}_e(t) + \mathbf{A}\delta\ddot{\mathbf{r}}(t) + \mathbf{B}\delta\dot{\mathbf{r}}(t) + \mathbf{C}\delta\mathbf{r}(t) && (6.188) \end{aligned}$$

Now defining the total reference trajectory and end effector positions as $\mathbf{r}(t) = \mathbf{r}_0 + \delta\mathbf{r}(t)$ and $\mathbf{x}(t) = \mathbf{x}_0 + \delta\mathbf{x}(t)$ respectively, equation (6.188) can be rewritten as:

$$\begin{aligned} \mathbf{f} &= \mathbf{d} + \mathbf{K}_p\mathbf{x}_e(t) + \mathbf{K}_v\dot{\mathbf{x}}_e(t) + \mathbf{A}\ddot{\mathbf{r}}(t) + \mathbf{B}\dot{\mathbf{r}}(t) + \mathbf{C}\mathbf{r}(t) && (6.189)\\ \mathbf{d} &= \mathbf{f}_o - \mathbf{A}\ddot{\mathbf{r}}_0 - \mathbf{B}\dot{\mathbf{r}}_0 - \mathbf{C}\mathbf{r}_0 && (6.190) \end{aligned}$$

By generalizing these equations to control the nonlinear end effector dynamics, Seraji establishes the following system:

$$\begin{aligned} \mathbf{M}(\mathbf{x},\dot{\mathbf{x}})\ddot{\mathbf{x}} + \mathbf{N}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{G}(\mathbf{x},\dot{\mathbf{x}})\mathbf{x} &= \mathbf{d}(t) + \mathbf{K}_p\mathbf{x}_e(t) + \mathbf{K}_v\dot{\mathbf{x}}_e(t)\\ &\quad + \mathbf{A}\ddot{\mathbf{r}}(t) + \mathbf{B}\dot{\mathbf{r}}(t) + \mathbf{C}\mathbf{r}(t) && (6.191) \end{aligned}$$

where $\mathbf{d}(t)$ corresponds to the operating point term $\mathbf{d}$ and is synthesized by the adaptive controller. Rewritten in terms of total tracking error $\mathbf{x}_e$:

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{x}}_e + (\mathbf{N} + \mathbf{K}_v)\dot{\mathbf{x}}_e + (\mathbf{G} + \mathbf{K}_p)\mathbf{x}_e &= \mathbf{d}(t) + (\mathbf{M} - \mathbf{A})\ddot{\mathbf{r}}(t) +\\ &\quad (\mathbf{N} - \mathbf{B})\dot{\mathbf{r}}(t) +\\ &\quad (\mathbf{G} - \mathbf{C})\mathbf{r}(t) && (6.192) \end{aligned}$$

From this equation it is clear that $\mathbf{d}(t)$ and $\mathbf{r}(t)$ drive the error system of the total controller. Therefore it is essential to adapt the gains of the system. Rearranging equation (6.192) into a state variable form:

$$\begin{aligned}\dot{\mathbf{z}}(t) &= \begin{bmatrix} 0 & \mathbf{I}_n \\ -\mathbf{M}^{-1}[\mathbf{G}+\mathbf{K}_p] & -\mathbf{M}^{-1}(\mathbf{N}+\mathbf{K}_v) \end{bmatrix} \mathbf{z}(t) + \begin{bmatrix} 0 \\ -\mathbf{M}^{-1}\mathbf{F}(t) \end{bmatrix} + \\ & \begin{bmatrix} 0 \\ -\mathbf{M}^{-1}(\mathbf{G}-\mathbf{C}) \end{bmatrix} \mathbf{r}(t) + \begin{bmatrix} 0 \\ -\mathbf{M}^{-1}(\mathbf{N}-\mathbf{B}) \end{bmatrix} \dot{\mathbf{r}}(t) + \begin{bmatrix} 0 \\ -\mathbf{M}^{-1}(\mathbf{M}-\mathbf{A}) \end{bmatrix} \ddot{\mathbf{r}}(t) \end{aligned} \tag{6.193}$$

where $\mathbf{z}(t) = [\mathbf{x}_e \ \ \dot{\mathbf{x}}_e]^T$. Now a simple linear reference model can be developed

$$\ddot{\mathbf{x}}_{e_m}(t) + \mathbf{D}_2\dot{\mathbf{x}}_{e_m}(t) + \mathbf{D}_1\mathbf{x}_{e_m}(t) = 0 \tag{6.194}$$

Defining $\mathbf{z}_m(t) = [\mathbf{x}_{e_m} \ \ \dot{\mathbf{x}}_{e_m}]^T$ the state space reference model becomes:

$$\dot{\mathbf{z}}_m(t) = \begin{bmatrix} 0 & \mathbf{I}_n \\ -\mathbf{D}_1 & -\mathbf{D}_2 \end{bmatrix} \mathbf{z}_m(t) \tag{6.195}$$

Since equation (6.195) is stable, there exists a positive definite $2n \times 2n$ matrix, $\mathbf{P}$, which satisfies the Lyapunov equation:

$$\mathbf{PD} + \mathbf{D}^T\mathbf{P} = -\mathbf{Q} \tag{6.196}$$

where $\mathbf{D}$ is the system matrix in equation (6.195).

By defining $\mathbf{e} = \mathbf{z}_m(t) - \mathbf{z}(t)$, and subtracting (6.193) from (6.195) a model reference error equation in $\mathbf{e}$ can be assembled. Seraji then uses a simple design method [Seraji, 1989c] to generate a Lyapunov function based on this equation and, in solving for $\mathbf{P}$, produces adaptation laws ensuring the convergence of $\mathbf{z}(t)$ to $\mathbf{z}_m(t)$:

$$\mathbf{v}(t) = \text{diag}(w_{pi})\mathbf{x}_e + \text{diag}(w_{vi})\dot{\mathbf{x}}_e \tag{6.197}$$

$$\mathbf{d}(t) = \mathbf{d}(0) + \delta_1 \int_0^t \mathbf{v}(t)dt + \delta_2\mathbf{v}(t) \tag{6.198}$$

$$\mathbf{K}_p(t) = \mathbf{K}_p(0) + \alpha_1 \int_0^t \mathbf{v}(t)\mathbf{x}_e(t)dt + \alpha_2\mathbf{v}^T(t)\mathbf{x}_e(t) \tag{6.199}$$

$$\mathbf{K}_v(t) = \mathbf{K}_v(0) + \beta_1 \int_0^t \mathbf{v}(t)\dot{\mathbf{x}}_e(t)dt + \beta_2\mathbf{v}^T(t)\dot{\mathbf{x}}_e(t) \tag{6.200}$$

$$\mathbf{C}(t) = \mathbf{C}(0) + \nu_1 \int_0^t \mathbf{v}(t)\mathbf{r}(t)dt + \nu_2\mathbf{v}^T(t)\mathbf{r}(t) \tag{6.201}$$

$$\mathbf{B}(t) = \mathbf{B}(0) + \gamma_1 \int_0^t \mathbf{v}(t)\dot{\mathbf{r}}(t)dt + \gamma_2\mathbf{v}^T(t)\dot{\mathbf{r}}(t) \tag{6.202}$$

$$\mathbf{A}(t) = \mathbf{A}(0) + \lambda_1 \int_0^t \mathbf{v}(t)\ddot{\mathbf{r}}(t)dt + \lambda_2\mathbf{v}^T(t)\ddot{\mathbf{r}}(t) \tag{6.203}$$

where $\{\delta_1, \alpha_1, \beta_1, \nu_1, \lambda_1\}$ are positive scalar integral gains and $\{\delta_2, \alpha_2, \beta_2, \nu_2, \lambda_2\}$ are zero or positive gains and $\{w_{pi}, w_{vi}\}$ are positive scalar weighting factors. By adaptively modifying the force vector $\mathbf{f}(t)$ in this way, $\mathbf{z}(t)$ will converge to $\mathbf{z}_m(t)$:

$$\begin{aligned}\mathbf{f}(t) &= \mathbf{d}(t) + \mathbf{C}(t)\mathbf{r}(t) + \mathbf{B}(t)\dot{\mathbf{r}}(t) + \mathbf{A}(t)\ddot{\mathbf{r}}(t) + \\ &\quad \mathbf{K}_p(t)\mathbf{x}_e(t) + \mathbf{K}_v(t)\dot{\mathbf{x}}_e(t) \qquad (6.204)\end{aligned}$$

Later, in 1989, Seraji [Seraji, 1989b] described a further variation on this approach, *Configuration Control*, by decentralizing the computations in cartesian space. In effect the expression in equations (6.197) through (6.203) appear virtually identical – but are scalars equations applied in each cartesian direction or

$$\begin{aligned}f_i(t) &= d_i(t) + c_i(t)\mathbf{r}_i(t) + b_i(t)\dot{\mathbf{r}}_i(t) + a_i(t)\ddot{\mathbf{r}}_i(t) + \\ &\quad K_{pi}(t)\mathbf{x}_e(t) + K_{vi}(t)\dot{\mathbf{x}}_e(t) \qquad (6.205)\end{aligned}$$

where, as before, $\{a_i, b_i, c_i\}$ are adaptive feedforward gains and $\{K_{pi}, K_{di}\}$ are adaptive PD feedback gains and d_i is an auxiliary signal to improve transient performance and $i : 1 \ldots n$ and $\mathbf{r}(t) \in \mathbf{R}^n$. As before, these coefficients are an integral of the generic form:

$$k_i(v(t), e(t), t) = k_i(0) + u_{1i}\int_0^t v_i(t)e_i(t)dt + u_{2i}v_i(t)e_i(t) \qquad (6.206)$$

where u_{1i} are positive scalar gains, u_{2i} are zero or positive, and $e_i(t)$ is the variable modified by the coefficient (e.g. $k_i(t) = K_{vi},\;\; e_i(t) = \dot{\mathbf{x}}_e(t)$). Again $r_i(t)$ is a weighted error of the form:

$$r_i(t) = w_{pi}\dot{\mathbf{x}}_{ei} + w_{vi}\dot{\mathbf{x}}_{ei} \qquad (6.207)$$

where w_{pi} and w_{vi} are weighting gains.

In summary Seraji noted the following characteristics of this algorithm:

- Computation is extremely simple, using discrete trapezoidal integration. For the kth time step:

$$\begin{aligned}f_i(k) &= d_i(k) + c_i(k)\mathbf{r}_i(k) + b_i(k)\dot{\mathbf{r}}_i(k) + a_i(k)\ddot{\mathbf{r}}_i(k) + \\ &\quad K_{pi}(k)\mathbf{x}_e(k) + K_{vi}(k)\dot{\mathbf{x}}_e(k) \qquad (6.208)\\ r_i(k) &= w_{pi}\dot{\mathbf{x}}_{ei}(k) + w_{vi}\dot{\mathbf{x}}_{ei}(k) \qquad (6.209)\\ k_i(k) &= k_i(k-1) + u_{1i}\frac{\Delta T}{2}[v_i(k)e_i(k) + v_i(i-1)e_i(i-1)] + \\ &\quad u_{2i}[v_i(k)e_i(k) - v_i(k-1)e_i(k-1)] \qquad (6.210)\end{aligned}$$

  governed by the task space dimension, M, the computational cost is $O(M)$.

- The manipulator parameters are assumed to be 'slowly time varying'.
- Initial manipulator parameters are not required to ensure convergence.
- Convergence is guaranteed through the Lyapunov Design method [Seraji, 1989b].
- The rate of convergence is governed by the integral adaptive gains.
- The rate of convergence is independent of initial values.

Indeed, Seraji demonstrated that even gross violations of the 'slowly time varying' assumption parameters does not destabilize the system.

6.3 Multiagent Controller Performance

Within the context of the multiprocess manipulator simulator, the RMDAC global goal generator is implemented as a subclass of `Controller` within the global goal process, `GP`. This subclass, `RMDAController`, retrieves the end effector location, $\mathbf{p}_{\mathrm{n}}$, through the kinematic bus, computes $\mathbf{f}_d$ through equations (6.210), and *broadcasts* both $\mathbf{p}_{\mathrm{n}}$, and $\mathbf{f}_d$ as the global goal couplet:

$$\mathcal{G} \equiv \langle \mathbf{p}_{\mathrm{n}}, \mathbf{f}_d \rangle \tag{6.211}$$

to link agents $\mathsf{A}_j : \; 1 \leq j \leq N$. Within each agent process, `LP`, the trajectory tracking behaviour controller observes this broadcast and commands the joint motor (or dynamic model) with the output:

$$\mathbf{u}_{cj} = \mathbf{u}_{\mathsf{track}} = \mathbf{J}_j^T(\mathbf{p}_{j-1}(t), \mathbf{p}_n(t))\mathbf{f}_d(t) \tag{6.212}$$

In the following section, a simulated manipulator (see figure 6.28) under multiagent control will demonstrate this technique through the implicit coordination of a reference manipulator and payload along a specified trajectory. Below, the reference manipulator, payload, and trajectory used in this simulation are described.

The Manipulator

The reference robot is a redundant planar revolute manipulator composed of ten links. Each link is of the form described in table 6.5. Inertia parameters assume a 0.75 metre cylindrical link 0.2 m in diameter with a mass of 10 kilograms.

Parameter	Symbol	Value
Name	`Name`	`link` i
Mass	m_i	1.000e+01 kg.
Pr. Inertias $[\mathbf{I}_{xx}, \mathbf{I}_{yy}, \mathbf{I}_{zz}]$	$\mathbf{I}_{jj}$	$[+0.0500 \;\; +0.4938 \;\; +0.4938]$ kg-m^2
centroid	$\mathbf{c}$	$[-0.3750 \;\; +0.0000 \;\; +0.0000]$ m.
Type	`Type`	`R` (revolute)
Initial Displacement	θ_{0_i} or d_{0_i}	$+0.000e+00°$
Initial Velocity	v_{0_i}	$+0.000e+00°$ per sec. (or m/s)
Twist	α_i	$+0.000e+00°$
Length	d_i	$7.500e-01$ m.
Back emf constant	K_{emf}	1.0
Saturation	`Saturation`	$+1.000e+06$ (unsaturated) N-m

Table 6.5: Reference Link, Joint, and Motor parameters. Centroid is relative to link frame.

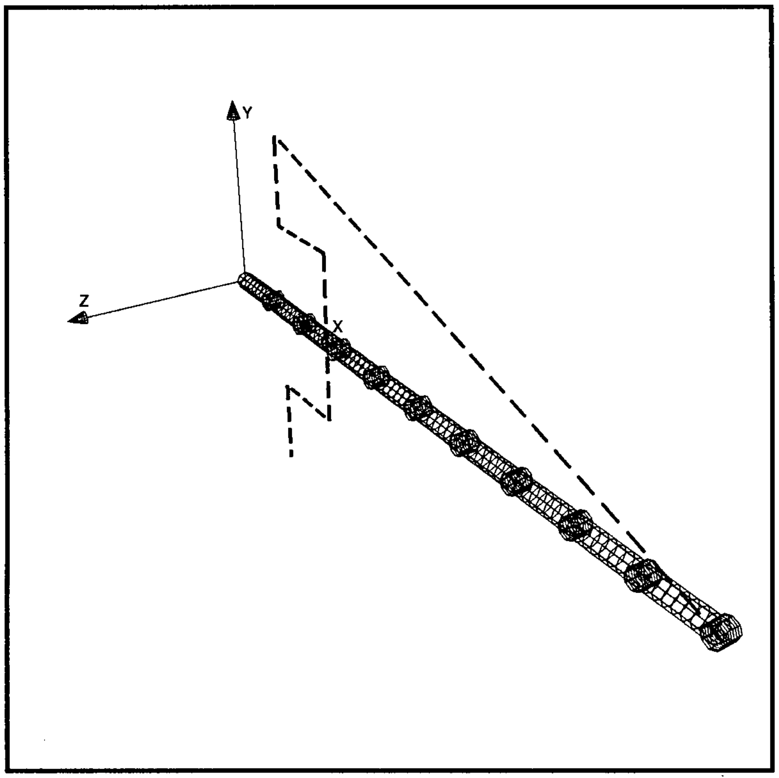


Figure 6.28: The Reference Manipulator: a ten degree of freedom planar revolute manipulator initially lying along the **x** axis.

Parameter	Symbol	Value
Mass	m_p	$1.000e+01$ kg.
Pr. Inertias $[\mathbf{I}_{xx}, \mathbf{I}_{yy}, \mathbf{I}_{zz}]$	$\mathbf{I}_{p_{jj}}$	$[+0.4938 \quad +0.4938 \quad +0.4938]$ kg-m^2
Centroid	$\mathbf{c}$	$[+0.00 \quad +0.00 \quad +0.00]$ m.

Table 6.6: Reference Payload parameters. Centroid is relative to gripper frame.

Time (sec.)	X (m)	Y (m)	Angle (rad.)
10.00	1.00e+00	2.00e+00	0.00e+00
16.00	1.00e+00	1.00e+00	1.57e+00
22.00	2.00e+00	1.00e+00	1.57e+00
28.00	2.00e+00	-1.00e+00	1.57e+00
34.00	1.00e+00	-1.00e+00	1.57e+00
40.00	1.00e+00	-2.00e+00	1.57e+00

Table 6.7: Knotpoints on the reference trajectory.

As mentioned previously, the motor model of chapter 2 is included within the MMS dynamic modeler, `DM`. The values K_k and K_{b_k} and B_{eff_k} may be set within the robot database file, but are left to the default (and arbitrary) values 0.5, 1.0, and 0.5 respectively.

The Payload

The payload, centered on the gripper (the coordinate frame of link n) of the manipulator, has been chosen to be identical to the mass and inertia of a manipulator link (see table 6.6). In this way, payload sensitivity studies may be assessed in terms of simple manipulator/payload mass ratios.

The Trajectory

The reference trajectory specification ("SquareWave.spt") describes a planar trajectory composed of 6 cubic segments (i.e. a straight line segment with parabolic velocity profile) of 40 seconds duration. A simple orientation trajectory rotates the end effector counterclockwise 90 degrees along the first leg and constant during the remaining trajectory. See table 6.7 and figure 6.29.

The trajectory presents some difficulties for the Jacobian based Algorithms. Since the manipulator is initially homed along the $\mathbf{x}$ axis, Jacobian *inverse* algorithms may encounter a kinematic singularity if motion along the $\mathbf{x}$ axis is required. Though the trajectory does not specify such motions, controller dynamics may demand such motions nevertheless resulting in numerical instability. The shallow inclination of the first

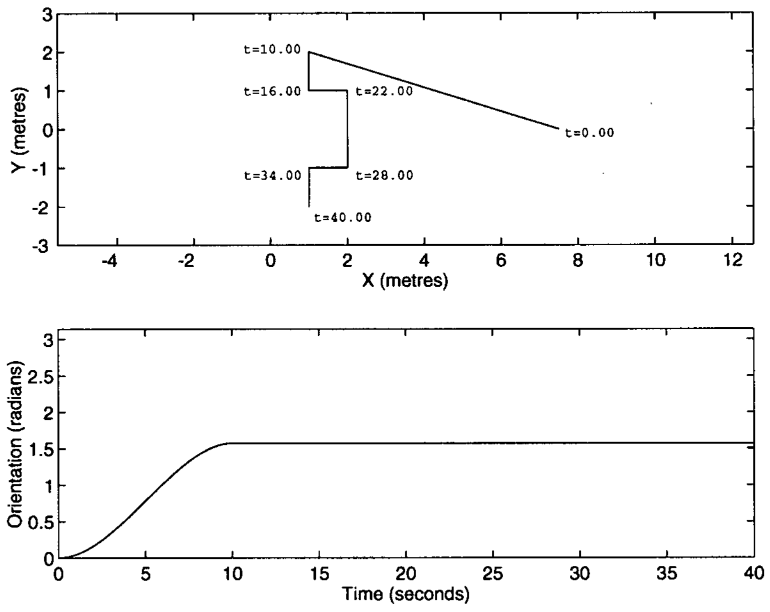


Figure 6.29: The Reference end effector position (top) and orientation (bottom) trajectories. Each segment employs a parabolic velocity profile in both position and orientation. The position trajectory plot is annotated with time milestones.

leg to the **x**-axis, relatively small initial velocity and acceleration vectors, and poor initial manipulator position, means *Transpose* algorithms tend to produce a delayed wallowing response. This has the effect of exaggerating integral wind up in adaptive controller as shown later.

6.3.1 Trajectory Tracking Performance

Having selected a set of integral gains for the RMDAC goal generator through a trial and error process (see table 6.8 and appendix D), the first experiment examines the trajectory tracking performance of the multiagent team. Initialized with the desired reference trajectory, the global goal process `GP` generates the global goal couplet, $\langle \mathbf{x}_n, \mathbf{f}_d \rangle$, through an `RMDAController` object. The couplet is broadcast to a team of ten link agents, each controlling a link by observing and responding to the global goal couplet through a global goal proxy.

The results of this demonstration, depicted in figure 6.31, clearly show that, despite decentralization, the manipulation system exhibits coordinated trajectory tracking behaviour. Note that while the end effector appears to track the desired trajectory, the manipulator seems to adopt arbitrary, 'jumbled', configurations. Since a geometric interference engine was not implemented in `DM`, physically impossible configurations are often observed. Nevertheless, this experiment demonstrates that global behaviour can be deterministic without resorting to local behaviour specification.

A root mean square (RMS) measure characterizes the end effector's tracking accuracy over the reference trajectory and is defined as:

$$\mathbf{e}_{\text{rms}} = \left[\frac{1}{N_s}\sum_{k=0}^{N_s}(\mathbf{p}_d(k) - \mathbf{p}_n(k))^T(\mathbf{p}_d(k) - \mathbf{p}_n(k))\right]^{\frac{1}{2}} \tag{6.213}$$

where N_s is the total number of timesteps. Similarly, the planar orientation error is measured through the RMS of the scalar rotation about the global $\mathbf{z}$ axis:

$$\theta_{\text{rms}} = \left[\frac{1}{N_s}\sum_{k=0}^{N_s}(\theta_d(k) - \theta_n(k))^T(\theta_d(k) - \theta_n(k))\right]^{\frac{1}{2}} \tag{6.214}$$

Figure 6.30 depicts the trajectory tracking error and end effector trajectory while table 6.9 documents the RMS position and orientation error over a step trajectory , the reference trajectory and a spiral trajectory.

The step trajectory simply jumps from $\mathbf{x}_{\text{start}} = [7.5\ 0.0\ 0.0]$ to $\mathbf{x}_{\text{finish}} = [6.5\ 1.0\ 0.0]$ and from $\theta_z = 0.0$ to $\theta_z = 1.57$ radians in rotation at $t = 1.0$s. This trajectory was primarily used to compare goal generation strategies. See appendix E.

Since the initial end effector position lies near a kinematic singularity and the initial end effector forces are small, initial tracking of the reference trajectory is characterized by large overshooting oscillations as RMDAC works to discover the parameters of the idealized trajectory tracking process. Trajectory tracking improves significantly as the desired trajectory exits this region. Despite these excursions, the RMS error over the total trajectory is limited to 2.8 mm. Similar, though smaller, end effector perturbations may be observed at segment transitions in the reference trajectory (16.00, 22.00, 28.00, and 34.00 seconds).

The global goal generator is equally effective over arbitrary trajectories. Figures 6.32 and 6.33 demonstrate the multiagent system's trajectory tracking performance over a 'spiral' trajectory. In this trajectory both the radius and the angular displacement about an axis $\mathbf{x}_{\text{axis}} = [4.5\ 0.0\ 0.0]$ vary according to a parabolic velocity profile from $\mathbf{x}_{\text{start}} = [4.5\ 2.0\ 0.0]$ to $\mathbf{x}_{\text{finish}} = [4.5\ -0.5\ 0.0]$. RMS position and orientation results are reported in table 6.9. It is apparent that centripetal accelerations encountered between 15 and 25 seconds, the maximum angular velocity of the trajectory about $\mathbf{x}_{\text{axis}}$, drives position and orientation errors on this trajectory.

The simulation does not occur in real time, as it is dominated by relatively slow socket interprocess communications and model integration. Typically, a 40.00 second run of the reference manipulator on an

SGI Indigo2 using an integration step size of 1 millisecond takes approximately 15 minutes. Define the *normalized execution time* as:

$$\Delta T_N = \frac{T_{\text{finish}} - T_{\text{start}}}{N \ N_s} \tag{6.215}$$

Where T_{start} and T_{finish} are real clock start and end times in seconds and N is the manipulator's degrees of freedom. The normalized execution time, the time committed to simulate a degree of freedom per timestep, is 2.2 ms for this reference scenario. Such measures are greatly affected by interprocess communications (i.e. disk and network) loads.

6.3.2 Model Free Goal Regulation

A lingering question, however, is whether the global goal generator, RMDAC, is actually a centralized manipulator controller harbouring a manipulator model. If this were so, RMDAC's performance would be governed by changes to the configuration and/or nonlinearity of the manipulator model. The following experiment demonstrates that RMDAC is insensitive to these factors, reinforcing that this goal generator is regulating an ideal dynamic tracking process and not any particular robot dynamic model. In effect, RMDAC is manipulator model free.

By examining tracking performance of a variety of planar manipulator configurations, the model-free characteristics of RMDAC can be demonstrated. In this experiment 5, 10, 15, and 20 degree of freedom manipulators are applied to identical reference payloads and trajectories.

Each manipulator has an identical mass and fully extended length to the reference manipulator and all links within a given manipulator possess identical physical parameters. Thus as N rises, link inertias, length, and centroid dimensions fall while agents remain computationally identical. This method permits an investigation of the sensitivity of RMDAC to manipulator design.

Table 6.10 documents RMDAC's relative insensitivity to changes in the robot model. Indeed, higher degrees of freedom (and therefore smaller links) appear to improve tracking slightly, probably due to the smaller link inertias in the higher degree of freedom manipulators. The end effector of the three additional cases may be observed within figures 6.34,6.35, and 6.36. Note the exaggerated end effector error in the five link case in comparison to the relatively smooth error curvature of the twenty link run. End effector error magnitudes are smaller and of shorter duration as the number of links rise.

Given identical integral gains, varying manipulator inertia influences initial trajectory tracking (i.e. the convergence rate of the adaptive controller) as expected. Once converged, however, the end effector tracks

	w_{pi}	w_{vi}	f	α_{1i}	β_{1i}	ν_{1i}	γ_{1i}	λ_{1i}
linear	1800	800	6	4	4	1	1	2
rotation	500	100	6	4	4	1	1	2

Table 6.8: The reference RMDAC 'Best Gains' determined through trial and error.

Trajectories	$\mathbf{e}_{\text{rms}}$	θ_{rms}
Step	1.132e-01	9.179e-02
Reference	2.779e-03	2.850e-02
Spiral	4.348e-03	7.372e-03

Table 6.9: 'Best Gains' RMS error in position and orientation for 'step', 'reference' and 'spiral' trajectories.

the desired trajectory regardless of the manipulator's order of nonlinearity. As long as these parameters remain slowly time varying, convergence is guaranteed regardless of manipulator complexity.

Normalized execution times are almost constant, indicating near linearity in degrees of freedom and that a real world, multiprocess implementation of a multiagent manipulator architecture[3] is *constant* in time. In a real world system, the dominant *computational* load of the MMS, an $O(N^3)$ dynamic model [Walker, 1982] disappears. Similarly the larger interprocess *communications* load, of $O(N)$, that dominate MMS overall could be drastically reduced or removed through more efficient design. An efficient implementation of multiagent control would exploit multiple processors and an *asynchronous* kinematic bus, rendering execution time independent of agent team size. Unlike the MMS synchronous bus in which communications are triggered by an external clock, asynchronous buses allow agents to read and/or write shared data at any time.

In short, this experiment demonstrates the feasibility of a decentralized, multiagent manipulator architecture and the relative independence of the RMDAC global goal generator to manipulator configuration.

6.3.3 Actuator Saturation

In previous demonstrations, each agent's motor could generate arbitrarily large torques. In reality, motor performance saturates at some torque value, $\mathbf{u}_{\text{sat}}$. In the following demonstration, the effects of saturation on trajectory tracking will be examined. With saturation limit of $\mathbf{u}_{\text{sat}} = 1000$ N-m applied to all joint actuators, each link agent will, again, adopt equation (6.212) as the global goal proxy modified to prevent

[3]Ideally, a multiprocess task that exhibiting $O(N)$ execution time on a single CPU (such as MMS) is constant time in a process per CPU environment.

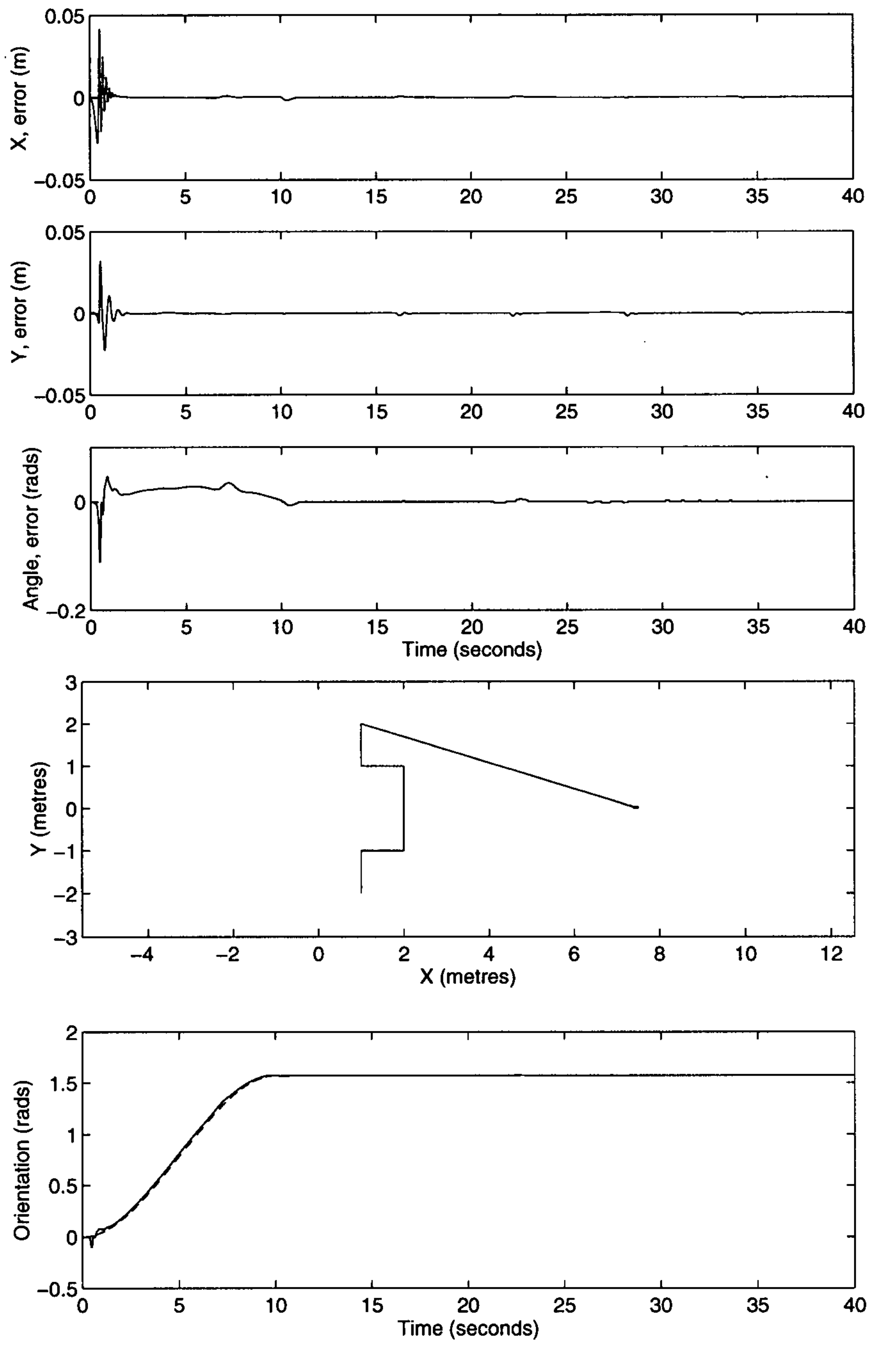


Figure 6.30: End effector trajectory tracking under 'Best Gains' RMDAC. Note oscillation prior to convergence.

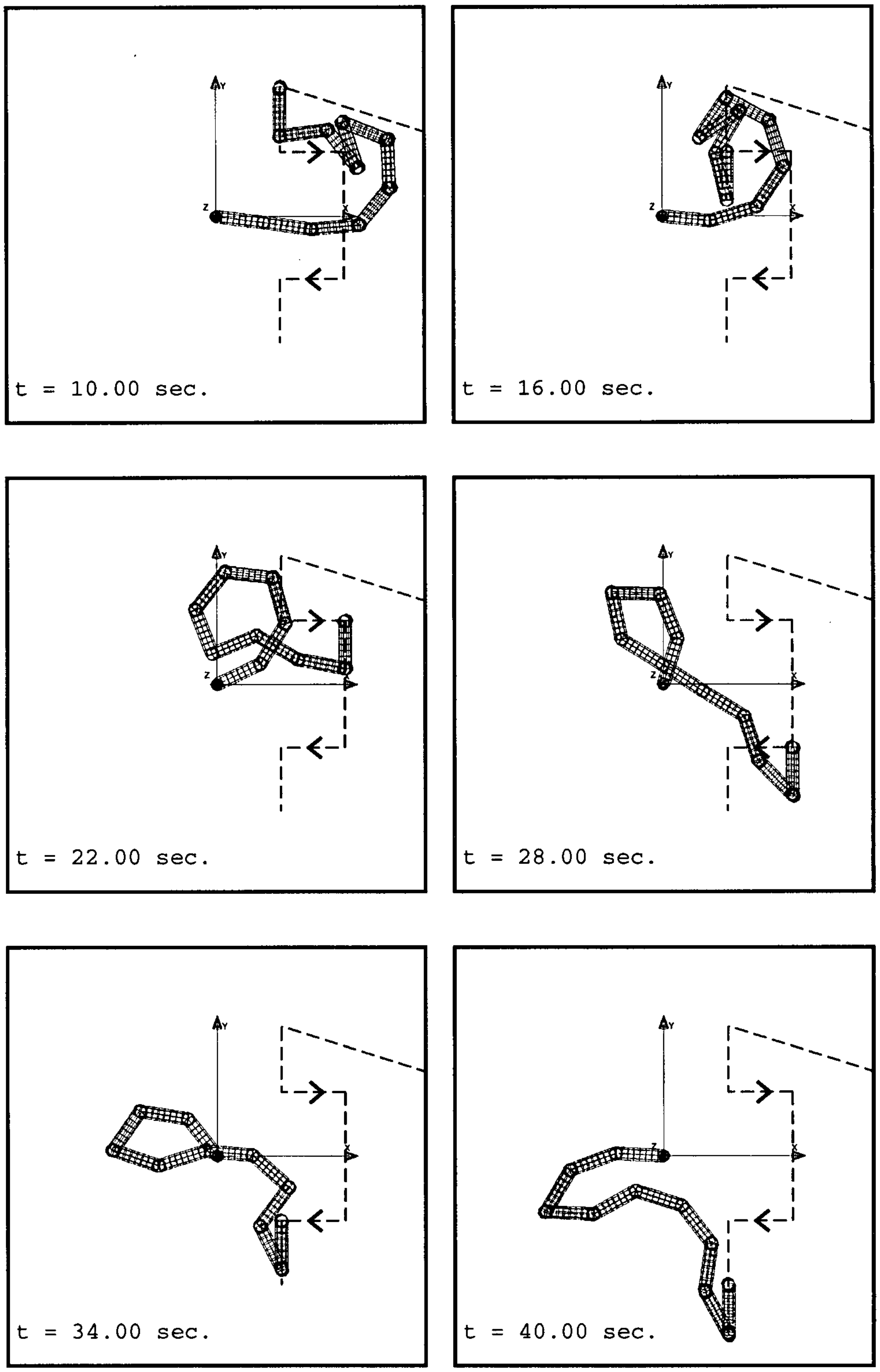


Figure 6.31: Trajectory tracking history for the reference trajectory under 'Best Gains' RMDAC.

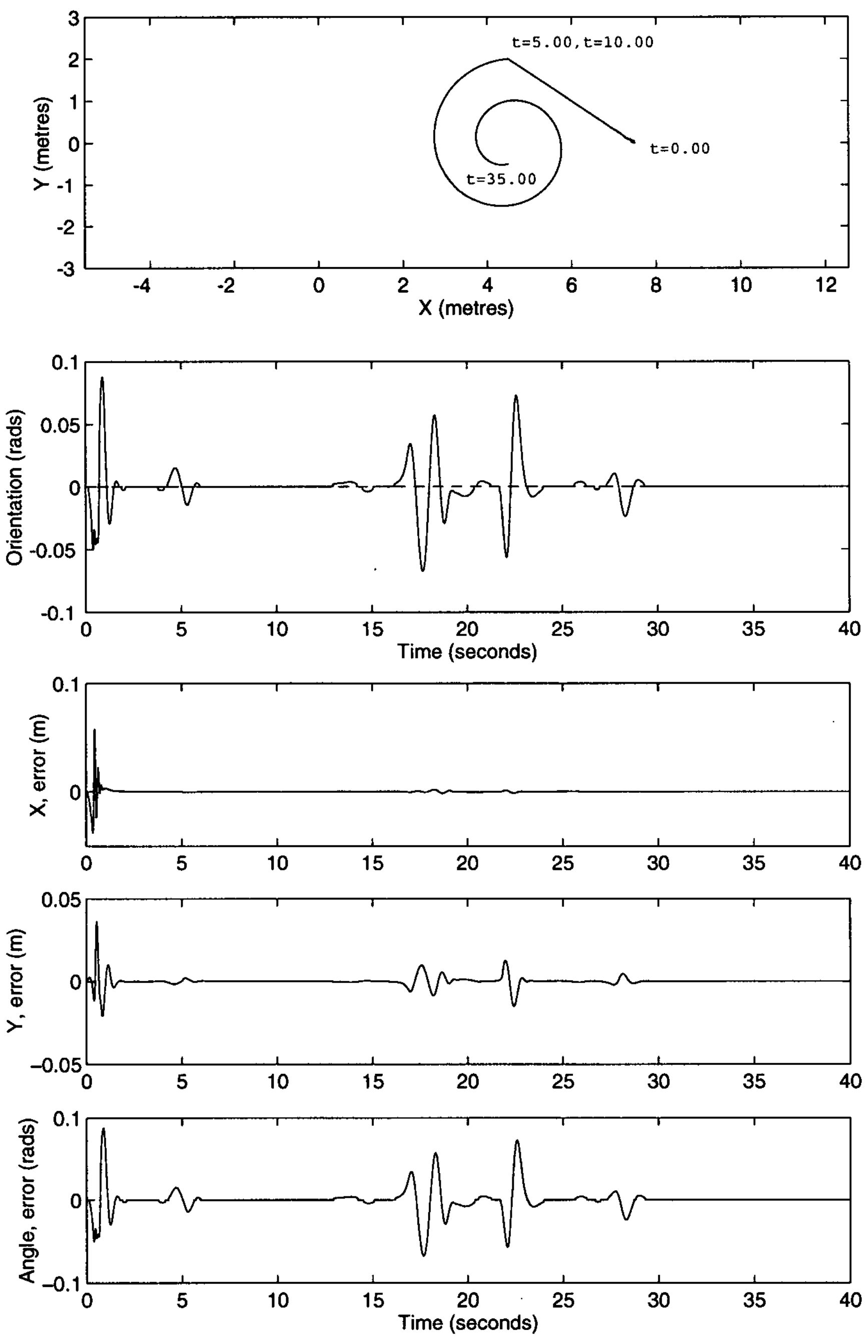


Figure 6.32: End effector tracking performance over an unusual spiral trajectory in which the radius and angular displacement vary according to a parabolic velocity profile.

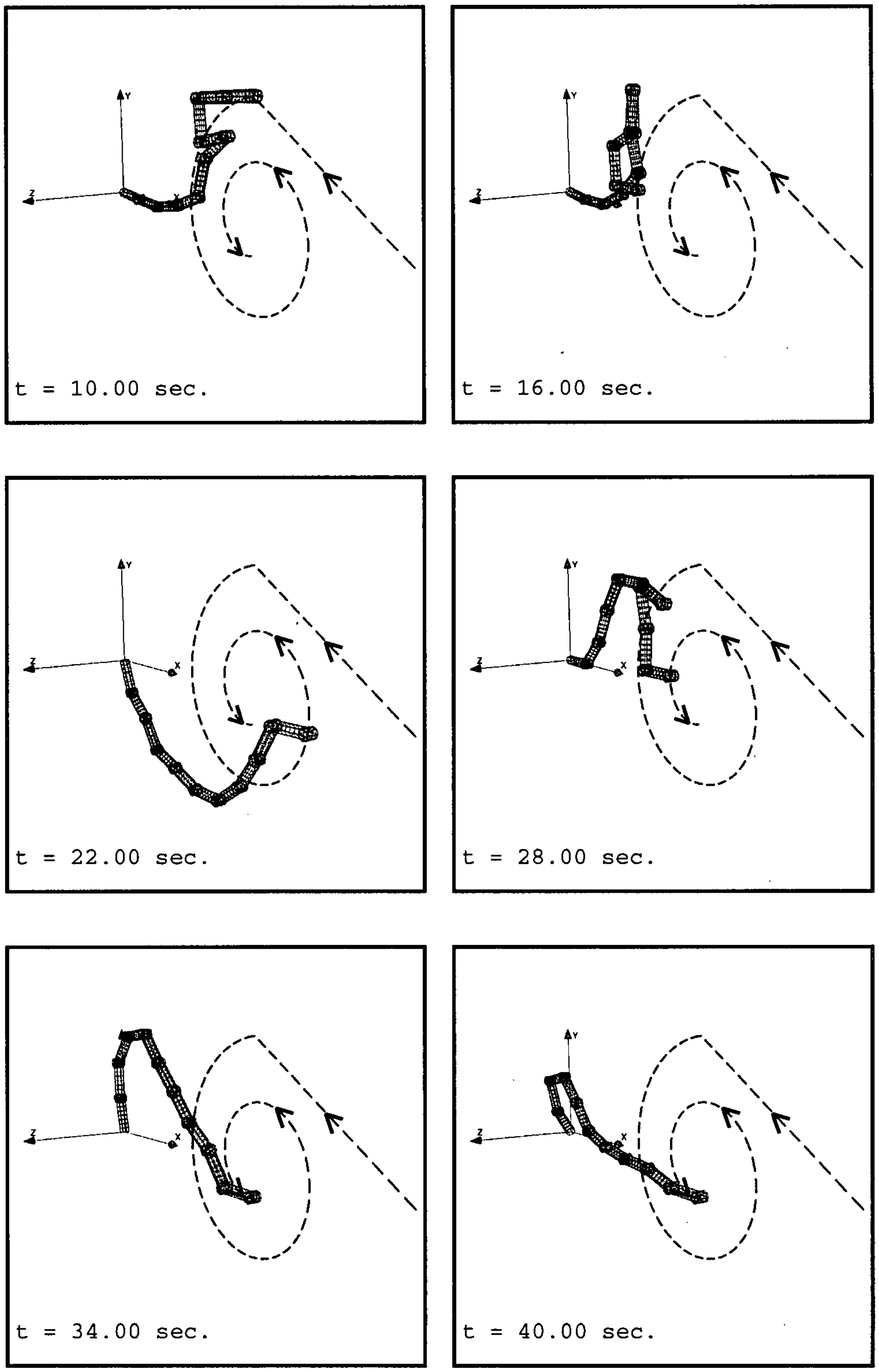


Figure 6.33: Trajectory tracking history for the spiral trajectory under 'Best Gains' RMDAC.

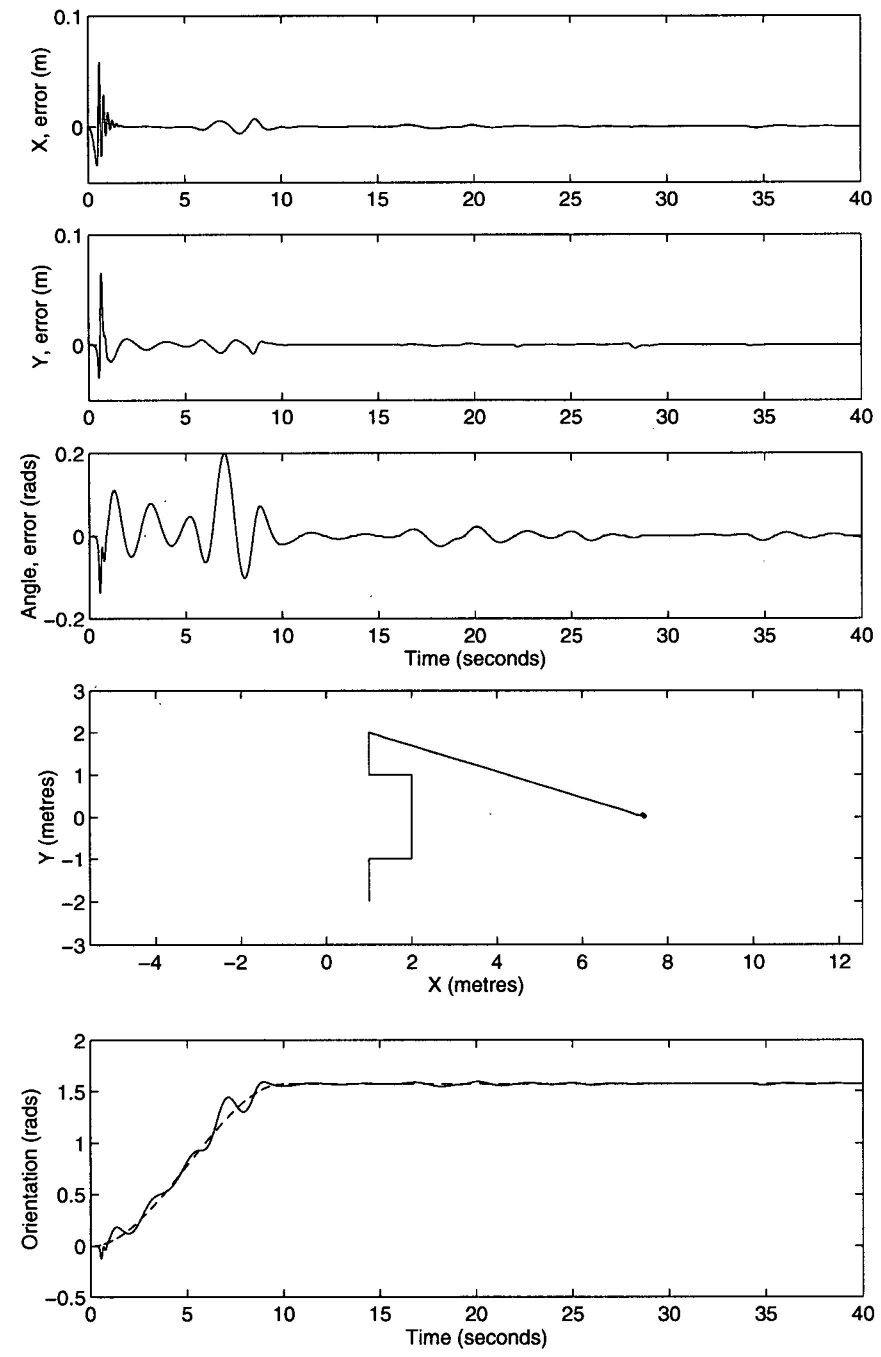


Figure 6.34: End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 5 degree of freedom planar manipulator.

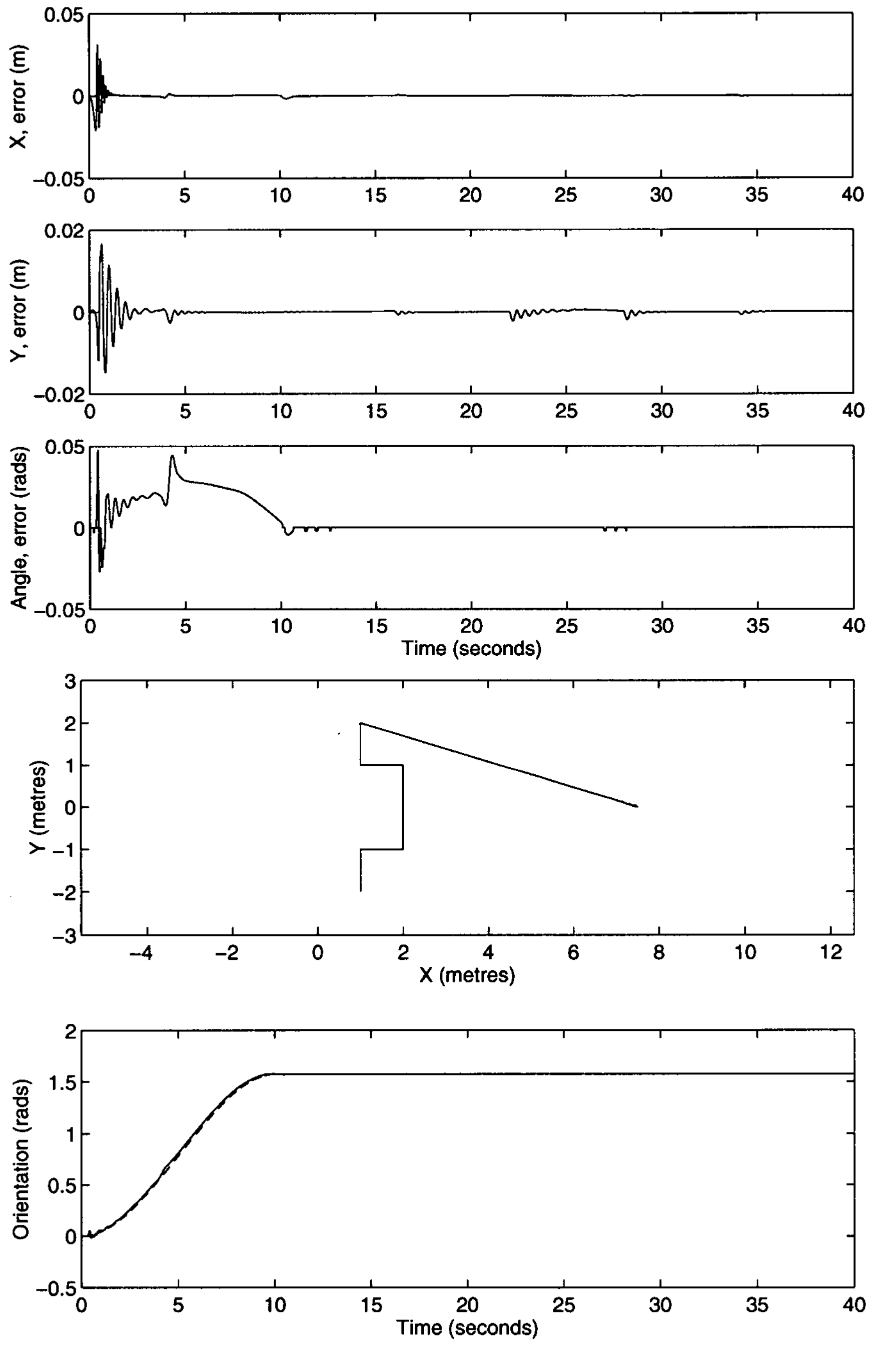


Figure 6.35: End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 15 degree of freedom planar manipulator.

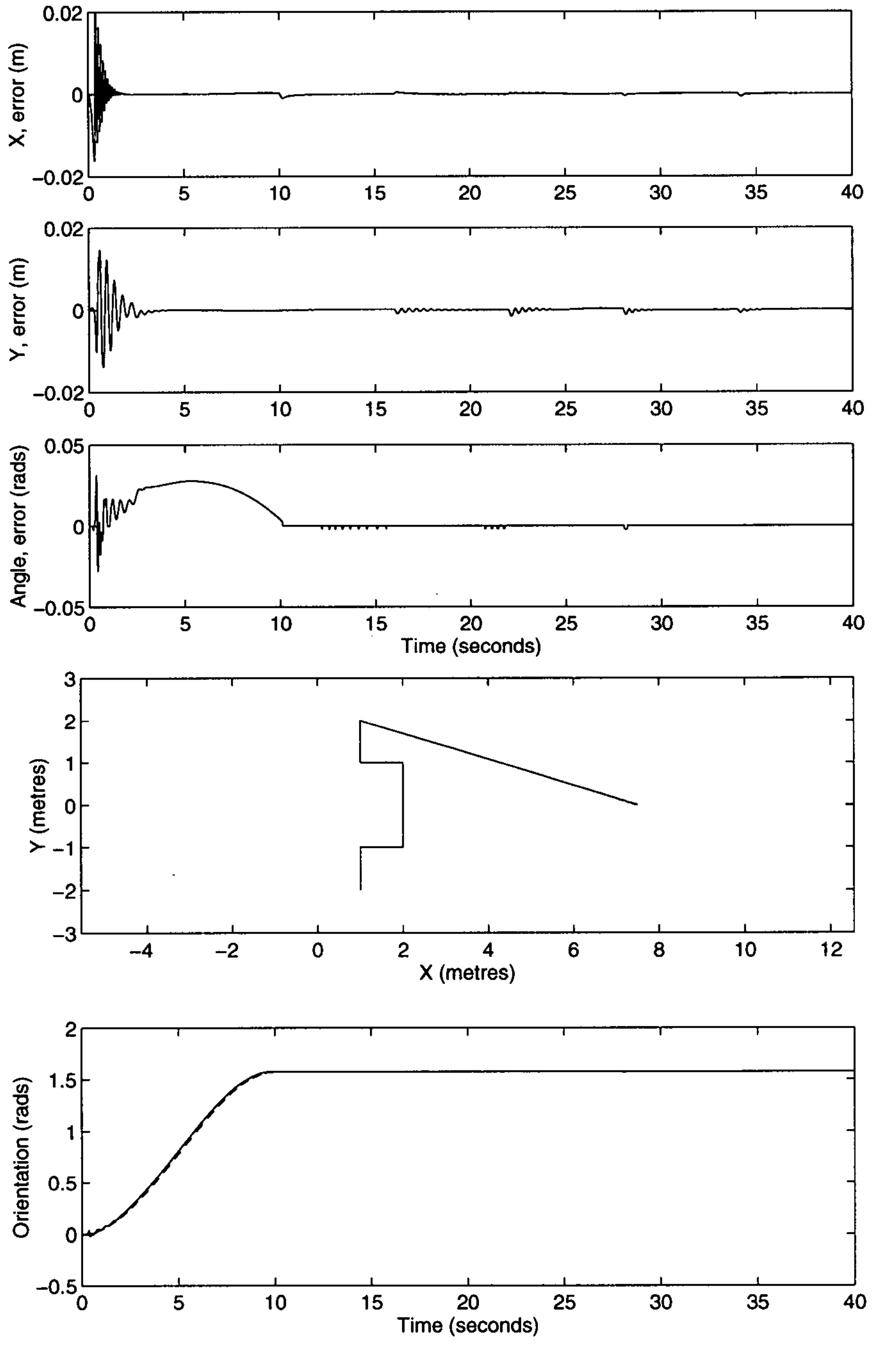


Figure 6.36: End effector trajectory tracking using 'Best Gains' RMDAC goal generator for a 20 degree of freedom planar manipulator.

D.o.F.	Reference Trajectory		
	$\sqrt{\overline{\lvert\mathbf{x}_e\rvert^2}}$	$\sqrt{\overline{\theta^2}}$	$\Delta T_N(sec.)$
5	5.331e-03	1.737e-02	0.0024
10	3.059e-03	6.014e-03	0.0022
15	2.404e-03	5.376e-03	0.0025
20	1.965e-03	5.196e-03	0.0027

Table 6.10: Tabulation of position and orientation RMS L_2 error and normalized execution time as a function of manipulator degrees of freedom.

torque values from exceeding $\mathbf{u}_{\text{sat}}$:

$$\mathbf{u}_{cj} = \begin{cases} \mathbf{J}_j^T(\mathbf{p}_{j-1}(t), \mathbf{p}_n(t))\mathbf{f}_d(t) & \text{if } -\mathbf{u}_{\text{sat}} \geq \mathbf{u}_{cj} \leq \mathbf{u}_{\text{sat}} \\ \mathbf{u}_{\text{sat}} & \text{if } \mathbf{u}_{cj} > \mathbf{u}_{\text{sat}} \\ -\mathbf{u}_{\text{sat}} & \text{if } \mathbf{u}_{cj} < -\mathbf{u}_{\text{sat}} \end{cases} \tag{6.216}$$

The results depicted in 6.37 show that end effector performance is virtually identical to figure 6.30. This is the first indication that the global goal generator's maintenance of the desired end effector trajectory tends to drive local disturbances (such as torque error due to saturation) into the Jacobian null space, a phenomenon discussed in the next chapter.

6.4 Summary

In this chapter, a global goal generator was specified and the implicit coordination of a multiagent manipulator controller demonstrated. The following points can be taken from this discussion:

- the RMDAC global goal generator models not a manipulator, but an ideal tracking process
- the generated global goal may be *broadcast* as a desired force-position couplet.
- a simulated manipulator can be controlled by N autonomous processes without a centralized manipulator model.
- each agent acting on the global goal, end effector control becomes linear in time on a single CPU, or constant in time if distributed.

By regulating an idealized trajectory tracking process rather than estimating and compensating for a specific manipulator architecture, RMDAC was shown to be an effective global goal generator free of any manipulator model.

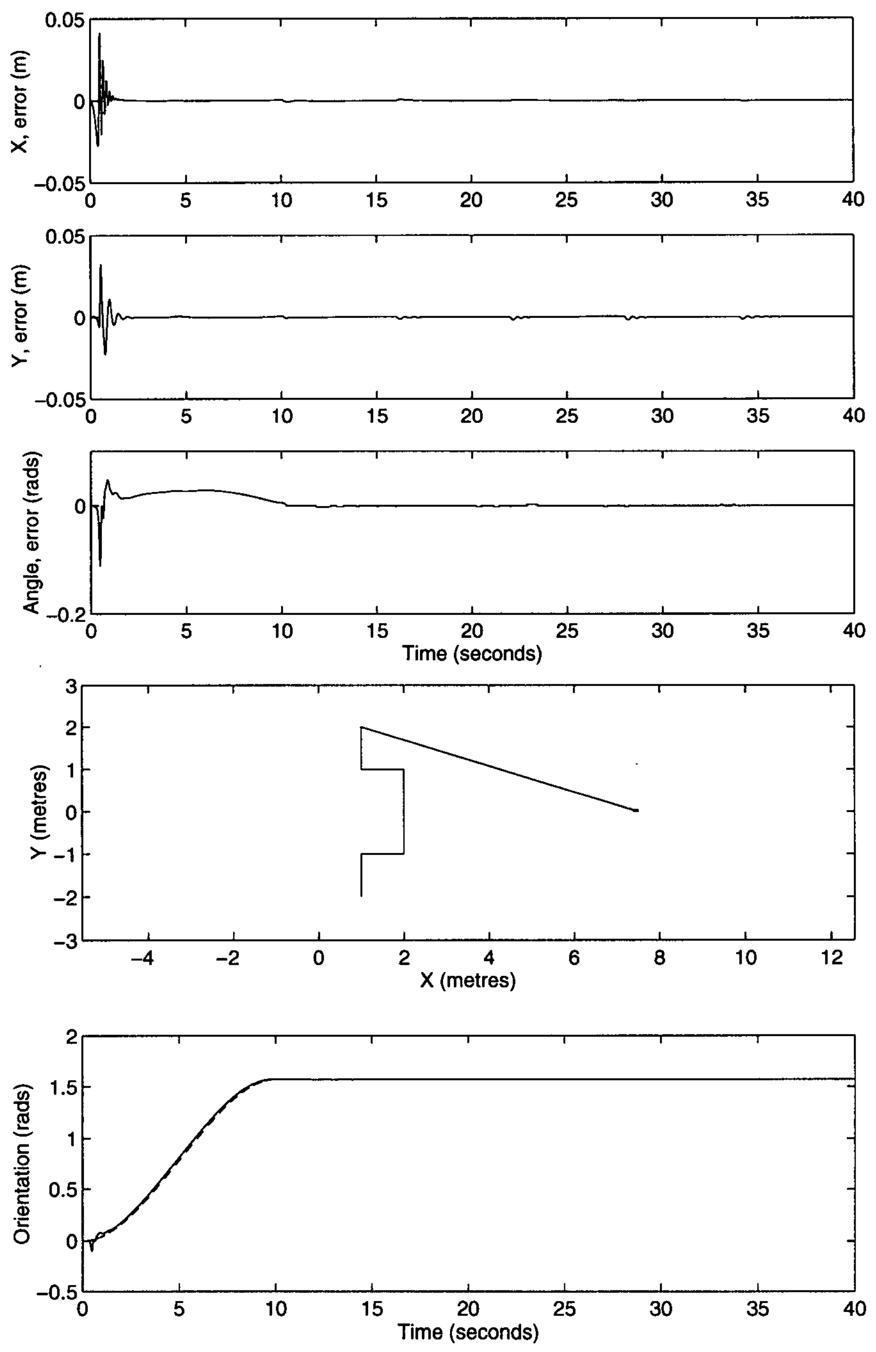


Figure 6.37: End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint force saturation ($\mathbf{u}_{\text{sat}} = 1000$ N-m).

The combination of this goal generation method and the global goal proxy described in the last chapter enables an arbitrary number of link processes to observe a single *broadcast* global goal *couplet*.

This simple computational architecture was successfully applied to the reference scenario through the multiprocess manipulator simulator and demonstrated, for the first time, the control of an end effector trajectory by a set of autonomous link processes devoid of any manipulator model. Given the definitions in chapter 4, this behaviour is clearly an example of an implicit coordination strategy.

The scalability and model independence of this strategy was explored in a performance comparison between manipulators of various configuration. Multiagent manipulator control was shown to be manipulator independent and, given one CPU per agent, constant in time.

The next chapter will examine the design and performance of additional goal systems within this architecture and discuss the arbitration of multiple conflicting goals.

Chapter 7

Multiple Goal Systems

7.1 Introduction

In the previous chapter, an adaptive end effector trajectory tracking controller, Configuration Control (or RMDAC), was reviewed and formulated into a global goal generation strategy. In this chapter additional global and local goals will be formulated and combinations of these goals demonstrated in multiagent manipulator control simulations.

While *multiple* goals in the form of primary and auxiliary controllers are not new to manipulator control, multiagent control adopts a decentralized architecture that changes how these goals might be generated and raises interesting questions about how agents should be coordinated to pursue multiple tasks. In particular, how can the convergence of multiple, simultaneous global goal systems be guaranteed *without* centralized coordination or allocation of agents to tasks? This chapter will address such questions in the context of redundant multiagent manipulator control and present a distributed global goal generation protocol. In particular, the impact of local goal systems on global stability and team coordination will be discussed and a general approach to multiple goal system design presented. The following specific local and global goal strategies will be presented in combination with trajectory tracking:

- global nonlinear obstacle avoidance.
- local nonlinear limit avoidance.
- local constant compliance centering
- local constant compliance retraction
- local variable compliance centering
- combined (local constant compliance centering and global nonlinear obstacle avoidance).

It will be shown in applying both compliant and adaptive goal systems, emergent coordination replaces traditional coordination strategies. Furthermore, it will be shown that through the combination of local and global goal systems, additional global behaviour emerges.

However, before venturing into multiple goal systems the next section will explore the fundamental interaction of primary and auxiliary goal systems and, based on established arguments, propose a Null Space Self Organization Theorem.

7.2 Auxiliary Behaviour and the Jacobian Null Space

In this section, a simple argument will demonstrate the inherent robustness of end effector control and, by analogy, any globally controlled multiagent system. The development demonstrates clearly that an underconstrained or redundant system under global control *automatically* inserts auxiliary behaviour into the null space of the global behaviour. Equally important, this argument suggests that *adaptive* global goal regulation is *equivalent* to explicit null space insertion.

Redundant manipulators are excellent vehicles to explore and demonstrate multiple global goal seeking. By explicitly inserting tasks into the null space, techniques such as RMRC and RMAC can guarantee the convergence of multiple tasks. Similarly, JTC can be applied to *augmented tasks* by ensuring the dimension of the auxiliary task, M_{aux} does not exceed the dimensions of the null space or $M_{\text{aux}} \leq N - M_{\text{primary}}$. While this is clearly a *necessary condition* that the number of constraints should not exceed the available degrees of freedom, it is not immediately clear why this simple rule is often *sufficient.* Surely, some additional care must be taken to ensure that, at the joint level, tasks do not conflict directly or $\mathsf{G}^j_{\text{aux}} \cap \mathsf{G}^j_{\text{primary}} = \emptyset$ for all $\mathsf{G}^j_{\text{aux}}$ in the manner of explicit null space insertion methods. An et al. [An, 1989] provide some insight into these issues.

An et al. intended to establish stability conditions for the convergence of kinematic learning controllers, those that estimate the end effector Jacobian through learning algorithms. The objective was to demonstrate that estimates of an inverse kinematic solution, $\hat{\mathbf{f}}^{-1}(\mathbf{x})$, converged to an actual inverse $\mathbf{f}^{-1}(\mathbf{x})$. However, by examining their arguments in a different light and assuming that $\hat{\mathbf{f}}^{-1}(\mathbf{x}) \equiv \mathbf{f}^{-1}(\mathbf{x})$, it becomes clear that end effector control *automatically* drives joint level perturbations into the Jacobian Null space. This property is sufficiently important to propose a simple theorem:

Theorem 2 (Null Space Self Organization) *Consider an N degree of freedom redundant manipulator in configuration* $\mathbf{q}^* \in \mathbf{R}^N$ *and at task space position,* $\mathbf{x}^* \in \mathbf{R}^M$*. If a task space control law is applied to the manipulator:*

$$\mathbf{q}_{k+1} = \mathbf{q}_k + (\mathbf{f}^{-1}(\mathbf{x}^*) - \mathbf{f}^{-1}(\mathbf{x}_k))$$

where the forward kinematic solution, $\mathbf{f}(\mathbf{q}) : \mathbf{R}^N \to \mathbf{R}^M$*, is invertible. Then a disturbance,* $\delta\mathbf{q}_k$*, is always reorganized into the Jacobian Null Space,* $\mathbf{N}(\mathbf{J})$*:*

$$\delta\mathbf{q}_{k+1} = \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*)\right] \delta\mathbf{q}_k \tag{7.217}$$

where $\mathbf{J}^{\dagger}$ *is the pseudoinverse of the task space Jacobian,* $\mathbf{J}$*.*

Proof

At time k, the manipulator is in position $\mathbf{q}^*$ with the end effector at $\mathbf{x}^*$. Now consider the effect of the small configuration space perturbation, $\delta\mathbf{q}_k$ on the manipulator configuration position becomes:

$$\mathbf{q}_k = \mathbf{q}^* + \delta\mathbf{q}_k \tag{7.218}$$

This deviation propagates to the end effector:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{f}(\mathbf{q}_k) \\ &= \mathbf{x}^* + \delta\mathbf{x}_k \\ &= \mathbf{x}^* + \mathbf{J}(\mathbf{q}^*)\delta\mathbf{q}_k \end{aligned}$$

Without correction, such deviations must corrupt the end effector trajectory.

Now suppose an end effector position correction strategy is adopted to correct such deviations, by commanding a new configuration, $\mathbf{q}_{k+1}$ through the simple rule [An, 1989]:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + (\mathbf{f}^{-1}(\mathbf{x}^*) - \mathbf{f}^{-1}(\mathbf{x}_k))$$

i.e. the difference between desired and current inverse solutions defines the joint error. Given:

$$\mathbf{f}^{-1}(\mathbf{x}_k) = \mathbf{f}^{-1}(\mathbf{x}^*) + \mathbf{J}^{\dagger}(\mathbf{x}^*)\delta\mathbf{x}_k$$

where $\mathbf{J}^{\dagger}$ is the pseudoinverse, $\mathbf{q}_k$ and $\delta\mathbf{x}_k$ may be substituted

$$\begin{aligned} \mathbf{q}_{k+1} &= \mathbf{q}_k + (\mathbf{q}^* - (\mathbf{f}^{-1}(\mathbf{x}^*) + \mathbf{J}^{\dagger}(\mathbf{x}^*)\delta\mathbf{x}_k)) \\ &= \mathbf{q}^* + \delta\mathbf{q}_k + \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*)\delta\mathbf{q}_k) \\ &= \mathbf{q}^* + (\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*))\delta\mathbf{q}_k \end{aligned}$$

Thus the correction to the manipulator configuration becomes:

$$\begin{aligned} \delta\mathbf{q}_{k+1} &= \mathbf{q}_{k+1} - \mathbf{q}^* \\ &= (\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*))\delta\mathbf{q}_k \end{aligned}$$

i.e. perturbations not already in the null space will be corrected into the null space. As k marches through time, $\delta \mathbf{q}_{k+1}$ will diminish if the absolute value of all the eigenvalues of $(\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*))$ are less than one [An, 1989]. Since $\mathbf{J}^{\dagger}$ is the pseudoinverse of $\mathbf{J}$, this is always true unless $\mathbf{q}^*$ lies on a kinematic singularity. Therefore, end effector control ensures that the manipulator reorganizes itself such that disturbances persist only in $\mathbf{N}(\mathbf{J})$.

Remarks

Now for a perfectly constrained manipulator in which the dimension of task and configuration spaces are identical, $\mathbf{J}^{\dagger} \equiv \mathbf{J}^{-1}$ and $\mathbf{q}_{k+1} = \mathbf{q}_k$, and the new position command is identical to the original setpoint. However, if the manipulator is *redundant* then the correction command $\delta \mathbf{q}_{k+1}$ is the projection of $\delta \mathbf{q}_k$ onto $\mathbf{N}(\mathbf{J})$ through the null space selection operator $(\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*))$. Thus perturbations initially in $\mathbf{R}(\mathbf{J})$ migrate into $\mathbf{N}(\mathbf{J})$ if the manipulator is redundant and the correction $\delta \mathbf{q}_{k+1}$ is exact. On reflection, this derivation is clearly reminiscent of the redundant manipulator control law in RMRC.

In summary the simple Null Space Self Organization theorem implies that perturbations in configuration space are driven into the null space by end effector control commands. Significantly, this property is not limited to manipulator kinematics but includes any multiagent system controlling some aggregate behaviour. Thus any multiagent system will exhibit the property that strict enforcement of a global goal ensures that local disturbances (or behaviours) are automatically driven into the null space.

This argument can be extended to the dynamic case in the following corollary to Theorem 2:

Corollary 2.1 *Consider an N degree of freedom redundant manipulator with configuration forces $\tau^* \in \mathbf{R}^N$ applying task space forces, $\mathbf{f}^* \in \mathbf{R}^M$. If a task space control law is applied to the manipulator:*

$$\mathbf{u}_{k+1} = \mathbf{u}_k + (\mathbf{J}^T(\mathbf{q}^*)\mathbf{f}^* - \mathbf{J}^T(\mathbf{q}^*)\mathbf{f}_k) \tag{7.219}$$

where the Jacobian Transpose $\mathbf{J}^T(\mathbf{q}) : \mathbf{R}^M \to \mathbf{R}^N$. Then a disturbance, $\delta\tau$, is always reorganized into $\mathbf{N}(\mathbf{J})$:

$$\delta\tau_{k+1} = \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{x}^*)\mathbf{J}(\mathbf{q}^*)\right] \delta\tau_k \tag{7.220}$$

where $\mathbf{J}^{\dagger}$ is the pseudoinverse of the task space Jacobian, $\mathbf{J}$.

Proof

Consider that actuators apply torque to each joint to provide a desired force at the end effector:

$$\mathbf{u}^* - [\mathbf{D}(\mathbf{q}^*)\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}^*)] = \mathbf{J}^T(\mathbf{q}^*)\mathbf{f}_d \tag{7.221}$$

with precise dynamic compensation in $\mathbf{f}_d = \mathbf{f}_{\text{comp}} + \mathbf{f}^*$:

$$\mathbf{u}^* = \mathbf{J}^T(\mathbf{q}^*)\mathbf{f}^* \tag{7.222}$$

the application of $\mathbf{u}^*$ produces the force $\mathbf{f}^*$ at the end effector. Now suppose at time k, the manipulator exerts joint forces $\mathbf{u}^*$ with the end effector forces at $\mathbf{f}^*$. Given a small perturbation $\delta\mathbf{u}_k$, the correction forces $\delta\mathbf{u}_{k+1}$ can be discovered through a derivation similar to Theorem 2. The task and configuration space forces at time k:

$$\begin{aligned}
\mathbf{u}_k &= \mathbf{u}^* + \delta\mathbf{u}_k \\
\mathbf{f}_k &= \mathbf{J}^{\dagger T}(\mathbf{x}^*)\mathbf{u}_k \\
&= \mathbf{f}^* + \delta\mathbf{f}_k \\
&= \mathbf{f}^* + \mathbf{J}^{\dagger T}(\mathbf{x}^*)\delta\mathbf{u}_k
\end{aligned}$$

Again without correction, torque perturbation deviations corrupt the end effector trajectory. Correcting the end effector force error at time $k+1$ through, $\mathbf{u}_{k+1}$ with the rule:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + (\mathbf{J}^T(\mathbf{q}^*)\mathbf{f}^* - \mathbf{J}^T(\mathbf{q}^*)\mathbf{f}_k) = \mathbf{u}_k - \mathbf{J}^T(\mathbf{x}^*)\delta\mathbf{f}_k \tag{7.223}$$

In effect, the projection of the error between desired and current end effector forces determines the configuration space force error. Given $\mathbf{u}_k$ and $\delta\mathbf{f}_k$:

$$\mathbf{u}_{k+1} = \mathbf{u}^* + (\mathbf{I} - \mathbf{J}^T(\mathbf{x}^*)\mathbf{J}^{\dagger T}(\mathbf{x}^*))\delta\mathbf{u}_k \tag{7.224}$$

Thus the applied correction torque becomes:

$$\delta\mathbf{u}_{k+1} = (\mathbf{I} - \mathbf{J}^T(\mathbf{q}^*)\mathbf{J}^{\dagger T}(\mathbf{x}^*))\delta\mathbf{u}_k$$

and configuration space force corrections are projected into a redundant manipulator's Jacobian null space through the $(\mathbf{I} - \mathbf{J}^T(\mathbf{q}^*)\mathbf{J}^{\dagger T}(\mathbf{x}^*))$ operator.

As before, fully constrained manipulators (i.e. $\mathbf{J}^\dagger \equiv \mathbf{J}^{-1}$) command the original joint force setpoint. Since joint forces balancing the end effector load reside, by definition, in $R(\mathbf{J}^T)$, forces that produce *no end effector force* must reside solely in $\mathbf{N}(\mathbf{J})$. Having established a Null Space Self Organization Theorem, the combined performance of multiple goal systems can now be discussed.

7.3 Auxiliary Global Goal Systems

To this point, global goals have been generated by a external, dedicated global goal processes (e.g. a user defined trajectory generator). However, it is not uncommon for agents, themselves, to both generate and pursue one or more global goal requests. Indeed many multiagent systems use agent based global goal broadcasts (e.g. [Mataric, 1994, Parker, 1992a]) as a foundation of interagent communication. This section will introduce and demonstrate a protocol that enables individual link agents to both *assert* and pursue multiple global goals within the agent team.

7.3.1 Obstacle Avoidance

Avoiding collisions with workspace obstacles is perhaps the most desirable auxiliary task in manipulation. The object is to avoid collisions in the manipulator's workspace while tracking a desired end effector trajectory. Despite some similarity with path planning, obstacle avoidance schemes often do not rely on detailed world models to ensure that the manipulator will not collide with objects in the work space. In this way, *auxiliary* obstacle avoidance task supports the *primary* trajectory tracking global goal.

In RMPC position control, obstacle avoidance strategies require the geometric resolution of both manipulator and obstacle boundary constraints. RMRC and RMAC are less complex methods, inserting obstacle avoidance controllers either directly or indirectly into the null space of the end effector Jacobian through the Jacobian insertion operator, $(\mathbf{I}-\mathbf{J}^{\dagger}\mathbf{J})$, or augmentation of the task space, $\mathbf{x}$, (and the Jacobian) with an avoidance function. Controllers include simple proportional [Slotine, 1988] and adaptive [Colbaugh, 1989] range controllers, nonlinear and optimal model based controllers (e.g. [Nakamura, 1984, Sung, 1996]). Similar methods have been applied to Jacobian Transpose control by applying virtual avoidance forces to the manipulator.

The next subsections will briefly overview two relevant JTC techniques and present a multiagent hybrid of these methods.

The Operational Space Formulation (OSF)

In the operational space formulation, Khatib [Khatib, 1985] described obstacles of known geometry with an APF or artificial potential field model, Φ. By composing 'virtual' repulsive forces derived from the negative gradient of Φ and applying them to "points subject to potential" or PSPs, the Jacobian transpose of the PSP can be used to propagate these forces to the manipulator and avoid the obstacle. The strategy was active only within a specific PSP-obstacle surface triggering range. Specifically, repulsive forces (again per unit mass),$\mathbf{f}^*_{(j,i)}$ were derived from the jth objects potential field model, Φ_j, applied to the ith links PSP:

$$\mathbf{f}^*_{(j,i)} = -\nabla\Phi_j(\mathbf{p}_{i_{\text{psp}}}) \tag{7.225}$$

a technique since emulated by many including Arkin [Arkin, 1987]. The potential field model Khatib used was based on an inverse law:

$$\Phi(\rho) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{7.226}$$

where η is a gain, ρ describes the obstacle in link coordinates and ρ_0 is a threshold distance. Applying the gradient operator, the repulsive force becomes an inverse square law:

$$\mathbf{f}(\rho) = \begin{cases} \eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2}\frac{\delta\rho}{\delta\mathbf{X}} & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{7.227}$$

As defined in [Khatib, 1985], ρ is a function of obstacle geometry, requiring a recognition and world modelling system to identify and store obstacles. With the jth obstacle modelled by an APF, Φ_j, the repulsive force at ith PSP is computed through simple superposition:

$$\mathbf{f}^*_i = \sum_j \mathbf{f}^*_{(j,i)} = \sum_i -\nabla\Phi_j \tag{7.228}$$

Thus the combined expression control law for trajectory tracking and real time obstacle avoidance became:

$$\tau = \mathbf{J}^T\mathbf{f}_d + \sum_i \mathbf{J}_i^T\mathbf{M}(\mathbf{x})\mathbf{f}_{(j,i)} \tag{7.229}$$

where $\mathbf{f}_d$ is as in equation(E.414) and $\mathbf{J}_i$ is the Jacobian of the ith PSP.

Discussion

Despite the effectiveness of this technique, Khatib anticipated problems with OSF. Since $\mathbf{f}_d$, is a cartesian PD controller and $\mathbf{f}_{(j,i)}$ are nonlinear forces, configurations are possible in which [Khatib, 1985]:

> "...local minima can occur in the resultant potential field. This can lead to a stable positioning of the robot before reaching its goal."

Khatib recognized that a manipulator, trapped by obstacle repulsive fields, might not be able to follow the desired end effector trajectory. Khatib believed that local procedures might be able to extract a robot from this predicament, in effect resolving constraints through some decision making process.

Though powerful, APF models are not practical for multiagent obstacle avoidance. The reliance on complete, global view of both the environment and the manipulator is both computationally burdensome, difficult to decentralize, and, with current sensing, prohibitively expensive to implement in real time.

An alternative to the global view is to combine local sensing with a simpler obstacle independent repulsion strategy, removing both the cost of sensing and recognition and the centralized world model. This strategy, adopted by Colbaugh in the implementation of an RMDAC based obstacle avoidance system, is briefly reviewed in the next section.

Obstacle Avoidance with Configuration Control

Colbaugh attacks the obstacle avoidance problem by *augmenting* the task space with an obstacle avoidance constraint and enforcing task space forces with a model reference decentralized adaptive controller. In redundant manipulator Configuration Control (for clarity, here referred to as RMDAC), the end effector position vector, $\mathbf{x}$, is *augmented* by additional kinematic constraints such as obstacle and joint limit boundaries:

$$\mathbf{x}_{\mathrm{a}} = \begin{bmatrix} \mathbf{x}_{ce} \\ \mathbf{x}_{c} \end{bmatrix} \tag{7.230}$$

Thus the end effector Jacobian, $\mathbf{J}_{ee}$, is also augmented by a constraint Jacobian, $\mathbf{J}_c$, through which constraint forces, $\mathbf{f}_c$, are applied to a manipulator's redundant degrees of freedom. Specifically:

$$\tau = \mathbf{J}_{\mathrm{a}}^T \mathbf{f}_a = \begin{bmatrix} \mathbf{J}_{ce} \\ \mathbf{J}_c \end{bmatrix}^T \begin{bmatrix} \mathbf{f}_{d_{ee}}(\mathbf{x}_{ce}) \\ \mathbf{f}_{dc}(\mathbf{x}_c) \end{bmatrix} \tag{7.231}$$

An RMDAC controller is then applied to the augmented state vector to produce an augmented force vector. As before these forces are transformed into joint torques through the (augmented) Jacobian transpose. Note that the augmentation process and the summation process in OSF are mathematically identical.

In RMDAC obstacle avoidance $\mathbf{f}_{dc}(\mathbf{x}_c)$ was based on a computed minimum distance from the obstacle surface and range sensor data. In [Colbaugh, 1989], Colbaugh defines a *critical point*,$(\mathbf{x}_c)_{ij}$ as the point on link i closest to the surface of obstacle j in local link coordinates. If the position, $(\mathbf{x}_o)_j$, of the jth obstacle is known and a clearance (or triggering) radius $(r_o)_j$ is defined, then the magnitude of the minimum distance is:

$$[d_c(\mathbf{q})]_{ij} = ||(\mathbf{x}_c)_{ij} - (\mathbf{x}_o)_j||_2 \tag{7.232}$$

and an avoidance constraint,$g_{ij}(\mathbf{q})$ can be formulated:

$$g_{ij}(\mathbf{q}) \equiv [d_c(\mathbf{q})]_{ij} - (r_o)_j \geq 0 \tag{7.233}$$

where

$$[\mathbf{x}_{ce} \;\; \dot{\mathbf{x}}_{ce}]^T = \begin{cases} [0 \;\; 0]^T & \text{if } g_{ij}(\mathbf{q}) \geq 0 \\ [-g_{ij} \;\; -\dot{g}_{ij}]^T & \text{if } g_{ij}(\mathbf{q}) < 0 \end{cases} \tag{7.234}$$

Now $\mathbf{f}_{dc}(\mathbf{q})$ is generated through the *feedback portion* of an RMDAC controller to $\mathbf{x}_{ce}$:

$$\mathbf{f}_{dc}(\mathbf{q}) = d_c(t) + K_{p_c}(t)\mathbf{x}_{ce} + K_{v_c}(t)\dot{\mathbf{x}}_{ce} \tag{7.235}$$

where $d_c(t)$,$K_{p_c}(t)$ and $K_{v_c}(t)$ are defined as in equation (6.206).

Discussion

Since there are no APF obstacle models, this technique is less compute intensive than OSF and enables direct sensing of $[d_c(\mathbf{q})]_{ij}$. Furthermore, the incorporation of $\mathbf{x}_c$ into the task state vector means that both $\mathbf{x}_{ee}$ and $\mathbf{x}_c$ are enforced by an adaptive controller. To be achieved, therefore, both tasks must occupy complementary regions of configuration space, $\mathbf{q}$ or:

$$R(\mathbf{J}_c^T) \cap R(\mathbf{J}_{ee}^T) = \emptyset \tag{7.236}$$

This is merely a corollary on the fact that, given r the redundant degrees of freedom, a maximum of r constraints may augment the task vector or:

$$\bigcap_{i=1}^{r} R(\mathbf{J}_{ci}^T) = \emptyset \tag{7.237}$$

If both tasks were compliant and if the above condition were not true, neither task would be achieved and an equilibrium would be established between the two tasks. Under adaptive control, however, there is no such compliance. Indeed, competition between adaptive controllers can lead to numerical instability. In other words, adaptive augmented state regulation ensures the convergence of *mutually exclusive* tasks, but cannot resolve *conflicting* tasks.

Obstacle Avoidance and Arbitration

Together OSF and RMDAC obstacle avoidance methods provide two important lessons on arbitration.

Conflicts between strictly enforced goals can only be resolved through explicit coordination. If multiple goals are maintained by similar adaptive schemes, any conflict between goal systems (e.g. $M_{\text{avoid}} > N - M_{\text{track}}$ avoidance tasks) can lead to competitive adaptation and, ultimately, instability. Clearly, agents that use *adaptive* augmented state systems must use some form of explicit arbitration to either enable, disable, or modify goal systems during goal conflict. Indeed, if the goal systems are global, a centralized process may be necessary to orchestrate a global arbitration strategy – an explicit coordination system.

Conflicts between compliant goals can be resolved through dynamic equilibrium. By using a compliant, albeit *nonlinear*, repulsion system and a compliant *linear* end effector force generator, OSF ensures collision avoidance and, when tasks conflict, stability. In short, OSF relies on environmental state, manipulator dynamics, and controller interaction to implement a goal coordination scheme – an emergent coordination system.

Next, a multiagent technique will be introduced that exploits Jacobian Transpose decentralization and a compromise between the rigid goal enforcement of RMDAC and the compliance of OSF to demonstrate decentralized obstacle avoidance.

7.3.2 Multiagent Obstacle Avoidance

Since both Khatib's and Colbaugh's obstacle avoidance methods rely on Jacobian transpose control, it is clear that these, too, may be decentralized just as the end effector trajectory tracking task. Thus the structure of the global goal proxy is not specific to end effector trajectory tracking, but is applicable to any global goal system. Hence the variable $\mathbf{p}_{\mathsf{app}}$ refers to a generic point of *application* for some desired force.

For obstacle avoidance, the desired applied force, $\mathbf{f}_d = \mathbf{f}_{\mathsf{psh}}$, a repulsive force away from the obstacle surface while the point of application is the 'point subject to hazard' (*psh*)[1] $\mathbf{p}_{\mathsf{app}} = \mathbf{p}_{\mathsf{psh}}$. Together these form the obstacle avoidance global goal couplet:

$$\mathcal{G}_{\mathsf{psh}} = \langle \mathbf{p}_{\mathsf{psh}}, \mathbf{f}_{\mathsf{psh}} \rangle \tag{7.238}$$

where $\mathbf{p}_{\mathsf{app}} = \mathbf{p}_{\mathsf{psh}}$ and $\mathbf{f}_d = \mathbf{f}_{\mathsf{psh}}$.

The structure and computation of the agent global proxy (4.128) remains unchanged, regardless of the source of $\mathcal{G}_{\mathsf{psh}}$. However, for the ith link, avoidance forces can only be provided by *superior* links $0 \leq i$ – those constituting the forward solution of the *psh*. Specifically:

$$\begin{aligned} \mathbf{p}_{\mathsf{psh}} &= f_{\mathsf{psh}}(\mathbf{q}) && (7.239) \\ \tau_{\mathsf{psh}} &= \left[\frac{\partial f_{\mathsf{psh}}(\mathbf{q})}{\partial \mathbf{q}} \right]^{\mathrm{T}} \mathbf{f}_{\mathsf{psh}} && (7.240) \end{aligned}$$

where $\mathbf{q}$ is the vector of generalized coordinates. Now if $\mathbf{p}_{\mathsf{psh}}$ lies on link i then the forward solution $f_{\mathsf{psh}}(\mathbf{q})$ is a function of links $j : 0 \leq j \leq i$. This means that only 'superior' links, $j \leq i$, participate in the obstacle avoidance effort making obstacle avoidance a *group* correlated goal[2].

[1] With nomenclature reminiscent of Khatib's PSPs, 'points subject to hazard' are not fixed a priori and, in fact, are virtually identical to Colbaugh's critical points

[2] However, if *both* end effector and base are rigidly constrained, theoretically all agents can drive the *psh* away from the obstacle based on the *same* global goal expression.

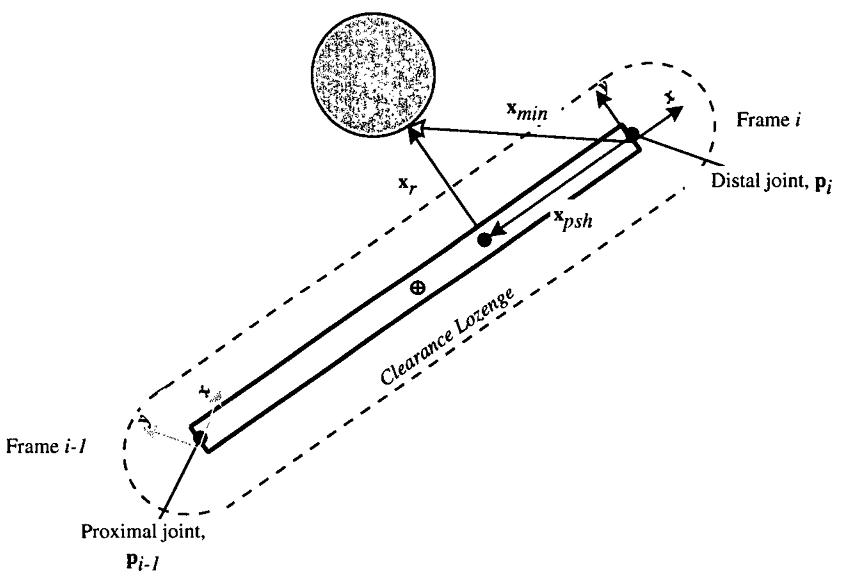


Figure 7.38: A simple link agent controller model.

This section will describe the design of an agent behaviour controller that, when triggered by a sensed obstacle boundary, *asserts* or broadcasts a global obstacle avoidance goal to the agent team. This *resolved motion obstacle avoidance* behaviour controller (RMOA) is formed of two fundamental components. The first monitors the local region with range sensors and formulates a response to an obstacle that intrudes a *clearance lozenge* surrounding the link. The second component polls the incoming goal stream for *multiple* global goals of the form (7.238) and develops a response based upon these requests.

Responding to Obstacles

Just as Colbaugh's obstacle avoidance system identified a critical point on each link, the sensor model in the multiagent obstacle avoidance behaviour controller determines a point subject to hazard or *psh* through a cylindrical sensor array. This array is aligned with the link axis and measures range to the nearest obstacle surface. Employing neither recognition nor surface mapping techniques, RMOA's sensor model simply returns a vector to the nearest obstructing surface, $\mathbf{x}_r$, expressed in the *link's* coordinate system.

The RMOA sensor can be modelled as a linear range sensor array or its equivalent described parametrically in link coordinates:

$$\mathbf{x}_{\text{arr}}(\alpha) \quad = \quad \mathbf{x}_1 + \alpha(\mathbf{x}_2 - \mathbf{x}_1) \quad 0 \leq \alpha \leq 1 \tag{7.241}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the proximal and distal coordinates of the link in link coordinates. With such an array, a visible surface can be interpreted as a function of the array length.

$$\mathcal{S}: \quad \mathbf{x}_{\text{surf}} = f_{\text{surf}}(\mathbf{x}_{\text{arr}}) \tag{7.242}$$

Defining $\min(\mathbf{x}_{\text{surf}})$ as the value of f_{surf} at which the following is true:

$$\frac{\partial f_{\text{surf}}}{\partial \mathbf{x}_{\text{arr}}} = 0 \tag{7.243}$$

$$\frac{\partial^2 f_{\text{surf}}}{\partial \mathbf{x}_{\text{arr}}^2} > 0 \tag{7.244}$$

a sensor array may be designed to report only the nearest hazard or $\min(\mathbf{x}_{\text{surf}})$:

$$\mathbf{x}_{\text{min}} = \min(\mathbf{x}_{\text{surf}}) \tag{7.245}$$

$$\mathbf{x}_{\text{psh}} = \mathbf{x}_r \cdot \mathbf{x}_i \tag{7.246}$$

where $\mathbf{x}_i$ is the basis vector of the ith frame in the x direction. Thus the minimum range from the *psh* to the surface, $\mathbf{x}_r$, is simply $\mathbf{x}_r = \mathbf{x}_{\text{min}} - \mathbf{x}_{\text{psh}}$.

With a linear array of range sensors distributed over each link, as in Colbaugh's implementation, each link may be enclosed by an artificial potential field, reminiscent of obstacle APFs used in OSF and depicted in 7.38. The obstacle avoidance strategy enforces a clearance range forming a nonlinear repulsion 'lozenge' about the link within which an APF, similar to equation (7.226), is activated. Within the lozenge, a repulsive force is determined based on the *magnitude* of the minimum distance or $\rho = |\mathbf{x}_r|$, from the link surface to the obstacle surface (see figure 7.39):

$$|\mathbf{f}_r(\mathbf{x}_r)| = \eta \left(\frac{1}{|\mathbf{x}_r|} - \frac{1}{c} \right) \frac{1}{|\mathbf{x}_r|^2} \tag{7.247}$$

The force acts in the direction of the range, $\frac{\mathbf{x}_r}{|\mathbf{x}_r|}$. Triggering the behaviour at a clearance range of c, the repulsive force becomes:

$$\mathbf{f}_r(\mathbf{x}_r) = \begin{cases} \frac{\mathbf{K}_p}{|\mathbf{x}_r|^3} \left(\frac{1}{|\mathbf{x}_r|} - \frac{1}{c} \right) \mathbf{x}_r & \text{if } |\mathbf{x}_r| \leq c \\ 0 & \text{if } |\mathbf{x}_r| > c \end{cases} \tag{7.248}$$

Through this RMDAC/nonlinear repulsion hybrid a margin of stability is gained during periods of overconstraint. The advantage the potential field approach has over the RMDAC obstacle avoidance goal is that the former permits some deflection and, therefore, some room for direct conflict between constraints. The disadvantage is that these same conflicts may result in instability if the obstacle intrusion becomes large.

Once computed, the $\mathbf{f}_r$ and $\mathbf{x}_{\text{psh}}$ are transformed into the global coordinates $\mathbf{f}_{\text{psh}}$ and $\mathbf{p}_{\text{psh}}$ respectively and *asserted* to the global goal process for retransmission to other agents.

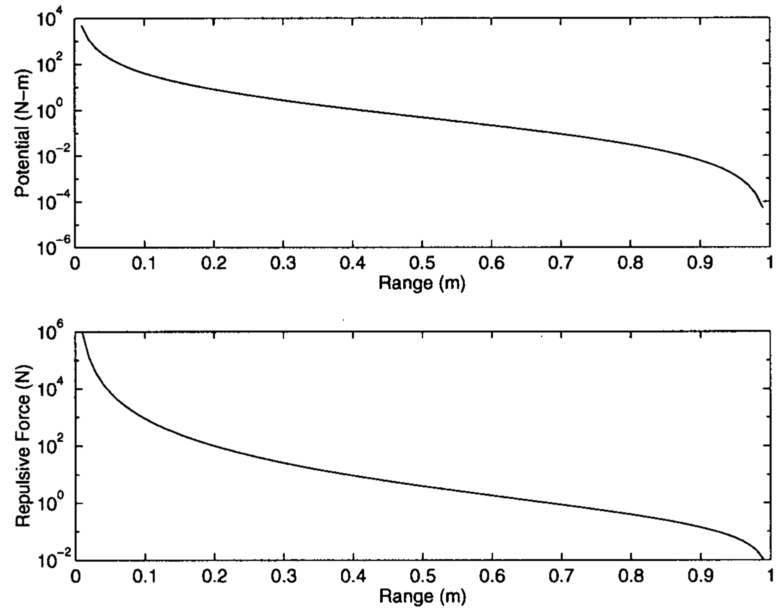


Figure 7.39: Semilog plots of the potential field (top) and repulsive force (bottom) as a function of normalized range.

Global Goal Assertion Rebroadcast

As described in the previous chapter, the global goal process GP maintains all the global goal processes. As described above, RMOA also enables each agent to generate global goal assertions. However, agents lack the necessary information to broadcast such assertions to the agent team (such information constitutes a local manipulator model – a feature worth avoiding if possible). Ideally such broadcasts might occur over a common communications bus to all agents possessing the RMOA controller. To mimic such a bus system, the global goal process receives global goal assertions from RMOA controllers for rebroadcast to *superior links* with RMOA controllers.

Thus RMOA controllers must transmit data (through a `MeasurandVector`) and a receive data (through a `SensorVector`) from the goal process, both of which are members of the agent's `IOStream`. For a given receiving agent, the global goal process determines that the agent is superior to (i.e. closer to the manipulator base in the kinematic chain than) the asserting agent and inserts the global goal couplet into the agents receive message queue. Unfortunately, this message filtering process is a clearly not a simple broadcast. However, a 'one way' or unidirectional bus structure from end effector to base simplifies the communication model, retaining a relatively model-free communications structure.

Arbitrating Between Multiple Goals

Having completed the assertion phase of the obstacle avoidance task, RMOA retrieves the global goal message queue and applies the global goal proxy expression to every received RMOA couplet to establish response forces for each avoidance request. For the jth agent:

$$\mathbf{u}_{\mathsf{track}} = \mathbf{J}_j^T(\mathbf{p}_n, \mathbf{x}_{j-1})\mathbf{f}_d \tag{7.249}$$

$$\mathbf{u}_{\mathsf{avoid}} = \sum_{k=1}^{N-j} \mathbf{J}_j^T(\mathbf{p}_{\mathsf{psh_k}}, \mathbf{x}_{j-1})\mathbf{f}_{\mathsf{psh_k}} \tag{7.250}$$

where $\mathbf{J}_j^T(\cdot,\cdot)$ is the global goal proxy and N is the number of agents. The response $\mathbf{u}_{\mathsf{avoid}}$ is passed to the arbitrator, A. Just as in previous obstacle avoidance methods arbitration can be simplified to simple linear combination. Recalling equation (4.92), for the jth agent:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1] \begin{bmatrix} \mathbf{u}_{\mathsf{track}} \\ \mathbf{u}_{\mathsf{avoid}} \end{bmatrix} \tag{7.251}$$

where the $\mathbf{k}_j$ is the arbitration vector. Significantly, the arbitration strategy of linear combination is *not* the product of some design process but from Newtons Second Law.

Since $\mathbf{u}_{\mathsf{avoid}}$ is a triggered behaviour controller, the arbitration vector could be rewritten as:

$$\mathbf{k}_j = [k_{\mathsf{track}} \;\; k_{\mathsf{avoid}}] \tag{7.252}$$

where

$$k_{\mathsf{track}} = 1.0 \quad k_{\mathsf{avoid}} = \begin{cases} 1 & \text{if } |\mathbf{x}_r| \leq c \\ 0 & \text{if } |\mathbf{x}_r| > c \end{cases} \tag{7.253}$$

Since this triggering occurs within $\mathbf{H}_{j\mathsf{avoid}}$ and not within A_j the goal system is self triggering.

Despite decentralization, goal assertion, and rebroadcast, the expression describing RMOA over the manipulator is identical to both equations (7.230) and (7.229) as would any system based on Jacobian Transpose obstacle avoidance. The benefit of this system is that the computation is distributed, each agent can assert unique (even multiple) avoidance strategies based on different sensing systems (e.g. thermal, ultrasonic, etc.), and, perhaps most importantly, the failure of a single RMOA controller threatens only a single link – and not the entire manipulator. Furthermore, by recognizing that task conflict can be reconciled through compliant and adaptive controllers a mechanism of decentralized arbitration has been identified.

Though the process may seem complex in comparison to RMDAC or OSF, closer examination of these centralized schemes reveals similar (if not more daunting) complexity in collecting and polling N (possibly

Strategy	e_{rms}	θ_{rms}
Avoidance	3.481e-03	5.989e-03

Table 7.11: RMS error for a combined tracking and obstacle avoidance strategy.

dissimilar) sensor arrays and computing the responses for N links in real time. In the multiagent structure discussed here, the global goal process transmits (at most) $N-1$ global goal couplets to each agent and receives at most $N-1$ global goal couplet assertions, each only 96 bytes long. Agents receive and transmit kinematic packets (144 bytes each) as well as `IOStream` and goal assertions (receive: $96(N-j)$ bytes, transmit: 96 bytes). It is hard to imagine a *centralized* data collection, analysis, and control system with lighter communication loads than decentralized RMOA.

7.3.3 Results

An experiment can now be constructed to explore the performance of this distributed, multiagent obstacle avoidance protocol. In the following test, an obstacle, a sphere 0.25m in diameter, is placed at a known interference location of the manipulator's motion though not obstructing the reference trajectory. The resulting performance, depicted in figure 7.40, demonstrates that the reference manipulator under multiagent control can successfully negotiate an interfering obstacle while tracking an end effector trajectory.

During pure trajectory tracking, the manipulator exhibits typical jumbled, unconstrained 'free' motion in the Jacobian null space. However, once an obstacle intrudes a link's clearance lozenge, links appear to 'rebound' from and 'slide' along a surface enveloping the obstacle. Despite these intrusions, the end effector exhibits good trajectory tracking performance as described in figure 7.41 and condensed into RMS error in table 7.11.

Examination of figures 7.42 to 7.46 reveals the assertion and propagation of avoidance torques from source to base agents. Exemplified in figure 7.46, an avoidance torque output is generated by link 9 in response to an obstacle encroaching its clearance lozenge. Link 9 then asserts an avoidance goal to the superior links (links 1 through 8). Thus link 9's avoidance torque waveforms are echoed and magnified as each superior agent formulates a local response to link 9's global goal assertion through its global goal proxy operator. Of course, each agent acts on global avoidance assertions *and* asserts global avoidance goals as well if an obstacle intrudes the local lozenge. Therefore, the jth link ($j < 9$) response is often the combination of global goal assertions from links $k : j < k \leq 9$. Not surprisingly, avoidance torque waveforms become increasingly

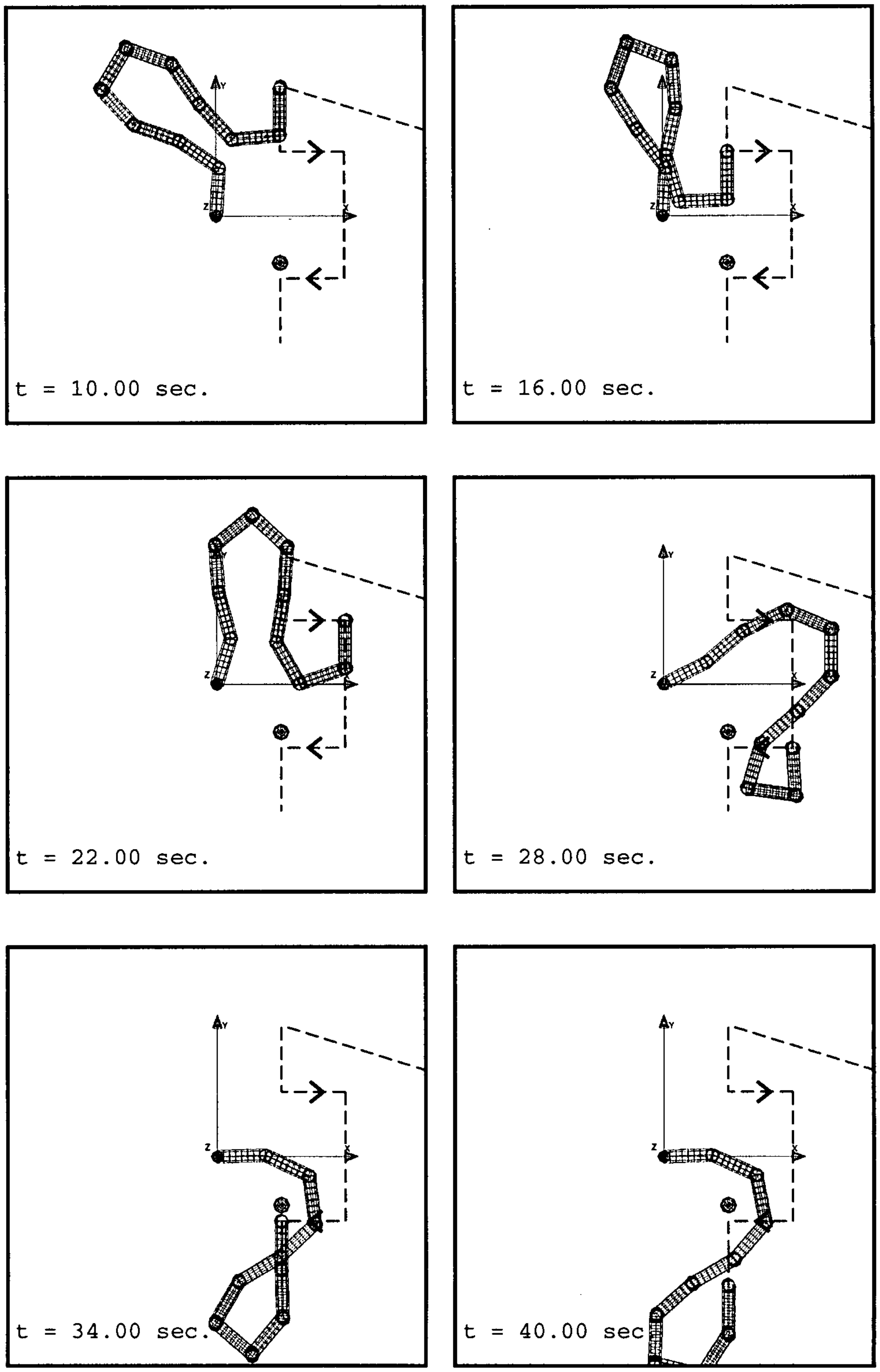


Figure 7.40: The reference manipulator avoids a small stationary sphere while tracking the reference trajectory. The sphere is 0.250m in diameter at $\mathbf{x}_{\text{obs}} = [1.00 \quad -0.750]^T$. The RMOA controller uses a clearance of 0.75 m and gain of $\eta = 100.0$.

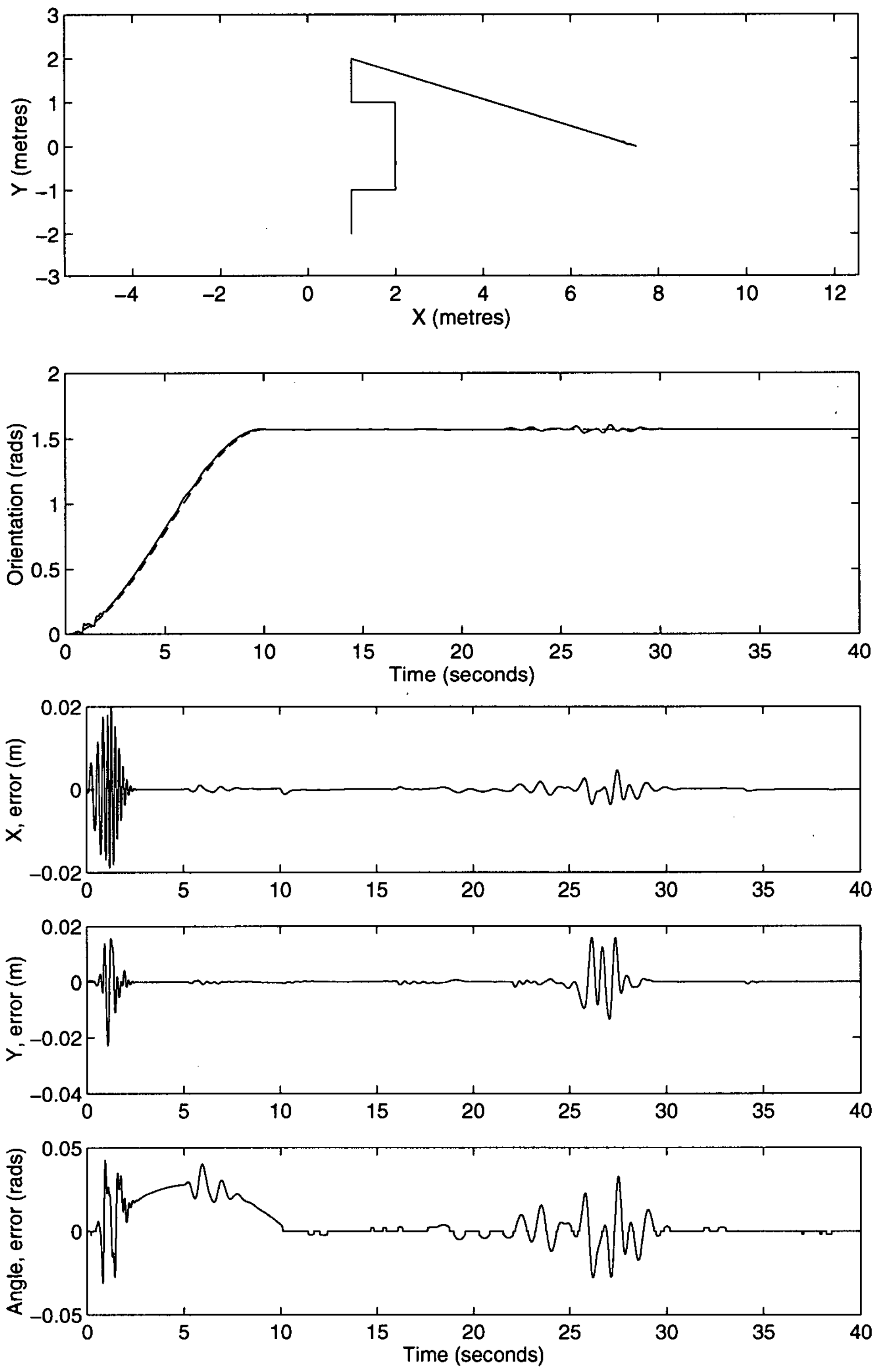


Figure 7.41: End effector trajectory tracking performance of the reference manipulator engaging in multiagent trajectory tracking and obstacle avoidance.

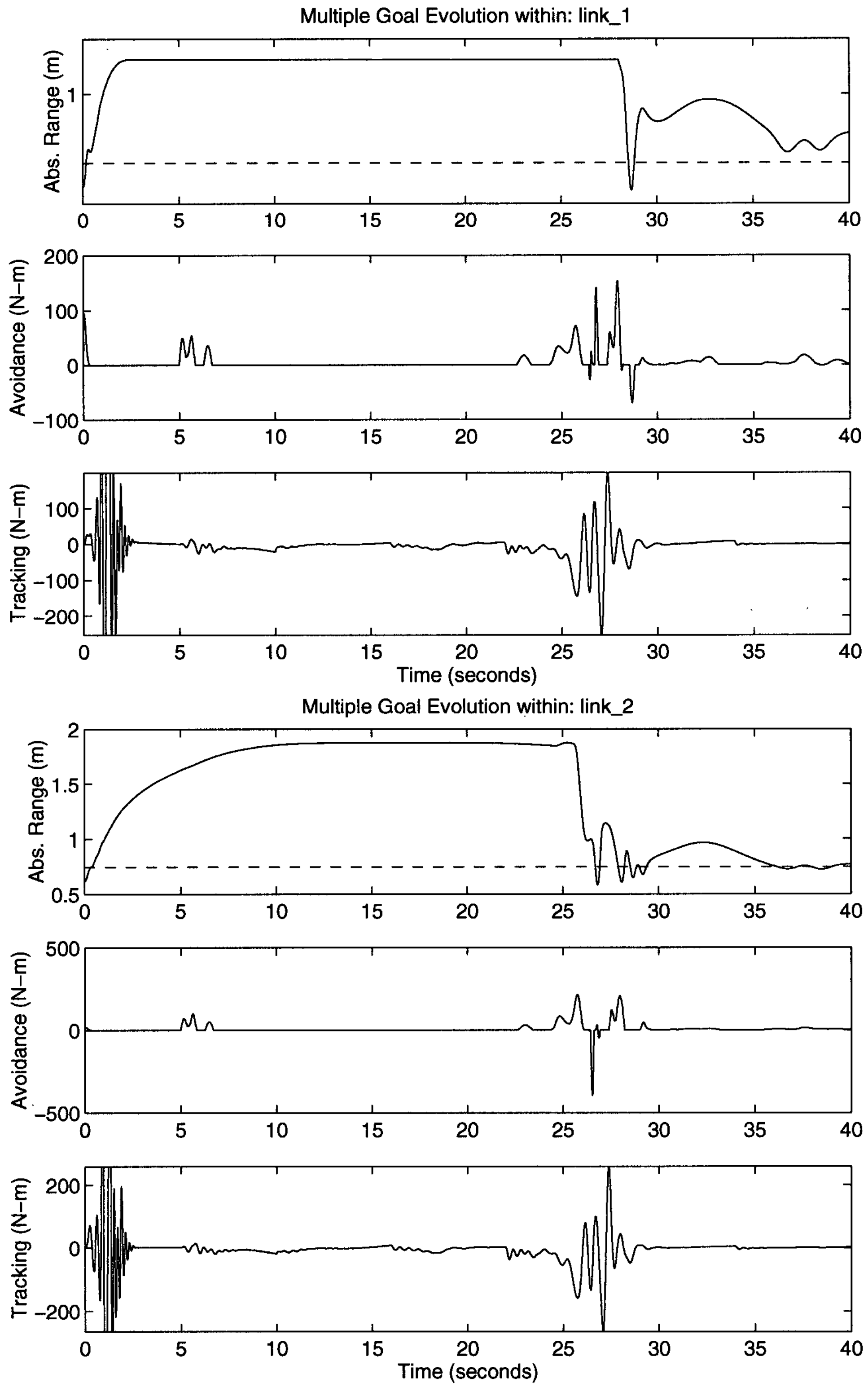


Figure 7.42: Evolution of the Global Goals within links 1(top) and 2 (bottom).

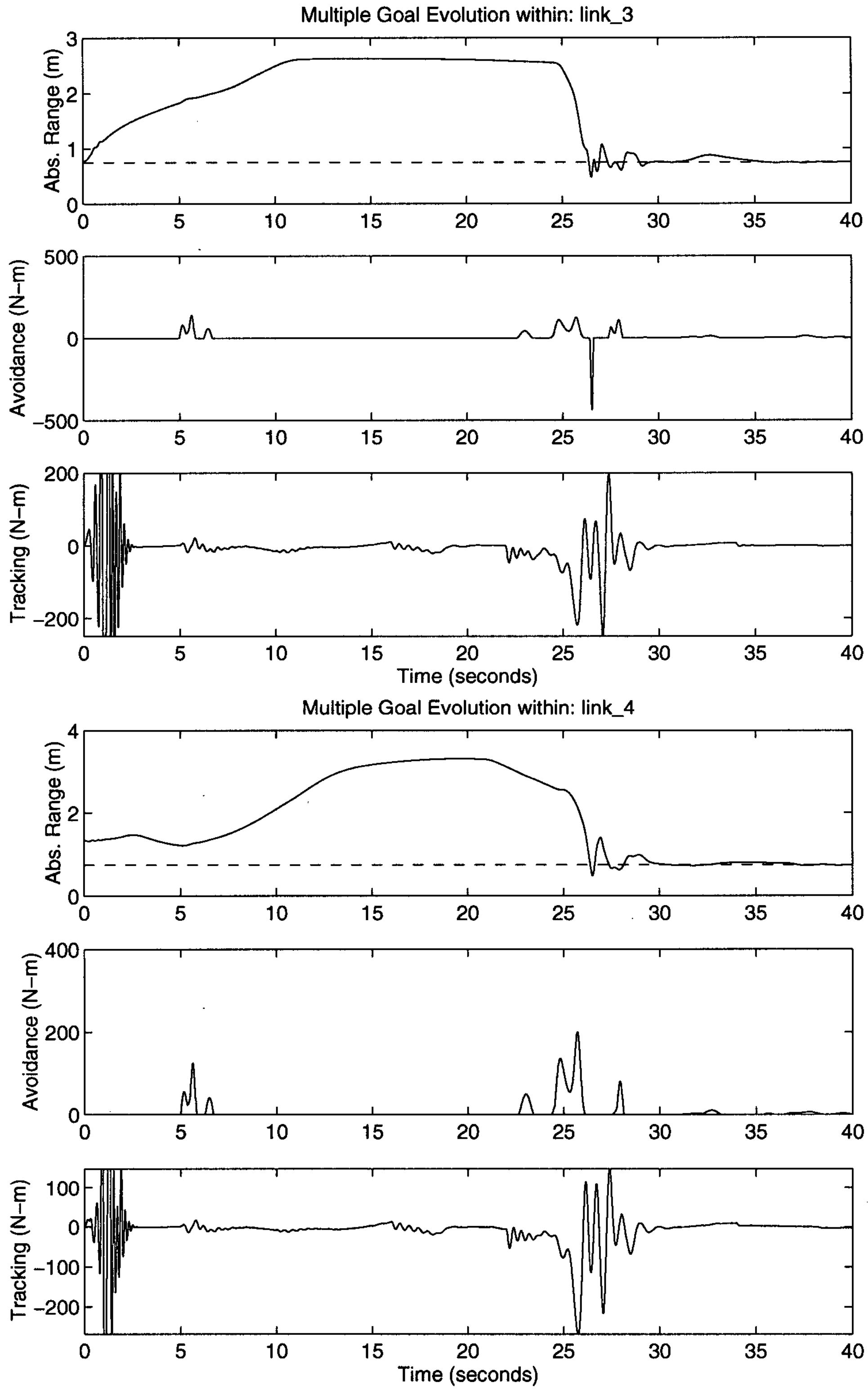


Figure 7.43: Evolution of the Global Goals within links 3(top) and 4 (bottom).

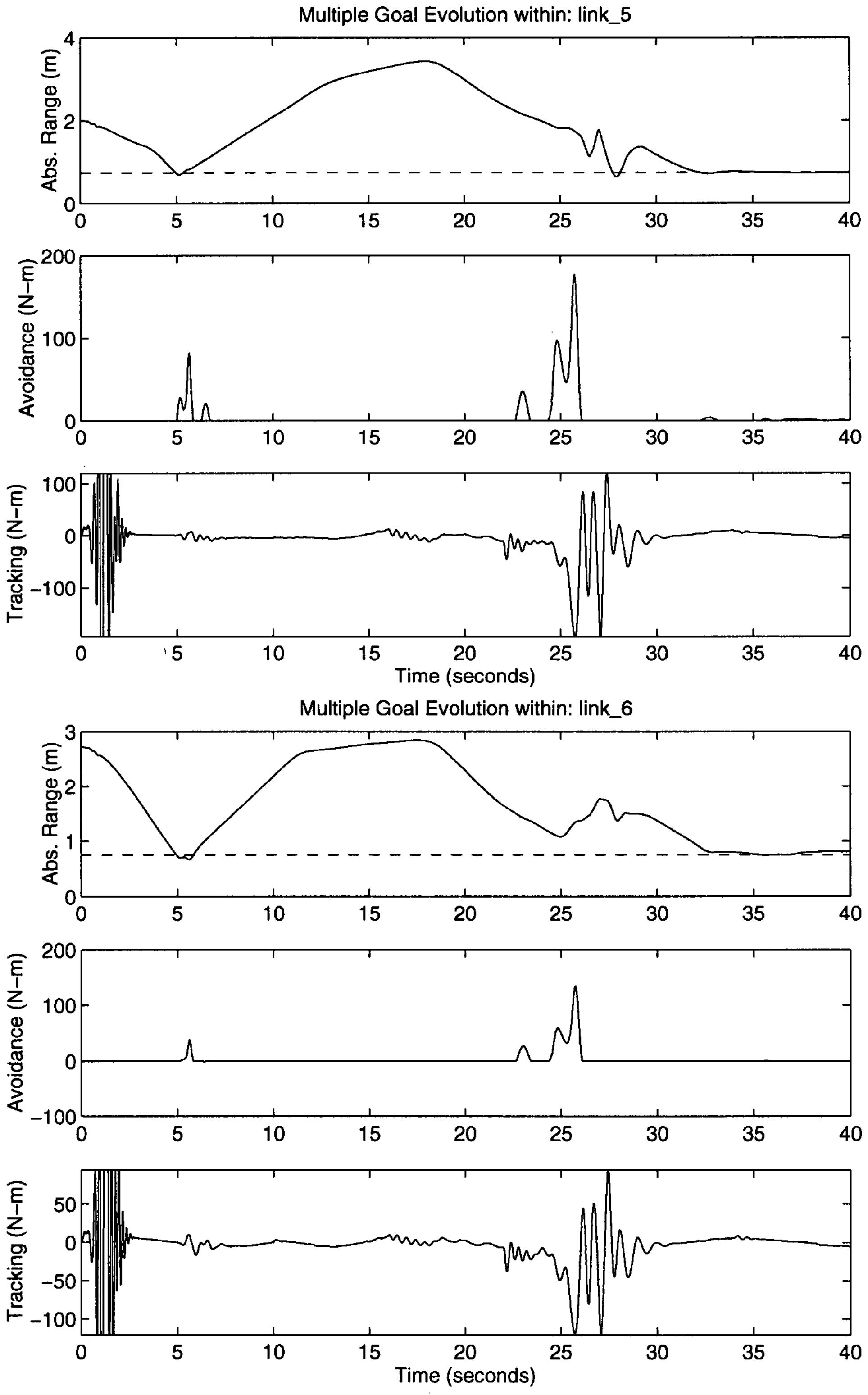


Figure 7.44: Evolution of the Global Goals within links 5(top) and 6 (bottom).

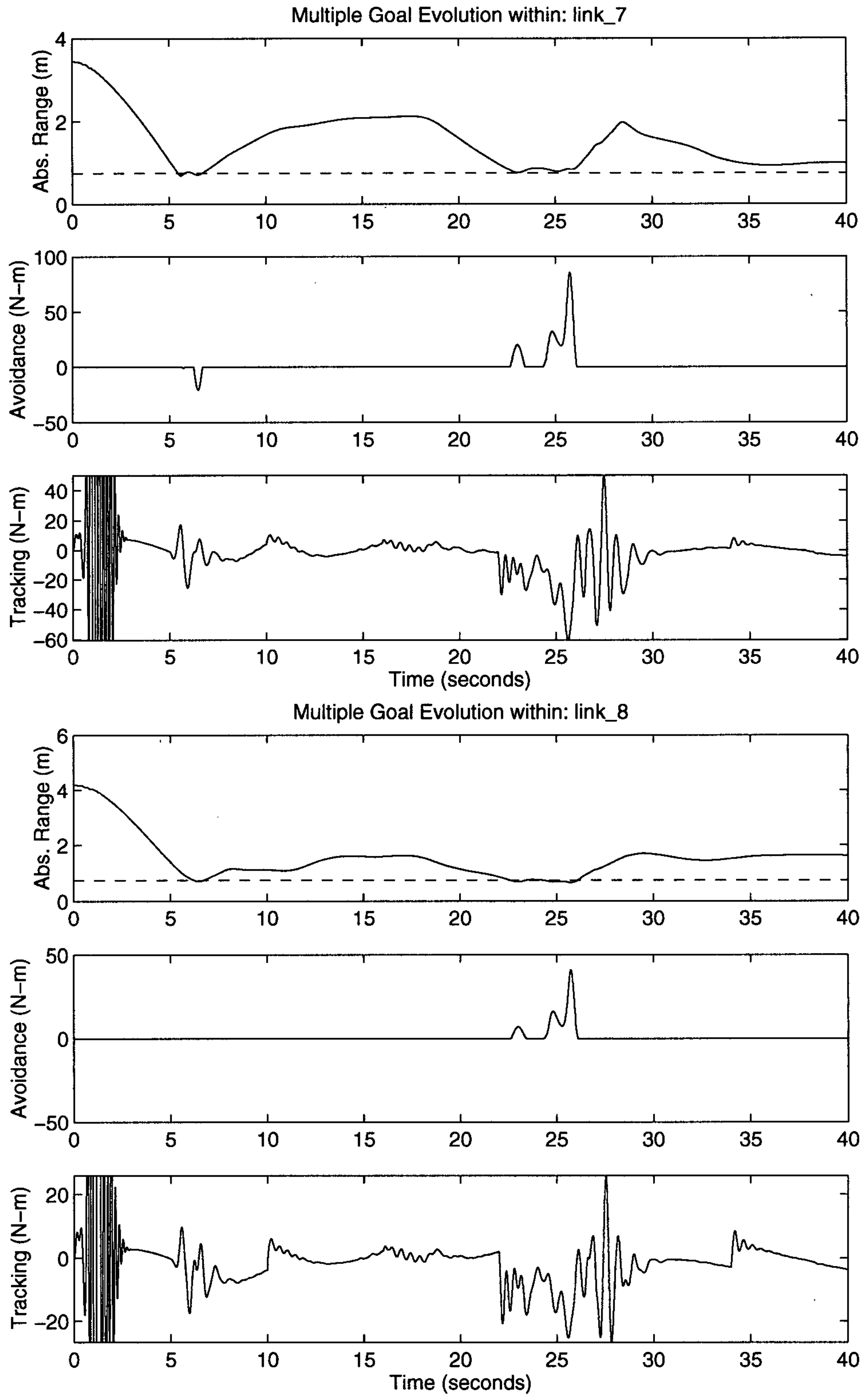


Figure 7.45: Evolution of the Global Goals within links 7(top) and 8 (bottom).

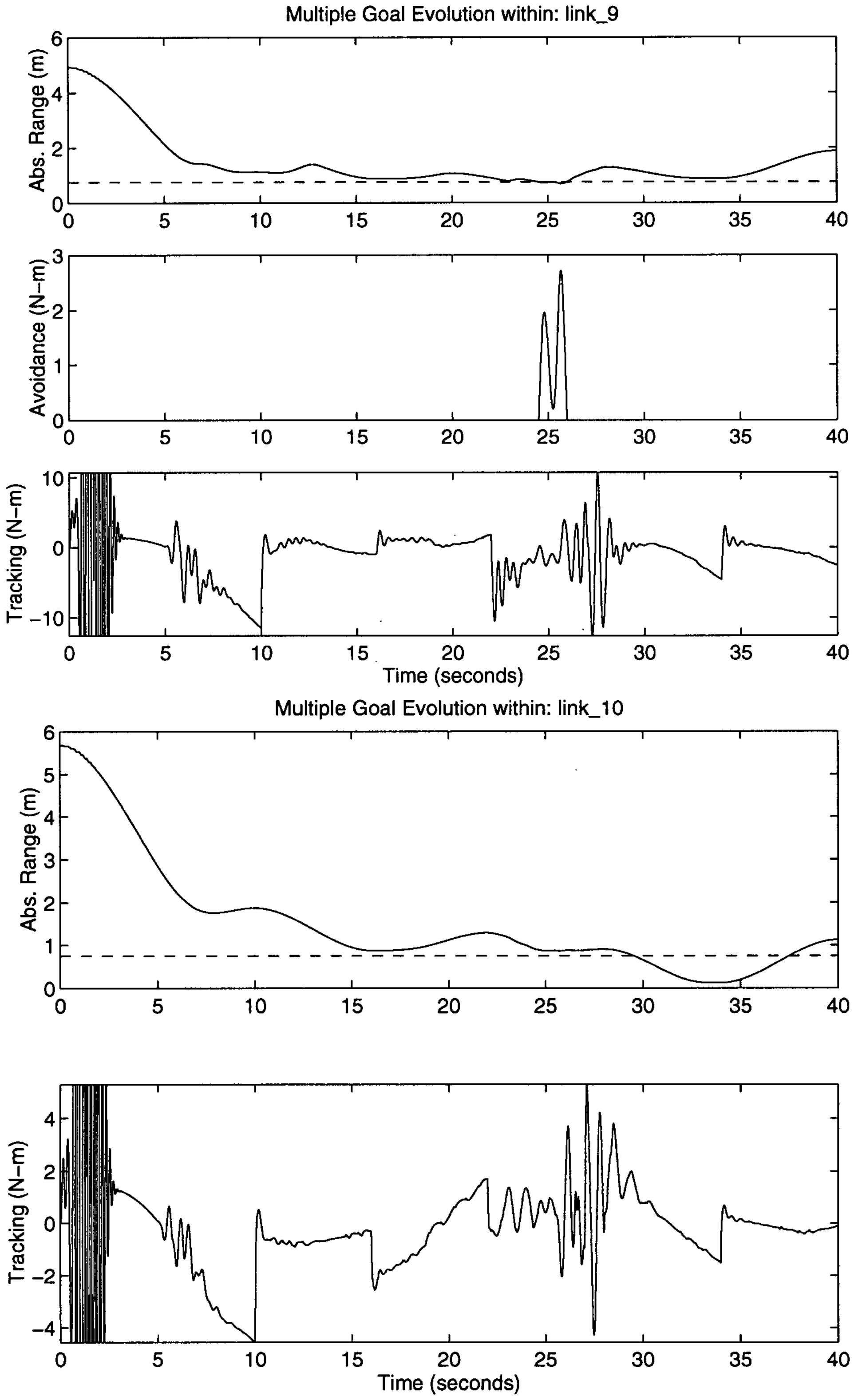


Figure 7.46: Evolution of the Global Goals within links 9(top) and 10 (bottom).

complex as $j \rightarrow 0$.

Of equal interest is the interaction between tracking and avoidance torques within each agent. Examining figure 7.44, both link 5 and link 6 exemplify the self arbitration interaction between the adaptive tracking goal and the relatively compliant avoidance goal. At approximately 24.5 seconds, both links receive the asserted avoidance request posted by link 9[3] . Figure 7.41 reveals that this disturbance is propagated to the end effector. In compensating for this disturbance, the global goal regulator changes the required tracking forces. An examination of links 5 and 6 reveals that these force changes appear as *inverted waveforms* of the avoidance torque profile over the same interval (approx. 24.5 to 25.5 seconds). Similar controller interactions can be observed within all the agents of the multiagent manipulator team.

This behaviour suggests that multiple global goal systems can, in effect, become self arbitrating in a redundant system. The combination and magnitude of tracking and avoidance torques are determined neither by a global explicit coordination mechanism nor a local agent explicit arbitration strategy, but through interaction of the system, the behaviour controllers, and the environment. Rather than modelling the environment and planning an explicit avoidance strategy, JTC obstacle avoidance schemes, including RMOA, demonstrate that obstacle avoidance and end effector tracking behaviours can be designed independently and superpositioned to achieve collision free motion in a cluttered environment. Though the independent global goals of trajectory tracking and obstacle avoidance are applied to the system, the system's global behaviour becomes a hybrid of the two systems, apparently without explicit or implicit coordination of the link agents.

7.3.4 Discussion

Unlike compliant task space controllers, end effector trajectories enforced through adaptive control must be changed to avoid mid course obstacles. Compliant controllers, however, can spontaneously avoid such obstacles by combining attractive and repulsive forces at the end effector. Overcoming linear repulsive controllers and unstable in combination with any other, adaptive controllers must actively formulate trajectories to avoid unexpected mid course obstacles.

One must agree with Khatib, that to avoid end effector collisions, the global goal generator must itself become an agent, adopting some obstacle avoidance strategy to construct safe *end effector* trajectories. Though such agent based obstacle avoidance strategies are numerous and would fit easily into the multiagent

[3]Occasionally, a link range plot indicates an obstacle intrusion *without* triggering a local avoidance response, implying $\mathbf{p}_{\mathrm{psh}_k} = \mathbf{x}_{j-1}$. Overlapping the clearance lozenges at the joints ensures avoidance behaviour in these cases.

system discussed in this study, ultimately they are ultimately another trajectory generator and will not be investigated here.

All goals are not always of the same priority, however. For tracking tasks obstacle avoidance and trajectory tracking are both of primary importance, divergence from either constituting a form of failure. Nevertheless, strict performance guarantees are often pursued for these less critical tasks (such as torque optimization, manipulability, etc.) and, in RMAC for example, have led to explicit task prioritization methods based on hierarchies of null space selection operators and switching logic [Nakamura, 1984]. In the next section, it will be shown that through appropriate specification of auxiliary goal systems, such prioritization/selection schemes can be greatly simplified.

7.4 Local and Global Goals Combined

Having demonstrated a global goal generation method in the last chapter and, in the last section, the interaction of multiple *global* goals, this section will investigate the interaction of *local* and global goals. Furthermore, this section will demonstrate that appropriate design of linear local and adaptive global behaviour controllers can result in self organizing, emergent behaviour of a multiagent team. Finally simulation results will be presented that establish multiagent control as a new, computationally lightweight method of redundancy resolution.

7.4.1 Why Local Goals?

Fundamentally, local goals provide each agent with control over local conditions. For example, a manipulator's free motion in $\mathbf{N}(\mathbf{J})$ can produce both undesirable and unpredictable dynamics. However, under local control this motion can be harnessed, enabling velocity minimization, free configuration space maximization, and joint limit avoidance. Therefore, the purpose behind local behaviour control is to reign in the less desirable characteristics of redundancy while achieving locally desirable states or properties. An additional objective, identified in the emergent multiagent control hypothesis, is that local control can lead to beneficial *globally* desirable states or properties.

Before entering into a discussion of local goal design, a brief examination of a common local auxiliary controller [Khatib, 1985], joint limit repulsion, will demonstrate the benefits and difficulties of local goal design in a multi-goal environment.

7.4.2 Continuous Nonlinear Joint Limit Repulsion

A persistent hazard of task space control, collision with joint limits, has led to the development of joint limit repulsion controllers. A good example, demonstrated here, is Khatib's joint limit repulsion strategy [Khatib, 1985]. A variant of his obstacle avoidance strategy, Khatib developed a force generator in which joint limit boundaries were enforced by the following nonlinear repulsive control law:

$$\mathbf{u}_{\text{limit}}(t) = \begin{cases} -\eta\left(\frac{1}{\rho_{\text{upper}}} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2} & \text{if } \rho_{\text{upper}} \leq \rho_0 \geq 0 \\ -\eta\left(\frac{1}{\rho_{\text{lower}}} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2} & \text{if } \rho_{\text{lower}} \leq \rho_0 \geq 0 \end{cases} \tag{7.254}$$

where

$$\rho_{\text{upper}} = q_{\text{upper}} - q \tag{7.255}$$

$$\rho_{\text{lower}} = q - q_{\text{lower}} \tag{7.256}$$

Combining this behaviour controller and trajectory tracking behaviour controller:

$$\mathbf{u}_{\text{track}} = \mathbf{J}_j^T(\mathbf{p}_n, \mathbf{x}_{j-1})\mathbf{f}_d \tag{7.257}$$

to each link agent controlling the reference manipulator or:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1]\begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{limit}} \end{bmatrix} \tag{7.258}$$

A physically realizable robot configuration history, i.e. within joint limits, can be observed. In the demonstration depicted in figure 7.48, the gain $\eta = 5.00$ is applied to the local joint limit repulsion goal. Clearly, the local repulsion goal prevent collision with the joint limit boundaries. Unfortunately, the non-linearity of these local goals can produce collision-like effects visible in figure 7.47, often disturbing the end effector trajectory. Indeed, some trial and error is required to settle upon a local gain sufficiently strong to prevent collision and yet maintain stability of the manipulator. Furthermore, the nonlinearity of the local goal exacerbates the problem by permitting virtually free motion in $\mathbf{N}(\mathbf{J})$ only to prevent joint space collisions precariously near the joint limit.

Clearly, arbitrary local goals do not guarantee stability of the system even in the presence of a relatively robust global goal generator. Some reliable design process is needed. The following sections will attack the local goal design problem by

- adopting a simple linear local controller

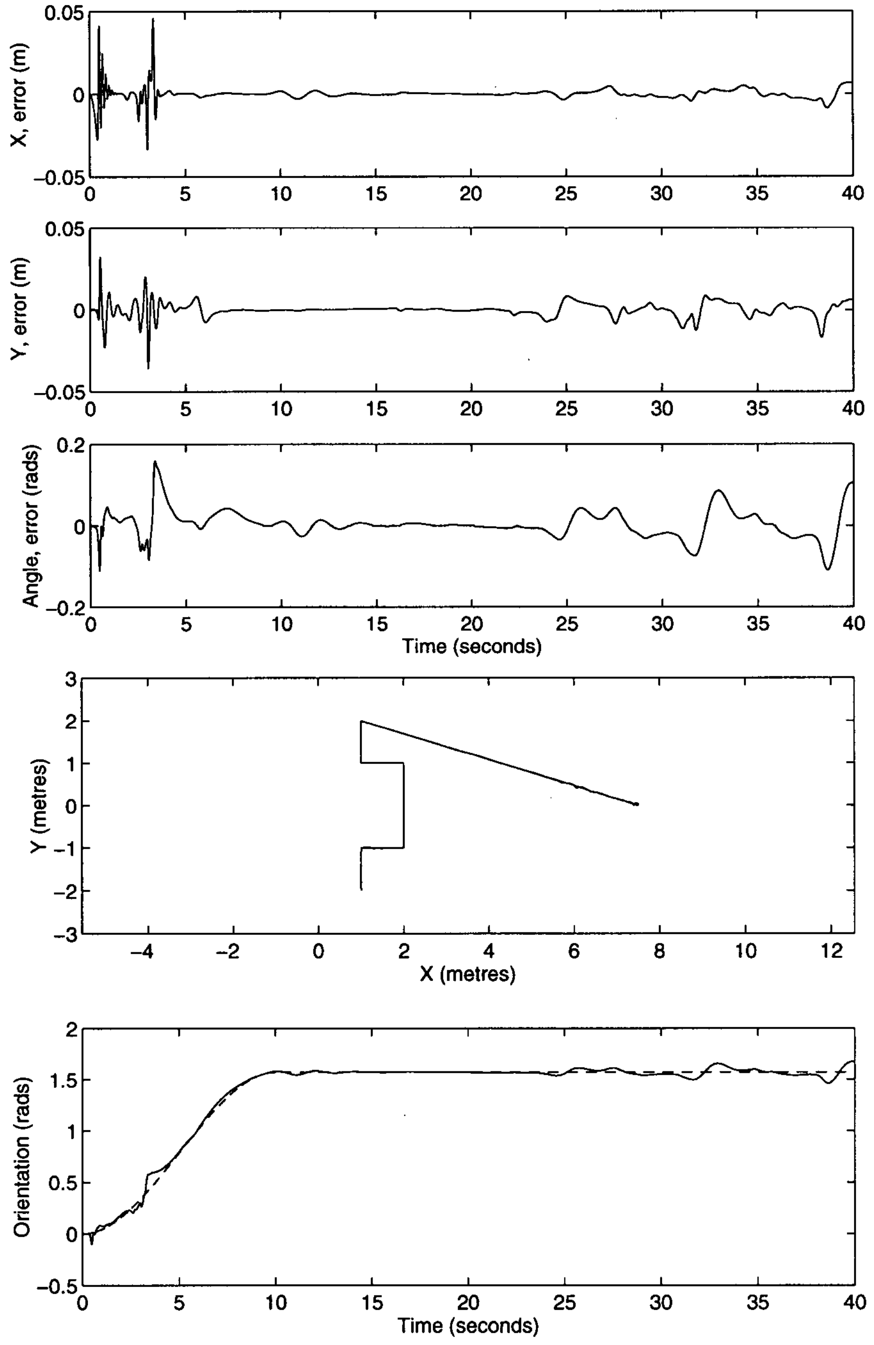


Figure 7.47: End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint limit avoidance ($\eta = 5.0$ N-m).

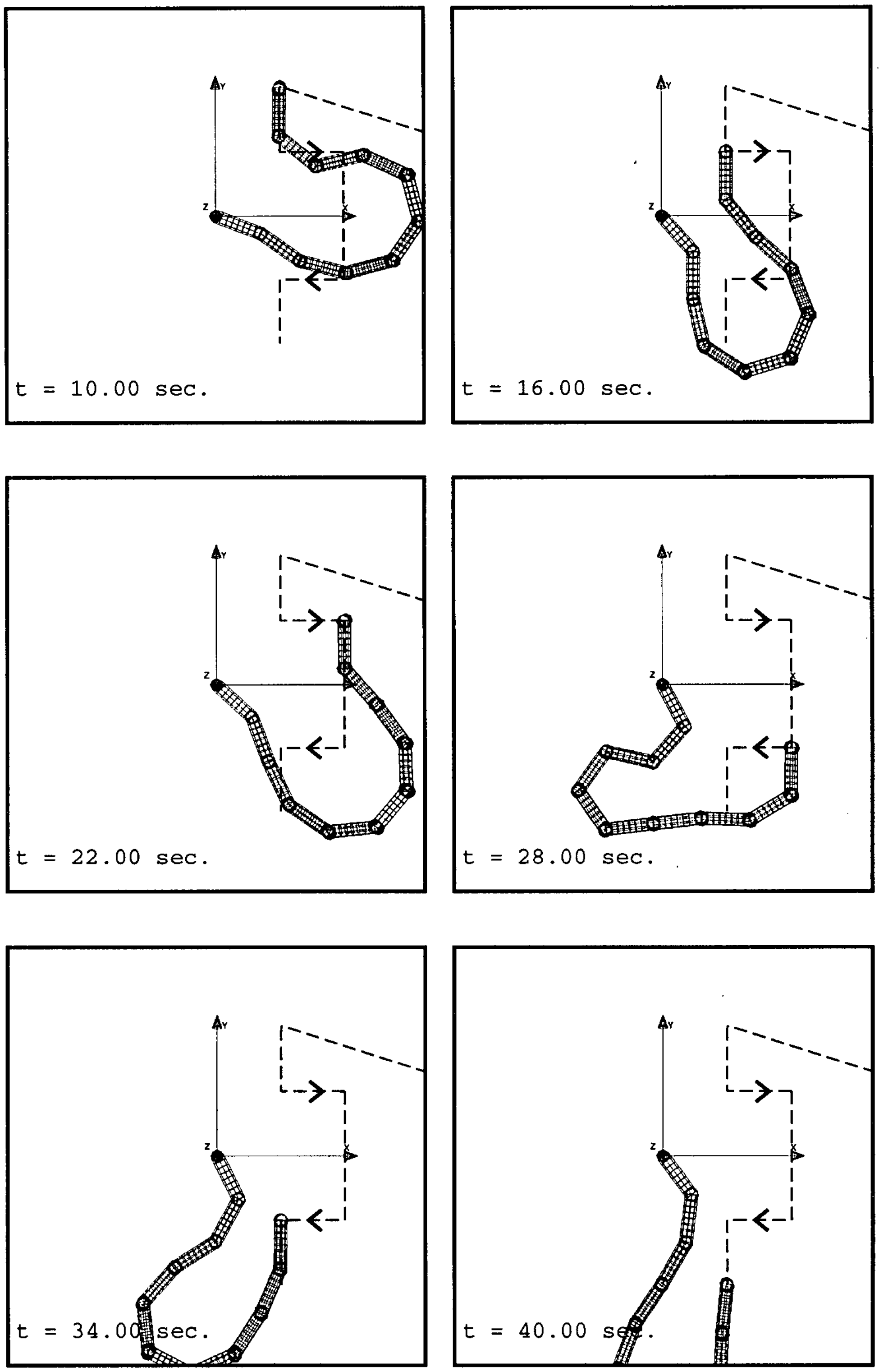


Figure 7.48: Both RMDAC end effector global and joint limit local goals are engaged.

- combining this with a linear global controller
- performing a linearized stability analysis.

7.5 Linear Local Goal Design

To separate the interaction and design of controllers from agent arbitrators assume, for the moment, that simple linear combination is an adequate arbitrator for the jth agent. For example:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1] \begin{bmatrix} \mathbf{u}_{\text{global}} \\ \mathbf{u}_{\text{local}} \end{bmatrix} \tag{7.259}$$

where $\mathbf{k}_{\mathsf{A}_j}$ is the arbitration vector. The local goal design problem then addresses how $\mathbf{u}_{\text{local}}$ should be formed. Just as in global behaviour control, a number of local behaviour controller models are possible: linear or nonlinear, continuous or discrete. Unlike global behaviour control alone, however, local behaviour control interacts with both local *and* global goal systems. To simplify the problem, a good starting point is to clarify the interaction between local and global goal systems through the adoption of a simple *linear* continuous local controller. By reviewing the design of a simple PD local controller, the implications of this selection on the global performance of the system and on the distribution of local control effort throughout the system can be predicted, simulated, and discussed.

Assume a PD controller is applied to the kth agent based on the assumption that the link is an *isolated linear system* of the form in equation (2.49):

$$J_{\text{eff}_k}\ddot{q}_{m_k} + (B_{\text{eff}_k} + K_k K_b)\dot{q}_{m_k} = K_k u_k - r_k d_k \tag{7.260}$$

where, as before:

$$J_{\text{eff}_k} = J_{\text{m}_k} + r^2 d_{kk} \tag{7.261}$$

$$d_k = \sum_{j\neq k} d_{jk}\ddot{q}_j + \sum_{i,j} C_{ijk}\dot{q}_i\dot{q}_j \tag{7.262}$$

Now for a simple PD controller and the with the gains k_q and k_w, reiterating equation (3.58):

$$u = k_q q_{e_k} + k_w \dot{q}_{e_k} \tag{7.263}$$

where $q_{e_k} = q_{d_k} - q_{m_k}$. Substituting and rearranging:

$$J_{\text{eff}_k}\ddot{q}_{m_k} + (B_{\text{eff}_k} + K_k K_b + K k_w)\dot{q}_{m_k} + K_k k_q q_{m_k} = K_k k_q q_{d_k} + K_k k_w \dot{q}_{d_k} - r_k d \tag{7.264}$$

Applying the Laplace Transform with zero initial conditions:

$$J_{\text{eff}_k} Q_{m_k}(s)s^2 + (B_{\text{eff}_k} + K_k K_b + K_k k_w) Q_{m_k}(s)s + K_k k_q Q_{m_k}(s) = K_k k_q Q_{d_k}(s) + K_k k_w Q_{d_k}(s)s - r_k D_k(s) \quad (7.265)$$

Developing the expression:

$$Q_{m_k}(s) = \left[\frac{K_k k_q + K_k k_w s}{\Delta_k(s)}\right] Q_{d_k}(s) - r_k \frac{D_k(s)}{\Delta_k(s)} \quad (7.266)$$

$$\Delta_k(s) = J_{\text{eff}_k} s^2 + (B_{\text{eff}_k} + K_k K_b + K_k k_w)s + K k_q \quad (7.267)$$

Now, this system will be stable if the roots of the characteristic equation $\Delta_k(s)$, reside in the left half plane. The steady state error is the product of two possible inputs Q_{d_k} and $D(s)$. Applying a unit step for both Q_{d_k} and $D(s)$, the final value theorem may be used to estimate the steady state error.

$$e_{ss}(t) = \lim_{s \to 0} s \left[\frac{1}{s} - \frac{K_k k_q + K_k k_w s}{\Delta_k(s)} + \frac{r g_b}{s \Delta_k(s)}\right] = \frac{r_k g_b}{K_k k_q} \quad (7.268)$$

Now, the remainder is that which remains constant in the robot equation, usually gravimetric forces. Therefore $g_b = |g_k|$ is the bound on the value of these forces. From this classical analysis one can conclude that the steady state error may be characterized by:

$$e_{ss}(t) < \frac{r_k g_b}{K k_q} \quad (7.269)$$

Thus as k_q increases steady state error diminishes. Using vector notation, we can describe the manipulator local controller behaviour as a typical PD manipulator controller or:

$$\mathbf{u}_{\text{local}} = \mathbf{K}_q \mathbf{q}_e(t) + \mathbf{K}_w \dot{\mathbf{q}}_e(t) \quad (7.270)$$

Given this classical PD joint controller construction, what are the implications of a similar local PD controller coexisting with a global goal system of the form discussed in chapter 6. The following sections will explore this topic through the combination of locally compliant and globally adaptive behaviour controllers.

7.6 Local Goal Strategies

In this section a number of local goal strategies will be described and combined with global goal strategies in simulation. Though based on the basic PD controller design above, each local goal adopts a specific compliance and/or setpoint combination to achieve unique global behaviour.

Since each link is controlled by both a three degree of freedom task space controller *and* a single degree of freedom joint controller, the manipulator will be overconstrained with a total of $n + 3$ constraints. As discussed earlier, this overconstraint should not produce a PD equilibrium offset at the end effector, the global RMDAC controller compensating for the local controllers in the steady state. However, those links that *are* fully constrained will, nevertheless, be able to transmit force and position requirements to other link agents by 'reflecting' control effort off the global goal regulator.

7.6.1 Fail Safe Locking and Robustness

The basic attraction of redundant manipulation is that a variety of configurations can achieve the same end effector task. Thus a disabled motor or controller need not bar the manipulation system from completing the task. However, this kinematic redundancy is usually not mirrored within traditional centralized controllers in which computer or algorithmic failure can be catastrophic.

In RMPC or off-line numerical methods, joint jamming or locking requires centralized detection and recomputation of the desired joint space solution. While tolerant of such failures, RMRC, RMAC, and traditional JTC, like RMPC, are model driven processes within centralized architectures. Software or hardware failures of any centralized controller can lead to catastrophic failure. The solution, backup controllers ('hot standbys') and/or remote operation, greatly complicates the control system and substantially elevates costs.

By exploiting a distributed computational model, Multiagent control is comparatively immune to system-wide computer failure. This architecture provides a simple solution to controller and/or motor failure with little or no computational penalty. Since end effector performance is not disturbed by frozen joints in a redundant manipulator under Jacobian transpose control, an agent can manage a joint failure through, for example, a 'fail-safe' behaviour that engages an emergency brake. This approach combined with the decentralized computation model described in chapter 6 provides the necessary hardware robustness to survive relatively severe controller and/or motor failures. By basing an agent's fail-safe behaviour on triggers such as suspicious sensor data or poor motor performance, a factor of safety can be designed into the manipulation system.

Of course, no system is immune to catastrophic failure and multiagent manipulator control is no exception. Multiagent control's weakness lies in each agent's dependency on kinematic packet transmission (assuming no local task space sensing is available). A failing rectified only through redundant communications [4].

Results

[4]Probably required by 'hot standbys' in RMAC anyway, redundant communications alone are, nevertheless, far less costly than redundant communications *and* redundant, centralized controllers.

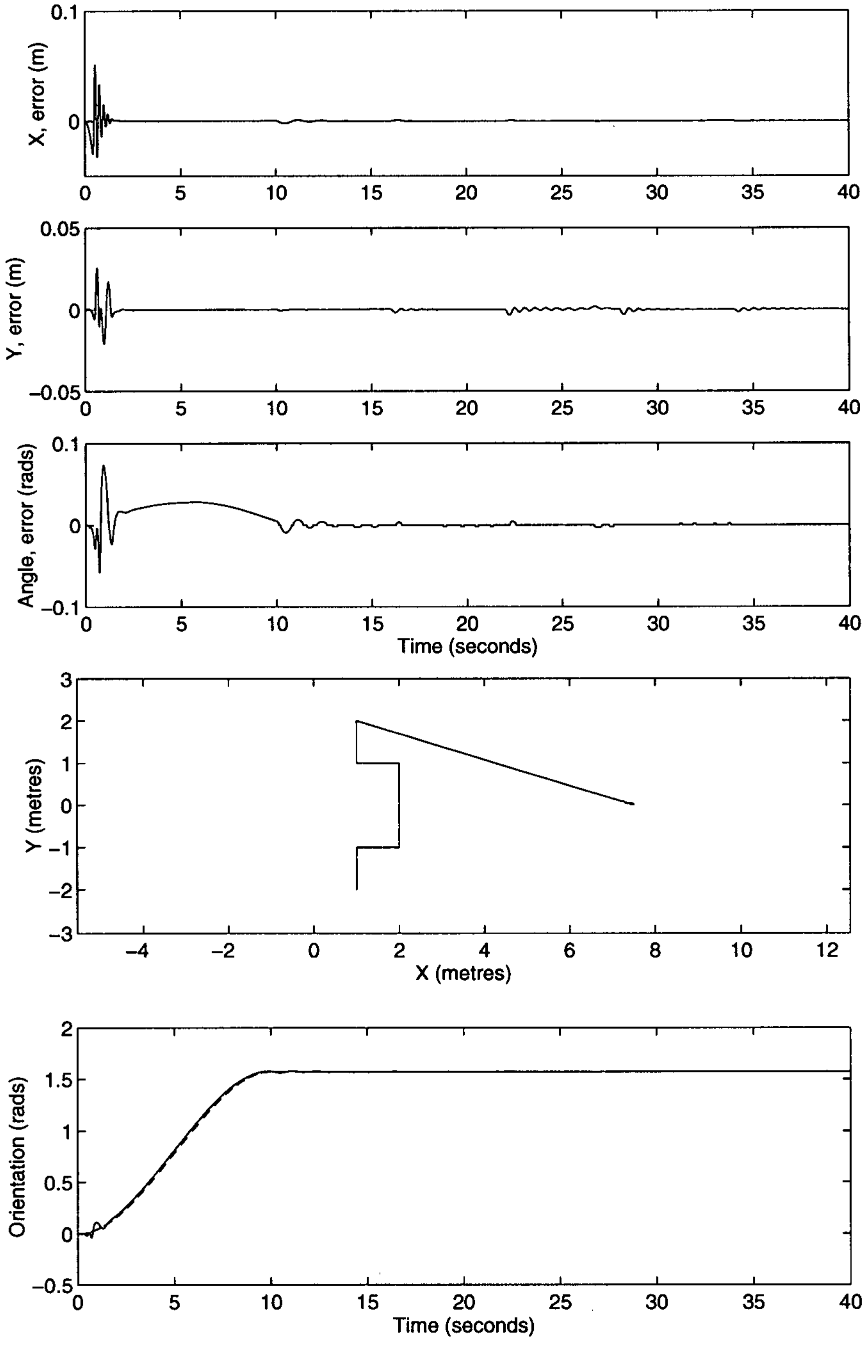


Figure 7.49: End effector trajectory error for the reference manipulator and payload under the failure of link 9.

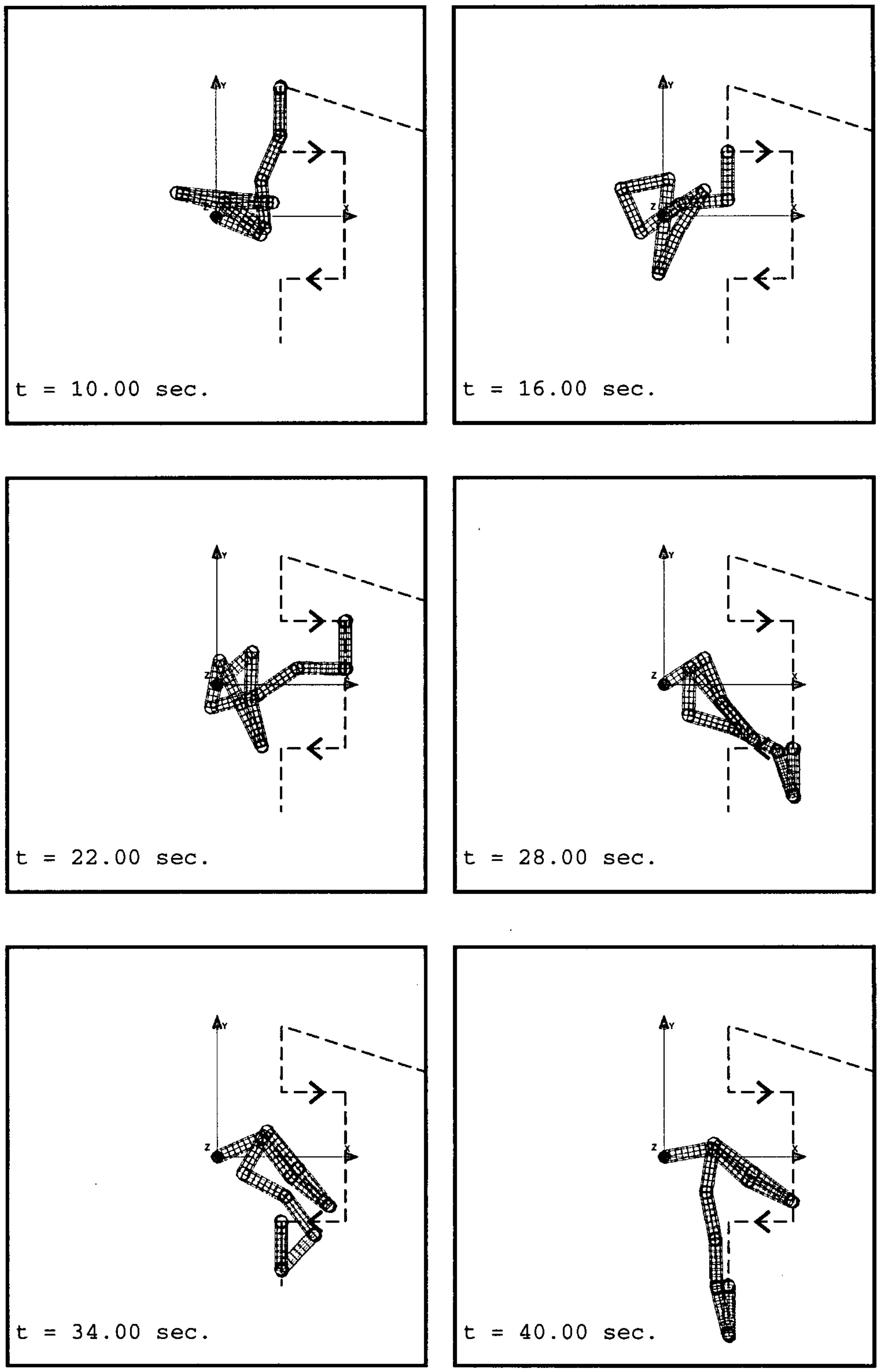


Figure 7.50: The manipulator configurations at the knotpoints of the reference trajectory for the reference manipulator and payload under the failure of link 9. The 'fail safe' braking behaviour locks link 9 to link 8.

Strategy	$\mathbf{e}_{\text{rms}}$	θ_{rms}
Reference with Failure	3.285e-03	5.528e-03
Reference	2.779e-03	2.850e-02

Table 7.12: Comparison of end effector RMS position and orientation error performance between reference and fail-safe behaviour of link 9.

The performance of a local fail safe braking strategy can be simulated through the application of a stiff local PD controller at the failed joint and the removal the global trajectory tracking behaviour from the joint's goal list. This has the effect of binding the failed link, j, to link $j-1$.

$$\mathbf{u}_{\text{brake}} = k_q q_{e_k}. \tag{7.271}$$

where k_q is large.

In the following simulation, each agent in the team pursues a global tracking goal while the disabled agent, `link 9`, employs a fail safe braking strategy. The multiagent system adopts the control laws:

$$\mathbf{u}_{cj} = \mathbf{u}_{\text{track}} \quad j = \{1, \ldots, 8, 10\} \tag{7.272}$$

$$\mathbf{u}_{c9} = \mathbf{u}_{\text{brake}} \tag{7.273}$$

In figure 7.50, the configuration history of the reference manipulator with link 9's goal system disabled and motion locked. Together, adaptive global goal enforcement and the decentralized architecture enable the manipulator to continue trajectory tracking without interruption or substantial increase in tracking error as depicted in figure 7.49 and recorded in table 7.12. The increased RMS error can be attributed to larger inertias in the link 8/ link 9 system that can effect the global behaviour's transitory response. RMAC methods can exhibit similar robust performance, but at the cost of greater computation (an $O(N^2)$ pseudoinversion and at least $O(N)$ dynamic model) and centralized control.

7.6.2 Constant Compliance Centering

Earlier in this chapter a joint limit avoidance local goal was introduced, exhibiting effective , though poorly behaved, performance. Another strategy is to minimize the probability of a joint limit collision by maximizing each joint's available free maneuvering range. One strategy for maximizing available 'maneuvering room' in joint space, is to enforce a joint centering strategy on the system. By specifying a time invariant setpoint midway between each joint's upper and lower bounds in the following manner:

$$q_{\text{mean}_k} = \frac{q_{\text{high}_k} + q_{\text{low}_k}}{2} \tag{7.274}$$

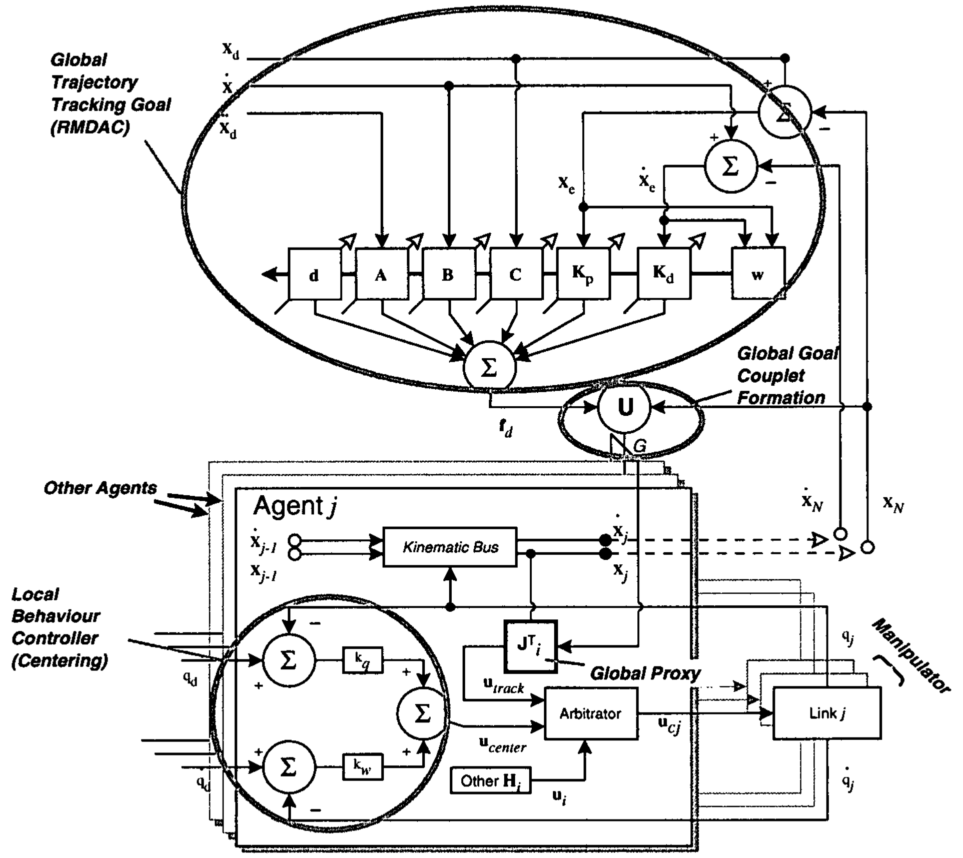


Figure 7.51: The structure of a multiagent system in which each agent has both local centering and global trajectory tracking (and other) behaviours.

a simple joint centering strategy can be devised. For simplicity, assume all agents are homogeneous, with identical PD controllers of the form:

$$\mathbf{u}_{\text{centering}} = k_q q_{c_k} + k_w \dot{q}_{c_k} \tag{7.275}$$

where $q_{c_k} = q_{\text{mean}_k} - q_k$. Where

$$k_{q_k} = C \;\; \forall k \tag{7.276}$$

and (k_{q_k}, k_{w_k}) are selected to be LHP stable and perfectly damped or:

$$k_{w_k} = 2\sqrt{k_{q_k}} \tag{7.277}$$

Figure 7.51 depicts the final structure in which each agent has both local centering and global tracking behaviours.

Intuitively, centering seems to be a reasonable method of ensuring each joint tends to remain in the middle of its range, but does this strategy, in fact, *globally maximize* maneuvering room (or globally *minimize* displacement from the midrange) over the trajectory?

Joint Centering As Optimization

Joint Centering can be shown to globally minimize joint displacement from the midrange over the end effector trajectory. Kazerounian and Wang [Kazerounian, 1988] demonstrated that locally minimizing joint acceleration through RMAC (a least squares pseudoinverse solution for joint acceleration), *globally* minimizes the performance index:

$$I = \int_{t_0}^{t_f} \dot{\mathbf{q}}^T \mathbf{A}(t) \dot{\mathbf{q}} \; dt \tag{7.278}$$

and, therefore, the joint velocity if the symmetric matrix $\mathbf{A}(t) = \mathbf{I}$. By augmenting this index with a potential energy term a similar development will show that insertion of a proportional joint controller into $\mathbf{N}(\mathbf{J})$ globally minimizes joint displacement:

$$I = \int_{t_0}^{t_f} \mathbf{q}^T \mathbf{K}_q \mathbf{q} + \dot{\mathbf{q}}^T \mathbf{A} \dot{\mathbf{q}} \; dt \tag{7.279}$$

subject to the forward kinematic solution $G((\mathbf{q}), t) = \mathbf{x}(t) - \mathbf{f}(\mathbf{q}) = 0$. The Hamiltonian becomes:

$$H(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{q}^T \mathbf{K}_q \mathbf{q} + \dot{\mathbf{q}}^T \mathbf{A} \dot{\mathbf{q}} + \lambda^T G((\mathbf{q})) \tag{7.280}$$

making an augmented performance index:

$$I^* = \int_{t_0}^{t_f} H(\mathbf{q}, \dot{\mathbf{q}}, t) \; dt \tag{7.281}$$

Kazerounian then applies the calculus of variations in I^* to produce:

$$\delta I^* = \int_{t_0}^{t_f} \left[\frac{\partial H}{\partial \mathbf{q}} - \frac{d}{dt} \frac{\partial H}{\partial \dot{\mathbf{q}}} \right] \delta \mathbf{q} dt + \left[\frac{\partial H}{\partial \dot{\mathbf{q}}(t_f)} \right] \delta \mathbf{q}(t_f) - \left[\frac{\partial H}{\partial \dot{\mathbf{q}}(t_0)} \right] \delta \mathbf{q}(t_0) \tag{7.282}$$

For an arbitrary variation $\delta \mathbf{q}$, the PI is minimized if:

$$\frac{\partial H}{\partial \mathbf{q}} - \frac{d}{dt} \frac{\partial H}{\partial \dot{\mathbf{q}}} = 0 \tag{7.283}$$

The term:

$$\frac{\partial H}{\partial \mathbf{q}} = 2\mathbf{K}_q \mathbf{q} + \lambda^T \frac{\partial G}{\partial \mathbf{q}} + \frac{\partial (\dot{\mathbf{q}}^T \mathbf{A})}{\partial \mathbf{q}} \dot{\mathbf{q}} \tag{7.284}$$

Of course the term $\frac{\partial G}{\partial \mathbf{q}}$ is simply $\mathbf{J}$, the Jacobian of the forward solution. The term:

$$\frac{d}{dt} \frac{\partial H}{\partial \dot{\mathbf{q}}} = 2\mathbf{A} \ddot{\mathbf{q}} + 2\dot{\mathbf{A}} \dot{\mathbf{q}} \tag{7.285}$$

Equation 7.283 then becomes:

$$2\mathbf{K}_q\mathbf{q} + \lambda^T\mathbf{J} + \frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - (2\mathbf{A}\ddot{\mathbf{q}} + 2\dot{\mathbf{A}}\dot{\mathbf{q}}) = 0 \tag{7.286}$$

Solving for $\ddot{\mathbf{q}}$:

$$\ddot{\mathbf{q}} = \mathbf{A}^{-1}(\mathbf{K}_q\mathbf{q} + \lambda^T\mathbf{J} + 0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}}\dot{\mathbf{q}}) \tag{7.287}$$

Twice differentiating the constraint, $G((\mathbf{q}))$, produces the familiar equation:

$$\mathbf{J}\ddot{\mathbf{q}} = \ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}} \tag{7.288}$$

into which $\ddot{\mathbf{q}}$ may be substituted :

$$\mathbf{J}\mathbf{A}^{-1}(\mathbf{K}_q\mathbf{q} + \lambda^T\mathbf{J} + 0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}}\dot{\mathbf{q}}) = \ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}} \tag{7.289}$$

$$\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T\lambda + \mathbf{J}\mathbf{A}^{-1}[\mathbf{K}_q\mathbf{q} + (0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}})\dot{\mathbf{q}}] = \ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}} \tag{7.290}$$

Now solving for λ:

$$\lambda = (\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T)^{-1}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) - (\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{A}^{-1}(\mathbf{K}_q\mathbf{q} + (0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}})\dot{\mathbf{q}}) \tag{7.291}$$

Backsubtituting into the expression 7.287:

$$\begin{aligned}\ddot{\mathbf{q}} &= \mathbf{A}^{-1}(\mathbf{K}_q\mathbf{q} + \mathbf{J}^T[(\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T)^{-1}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) - (\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{A}^{-1}[\mathbf{K}_q\mathbf{q} + (0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}})\dot{\mathbf{q}}]] \\ &\quad +(0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}})\dot{\mathbf{q}} \qquad (7.292) \\ &= \mathbf{J}_A^\dagger(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + (\mathbf{I} - \mathbf{J}_A^\dagger\mathbf{J})\mathbf{A}^{-1}[\mathbf{K}_q\mathbf{q} + (0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{A})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{A}})\dot{\mathbf{q}}] \qquad (7.293)\end{aligned}$$

where:

$$\mathbf{J}_A^\dagger = \mathbf{A}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T)^{-1} \tag{7.294}$$

is the *weighted* pseudoinverse of the Jacobian matrix. If $\mathbf{A} = \mathbf{I}$ and $\mathbf{K}_q = 0$, then $\mathbf{J}_A^\dagger = \mathbf{J}^\dagger$, and (7.293) is equivalent to the minimum velocity solution.If the only joint value requirements are the satisfaction of the end effector forward kinematic solution constraint at t_f and t_0, then the joint velocities must satisfy the boundary conditions [Kazerounian, 1988]:

$$\dot{\mathbf{q}} = \mathbf{J}_A^\dagger\dot{\mathbf{x}} \tag{7.295}$$

at both t_f and t_0.

However, both [Kazerounian, 1988] and [Colbaugh, 1989] show that by setting $\mathbf{A} = \mathbf{D}(\mathbf{q})$, free motion in the null space of $\mathbf{J}_A$ minimizes the kinetic energy of the system. Substituting into equation (7.293):

$$\ddot{\mathbf{q}} = \mathbf{J}_D^{\dagger}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + (\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})\mathbf{D}^{-1}[\mathbf{K}_q\mathbf{q} + (0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{D})}{\partial\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{D}})\dot{\mathbf{q}}] \tag{7.296}$$

and recognizing the relation:

$$\mathbf{C}(\dot{\mathbf{q}}, \mathbf{q}) = (\dot{\mathbf{D}} - 0.5\frac{\partial(\dot{\mathbf{q}}^T\mathbf{D})}{\partial\mathbf{q}}\dot{\mathbf{q}}) \tag{7.297}$$

equation (7.296) reduces to:

$$\ddot{\mathbf{q}} = \mathbf{J}_D^{\dagger}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + (\mathbf{I} - \mathbf{J}_A^{\dagger}\mathbf{J})\mathbf{D}^{-1}[\mathbf{K}_q\mathbf{q} - \mathbf{C}\dot{\mathbf{q}}] \tag{7.298}$$

with the natural boundary conditions [Kazerounian, 1988] at t_0 and t_f:

$$\dot{\mathbf{q}} = \mathbf{J}_D^{\dagger}\dot{\mathbf{x}} \tag{7.299}$$

If $\mathbf{K}_q = 0$ equation (7.298) describes the accelerations within a redundant manipulator under end effector control alone. Observe that these accelerations are weighted (as one might expect) by the manipulator inertia matrix and that accelerations in the Null space are governed by coriolis and centrifugal forces. By restructuring this equation as a force-torque expression (as in [Colbaugh, 1989]), it can be shown that insertion of a simple proportional controller into the Jacobian null space is equivalent to a minimum displacement strategy.

Recall the structure of the Robot equation in both task space and configuration space coordinates:

$$\begin{aligned} \tau &= \mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) & (7.300) \\ \mathbf{f} &= \mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{N}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{p}(\mathbf{x}) & (7.301) \end{aligned}$$

where $\mathbf{M}(\mathbf{x})$, $\mathbf{N}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$ and $\mathbf{p}(\mathbf{x})$ are related to $\mathbf{D}(\mathbf{q})$,$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$, and $\mathbf{g}(\mathbf{q})$ respectively in equations (E.421), (E.422), and (E.423). Now given a redundant manipulator with the inertia-weighted pseudoinverse:

$$\mathbf{J}_D^{\dagger} = \mathbf{D}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{D}^{-1}\mathbf{J}^T)^{-1} \tag{7.302}$$

the task space equation can be rewritten:

$$\mathbf{f} = (\mathbf{J}\mathbf{D}^{-1}\mathbf{J}^T)^{-1}[\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}] + \mathbf{J}_D^{\dagger T}[\mathbf{C}\dot{\mathbf{q}} + \mathbf{g}] \tag{7.303}$$

Substituting equation 7.298 into the configuration space robot equation:

$$\tau = \mathbf{D}\left[\mathbf{J}_D^{\dagger}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + (\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})\mathbf{D}^{-1}[\mathbf{K}_q\mathbf{q} - \mathbf{C}\dot{\mathbf{q}}]\right] + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{7.304}$$

Expanding $\mathbf{J}_D$ and simplifying:

$$\tau = \mathbf{D}\mathbf{D}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{D}^{-1}\mathbf{J}^T)^{-1}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + \mathbf{D}(\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})\mathbf{D}^{-1}[\mathbf{K}_q\mathbf{q} - \mathbf{C}\dot{\mathbf{q}}] + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{7.305}$$

Simplifying:

$$\tau = \mathbf{J}^T(\mathbf{J}\mathbf{D}^{-1}\mathbf{J}^T)^{-1}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + (\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})[\mathbf{K}_q\mathbf{q} - \mathbf{C}\dot{\mathbf{q}}] + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{7.306}$$

and substituting $\mathbf{f}$ and rearranging:

$$\tau = \mathbf{J}^T\mathbf{f} - \mathbf{J}^T\mathbf{J}_D^{\dagger T}[\mathbf{C}\dot{\mathbf{q}} + \mathbf{g}] + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + (\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})[\mathbf{K}_q\mathbf{q} - \mathbf{C}\dot{\mathbf{q}}] \tag{7.307}$$

or simply:

$$\tau = \mathbf{J}_D^T\mathbf{f}_d + (\mathbf{I} - \mathbf{J}_D^{\dagger}\mathbf{J})[\mathbf{K}_q\mathbf{q} + \mathbf{g}(\mathbf{q})] \tag{7.308}$$

From this last equation, one can conclude that proportional control in the null space of $\mathbf{J}_D$ globally minimizes displacement if $\mathbf{g}(\mathbf{q}) \approx 0$. The effects of the addition of derivative control is not clear using these arguments. A PI based on damping energy term $\dot{\mathbf{q}}^T\mathbf{K}_w\dot{\mathbf{q}}$ becomes $\dot{\mathbf{q}}^T[\mathbf{K}_w + \mathbf{D}]\dot{\mathbf{q}}$ and ultimately acts to bias the weighted pseudoinverse.

The above development has shown that a proportional centering strategy inserted into $\mathbf{N}(\mathbf{J}_D)$ globally minimizes displacement energy (and therefore displacement). However, in the current technique there is no explicit insertion process, rather the 'natural' evolution of perturbations described in section 7.2 performs this coordination automatically. Unfortunately, since RMDAC requires finite time to ensure end effector tracking and, moreover, cannot guarantee convergence of parameters[5], the actual performance is probably a suboptimal displacement energy solution. Nevertheless, in accepting suboptimality, a decentralized, self coordinated system can be demonstrated.

Results

In the following simulations each agent employs a joint centering local goal in conjunction with the reference trajectory tracking global goal. In effect, each PD controller models a spring-damper system located at each link's joint. The agent output:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T\mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1]\begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{centering}} \end{bmatrix} \tag{7.309}$$

In the demonstration run, depicted in figures 7.52 and 7.53, local gains k_q and k_w are set to 100 and 20 respectively, with all controllers operating at 120Hz. The results may be viewed from both a local and aggregate perspective.

[5] i.e. equations (E.421), (E.422), and (E.423) are not necessarily true

Locally, each agent's centering behaviour keeps each joint near its midrange and, as a consequence, minimizes clashes with joint limits. Lacking the nonlinear boundary controller, centering behaviours produce less disturbed end effector performance and greater predictability than the joint limit behaviour described earlier, though they do not guarantee joint limits will not be exceeded.

The *aggregate* effect is twofold. First, the local strategy induces the predicted local minimization. Secondly, the local strategy enforces a manipulator 'shaping' policy much like a 'leaf spring'. Both are the product of complex interactions between local and global behaviour controllers.

The local centering strategy seems to maximize the available maneuvering volume within the limits of the usable C-space and exhibit a smooth curvature, an unplanned (though not unforeseeable) global behaviour. Just as in end effector trajectory tracking, a root mean square (RMS) measure characterizes the optimality of the manipulator's centering behaviour over the reference trajectory and is defined as:

$$\mathbf{q}_{\mathrm{rms}} = \left[\frac{1}{N_s} \sum_{k=0}^{N_s} (\mathbf{q}_d(k) - \mathbf{q}(k))^T (\mathbf{q}_d(k) - \mathbf{q}(k)) \right]^{\frac{1}{2}} \tag{7.310}$$

where N_s is the total number of timesteps. Table 7.13 documents the erosion of RMS end effector tracking accuracy as well as the reduction of the RMS joint displacement error with increasing local goal stiffness. Though slowing the global goal's convergence to an ideal tracking process, even relatively small local stiffnesses greatly improve the optimal displacement measure, $\mathbf{q}_{\mathrm{rms}}$. Dynamically, the aggregate behaviour mimicks a spring-loaded or leaf spring mechanism, similar to configurations described in [Slotine, 1988].

Despite the simplicity of this strategy, examination of the torque histories (figures 7.54-7.58), reveals that this behaviour is the product of complex interaction between local and global constraints. As established earlier, end effector control drives disturbance forces into $\mathbf{N}(\mathbf{J})$. From the lone agent's perspective, the adaptive correction process allows local goals, $\delta\mathbf{u}$, to be partially fulfilled while fulfilling the global objective. In terms of multiagent control, this process effectively *reflects* a local goal to the agent team. In this sense, cartesian adaptive controllers provide a communication infrastructure for an *emergent coordination strategy* between arbitrary local goal systems.

The torque histories in figures 7.54-7.58 clearly document this coordination through the cancellation of local link centering forces, but only insofar as they interfere with the global goal. This cancellation is indicated by mirrored waveforms between the link's centering and trajectory tracking torque plots. In the steady state ($\dot{\mathbf{q}} = 0$), these forces balance exactly as predicted by the stability analysis.

In an analysis of the locally and globally compliant systems it will be shown in the next section that these systems are coupled and that the performance of the combined system is a nonlinear hybrid of both systems.

No.	k_q	k_w	$\mathbf{e}_{\text{rms}}$	θ_{rms}	$\mathbf{q}_{\text{rms}}$
01	0.00	0.00	3.059e-03	6.014e-03	7.848e+00
05	30.00	10.95	4.573e-03	8.034e-03	2.813e+00
04	50.00	14.14	5.023e-03	9.346e-03	2.738e+00
02	100.00	20.00	5.963e-03	1.424e-02	2.700e+00
03	225.00	30.00	8.183e-03	2.988e-02	2.675e+00

Table 7.13: Tabulation of local joint centering PD gains versus RMS task space position error and RMS centering error for a simple joint centering strategy.

The same may be said of the locally compliant and globally adaptive multiagent system: isolated stability of local and global systems does not guarantee combined stability. To ensure stability of the RMDAC generator, the manipulator's parameters must not appear to change suddenly. For example, setting local gains above $k_q = 400$ and $k_w = 40$ produces instability at the reference control rate of 120Hz. Stability can be recovered by either raising local control rates or using underdamped local derivative gains. Either tactic reduces the apparent change in manipulator parameters at the end effector.

As the end effector approaches the origin, the local deflections become greater (and the 'leaf spring' more 'compressed'), forcing RMDAC to become increasingly stiff and, often, less stable. The combined effect is a stability gradient imposed on the work space dependent both on the position of the end effector and manipulator configuration. Thus marginally stable centering/tracking goal combinations at the start of the trajectory may become unstable near the origin. Indeed, for marginally stable cases many explode numerically either in the first few seconds of the reference trajectory or approximately at the midway point ($t = 20.0$ sec.).

While noting that local stiffness is a source of instability, large local position gains are not necessary to institute a manipulator shaping policy in $\mathbf{N}(\mathbf{J})$. Rather, it is the existence of such local behaviours that dominate the manipulator shape as indicated by the optimality measures, $\mathbf{q}_{\text{rms}}$, in table 7.13.

7.6.3 Constant Compliance Retraction

The 'leaf spring' configuration behaviours demonstrated thus far maximize the available maneuvering volume for each link, minimizing the curvature along the manipulator. In effect, as the end effector approaches the origin, the local centering behaviours becomes more compressed, requiring greater stiffness by the adaptive global goal generation system to maintain the trajectory. Since stability and global goal stiffness are coupled, this strategy imposes a stability gradient on the system. This approach also biases the manipulator position

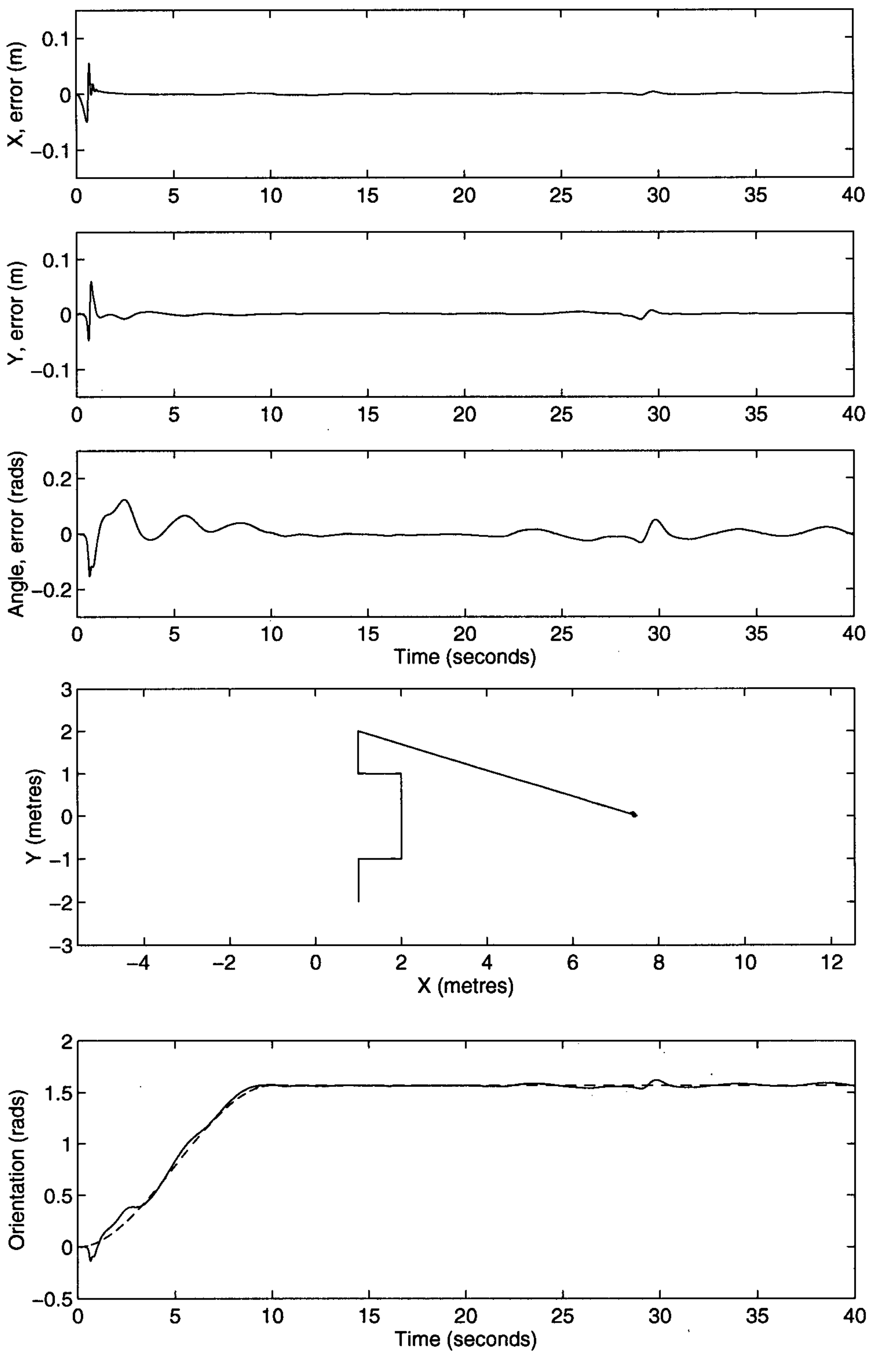


Figure 7.52: End effector trajectory for the reference manipulator and payload under both RMDAC end effector global and joint centering local goals.

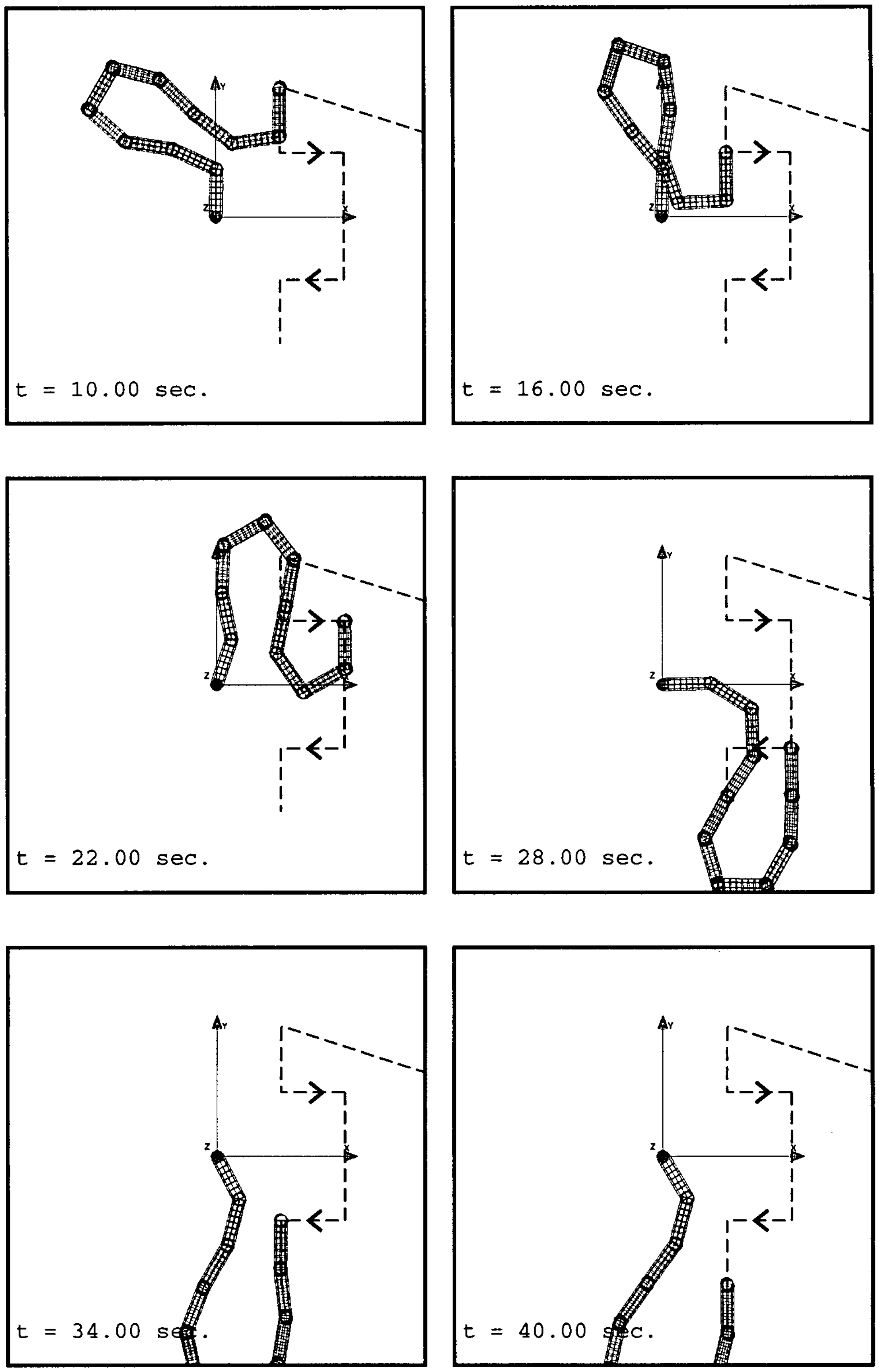


Figure 7.53: The manipulator configurations at the knotpoints of the reference trajectory for the reference manipulator and payload under both RMDAC end effector global and joint centering local goals. Note the manipulator *simultaneously* adopts a 'leaf spring' configuration while tracking the reference trajectory, indicating that joint centering forces are acting in $\mathbf{N}(\mathbf{J})$.

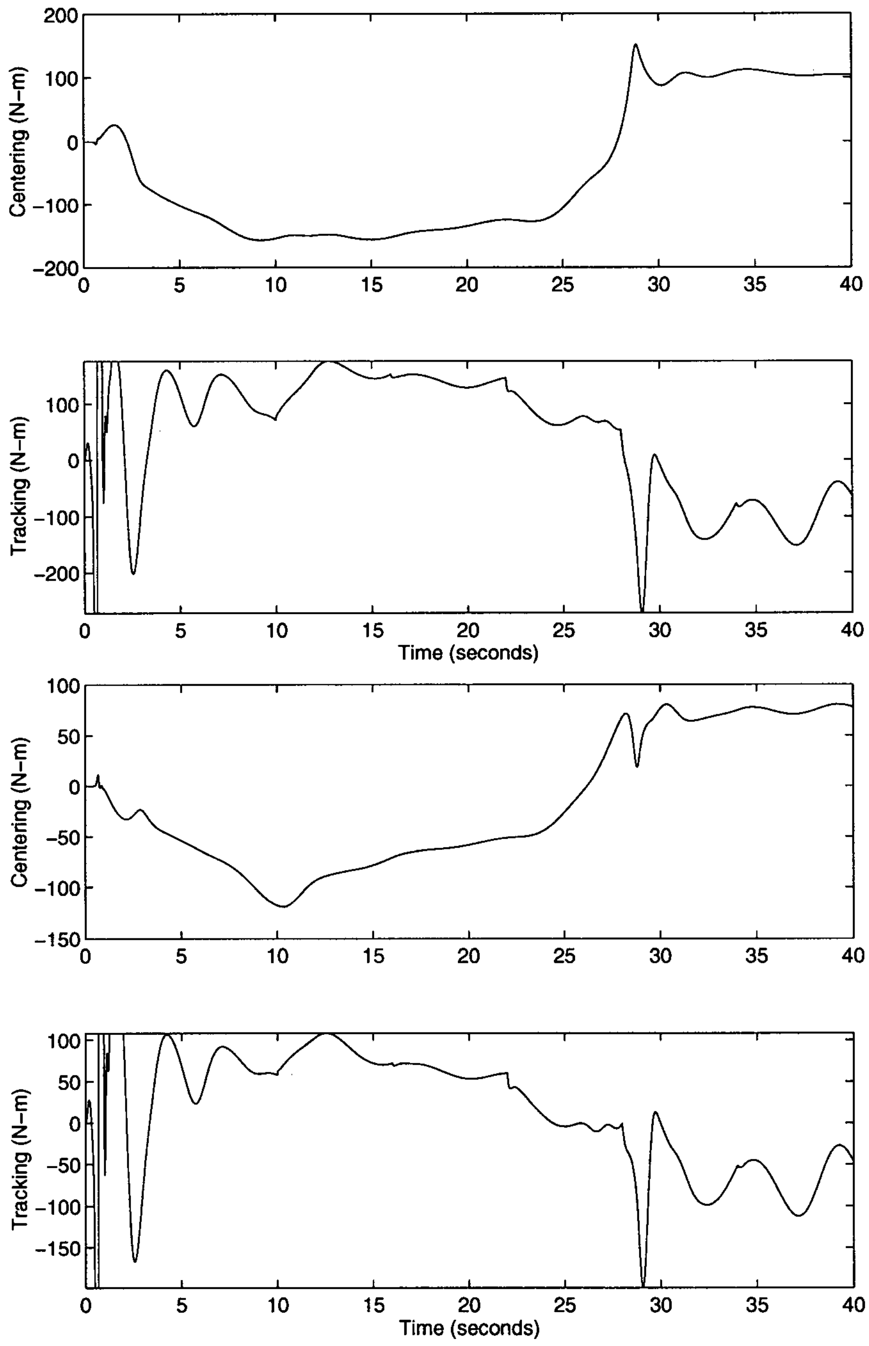


Figure 7.54: Centering and tracking torques for links 1(top) and 2 (bottom) of the reference manipulator tracking the reference trajectory.

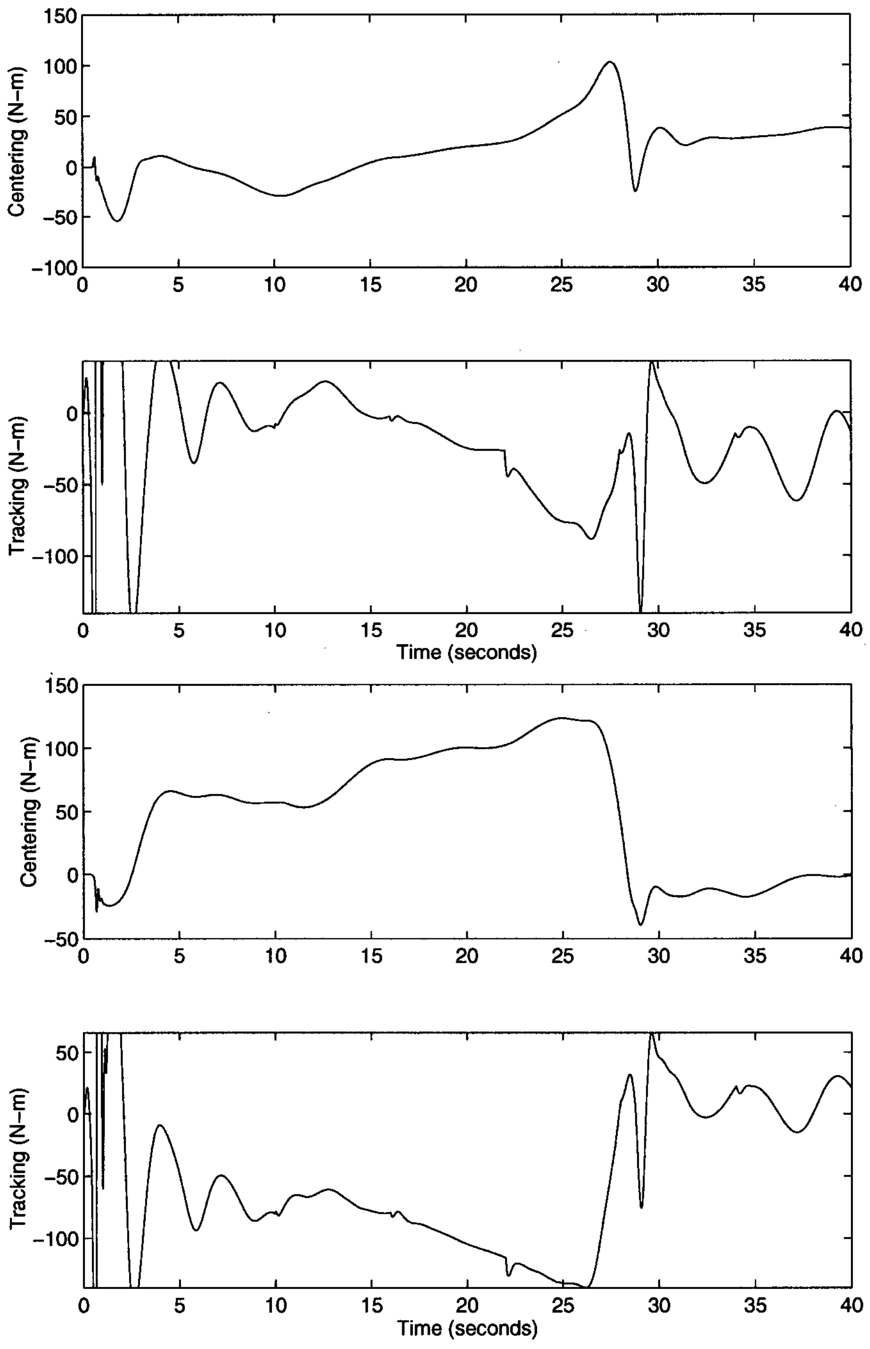


Figure 7.55: Centering and tracking torques for links 3 (top) and 4 (bottom) of the reference manipulator tracking the reference trajectory.

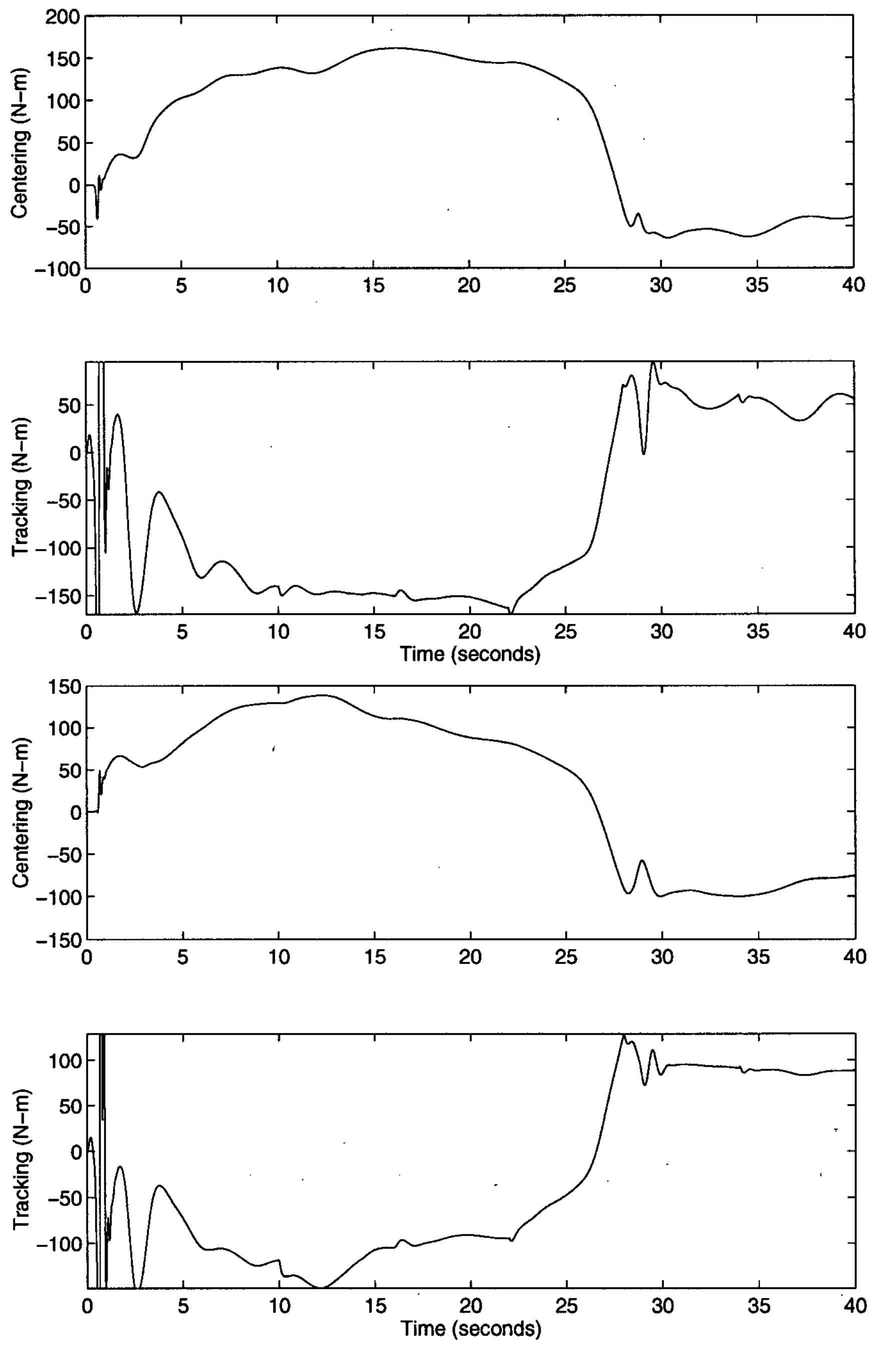


Figure 7.56: Centering and tracking torques for links 5 (top) and 6 (bottom) of the reference manipulator tracking the reference trajectory.

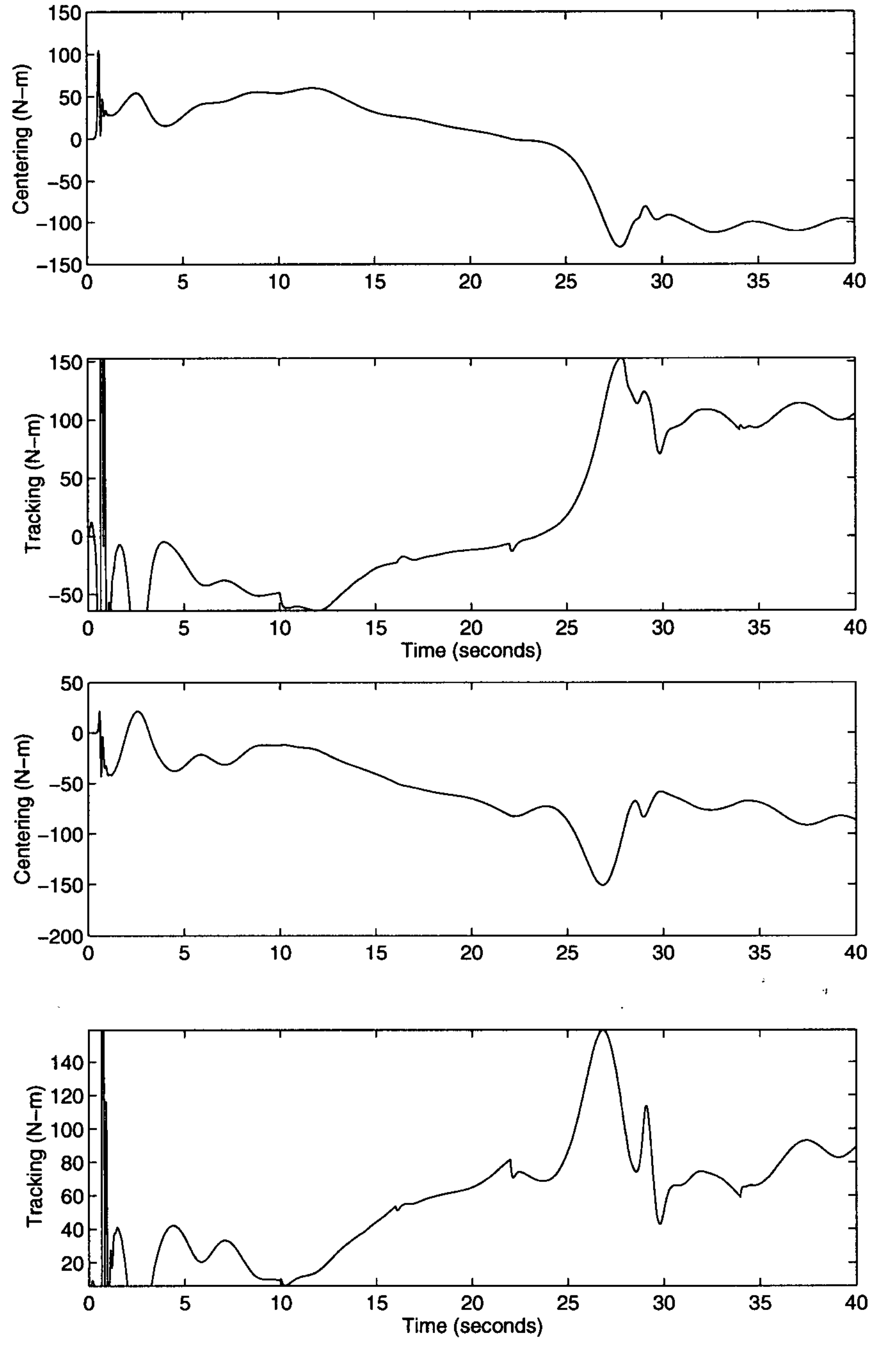


Figure 7.57: Centering and tracking torques for links 7 (top) and 8 (bottom) of the reference manipulator tracking the reference trajectory.

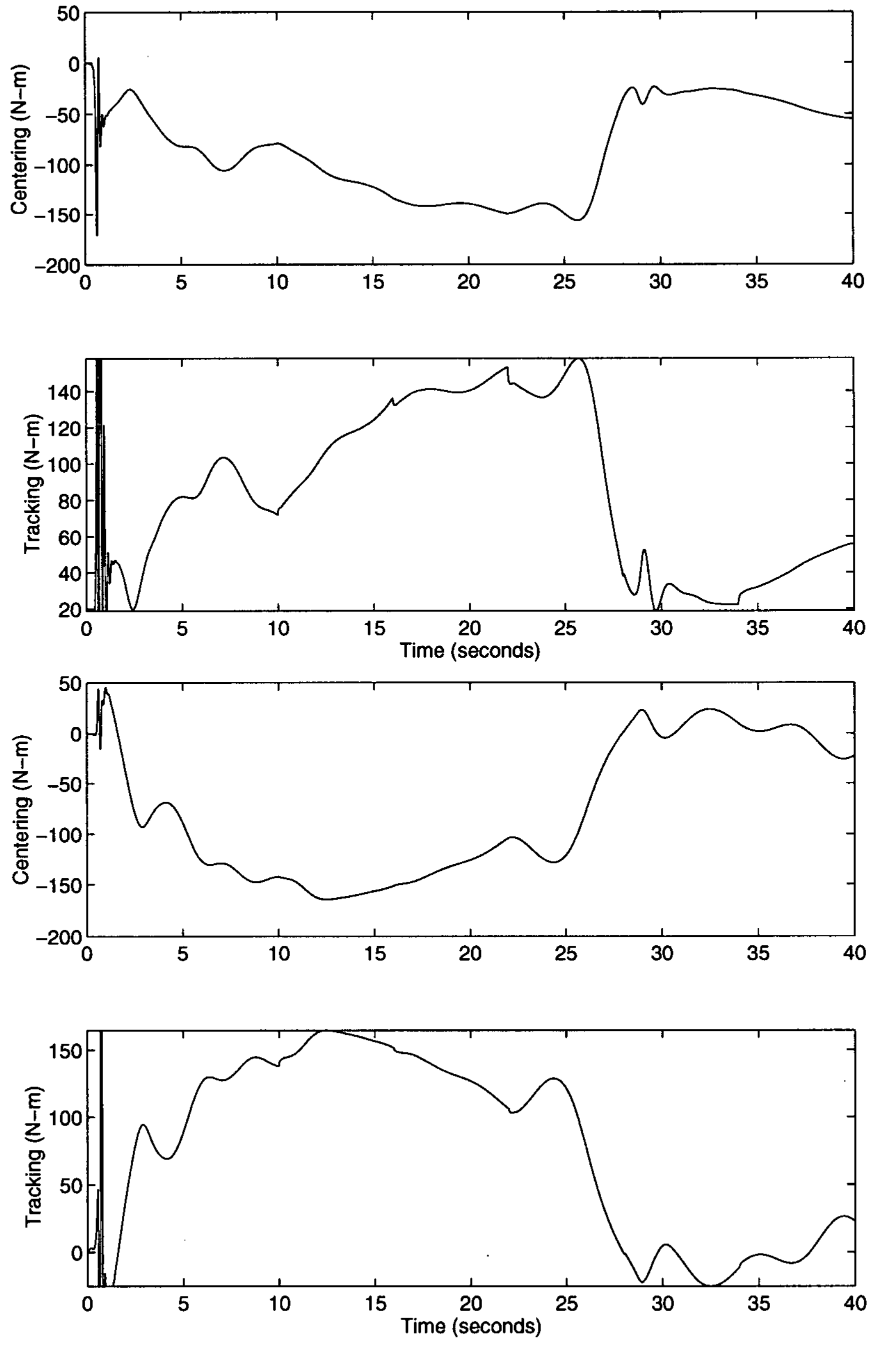


Figure 7.58: Centering and tracking torques for links 9 (top) and 10 (bottom) of the reference manipulator tracking the reference trajectory.

towards the edge of the work volume and, by coincidence, closer to kinematic singularities.

Qualitatively, these effects can be reversed by applying a retraction behaviour to the entire manipulation system. Though one might be able to design a global goal to implement a global retraction generator, a simple local strategy can produce a retraction behaviour. For a serial planar manipulator, retraction of the end effector to the origin may be easily affected by *alternately* applying maximum and minimum boundary setpoints along the length of the manipulator. Formally:

$$q_{d_k} = \begin{cases} q_{\text{high}_k} & \text{if } k \text{ even} \\ q_{\text{low}_k} & \text{if } k \text{ odd} \end{cases} \quad \forall k \tag{7.311}$$

again:

$$\mathbf{u} = k_q q_{e_k} + k_w \dot{q}_{e_k} \tag{7.312}$$

where $q_{e_k} = q_{d_k} - q_k$. The agent's control effort:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1] \begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{retraction}} \end{bmatrix} \tag{7.313}$$

Adopting the local centering behaviour gains and setpoints in table 7.14 in the multiagent control of the reference manipulator, one can observe that the global shaping behaviour is markedly distinct from previous centering strategies, though the controller dynamics are identical. This strategy is a good example of local behaviour leading to complex aggregate or global behaviour and the simplicity of deriving such behaviour. The alternate extreme setpoint strategy serves to contract the manipulator in a manner reminiscent of a coil spring.

The results in table 7.15 and figure 7.60 document improved trajectory tracking and, in figure 7.59, a relatively compact manipulator volume. Trajectory tracking is improved in part due to end effector rotation that conveniently coincides with the desired orientation trajectory. Furthermore, as all the links rotate in the first instants of motion, the manipulator spontaneously leaves the region of kinematic singularity (where the Jacobian Transpose performs poorly) improving both the projection of the end effector goal on each actuator and the image of the robot's dynamics to the end effector goal generator.

Tertiary dynamics, also residing in $\mathbf{N}(\mathbf{J})$, differ between centering and retraction behaviours. Since PD local behaviour controllers resemble a spring damper system, some form of tertiary oscillatory dynamics should be expected. Not surprisingly, the centering behaviour, modelling a leaf spring, tends to exhibit transverse oscillations more readily than the retraction behaviour that tends to exhibit longitudinal oscillations.

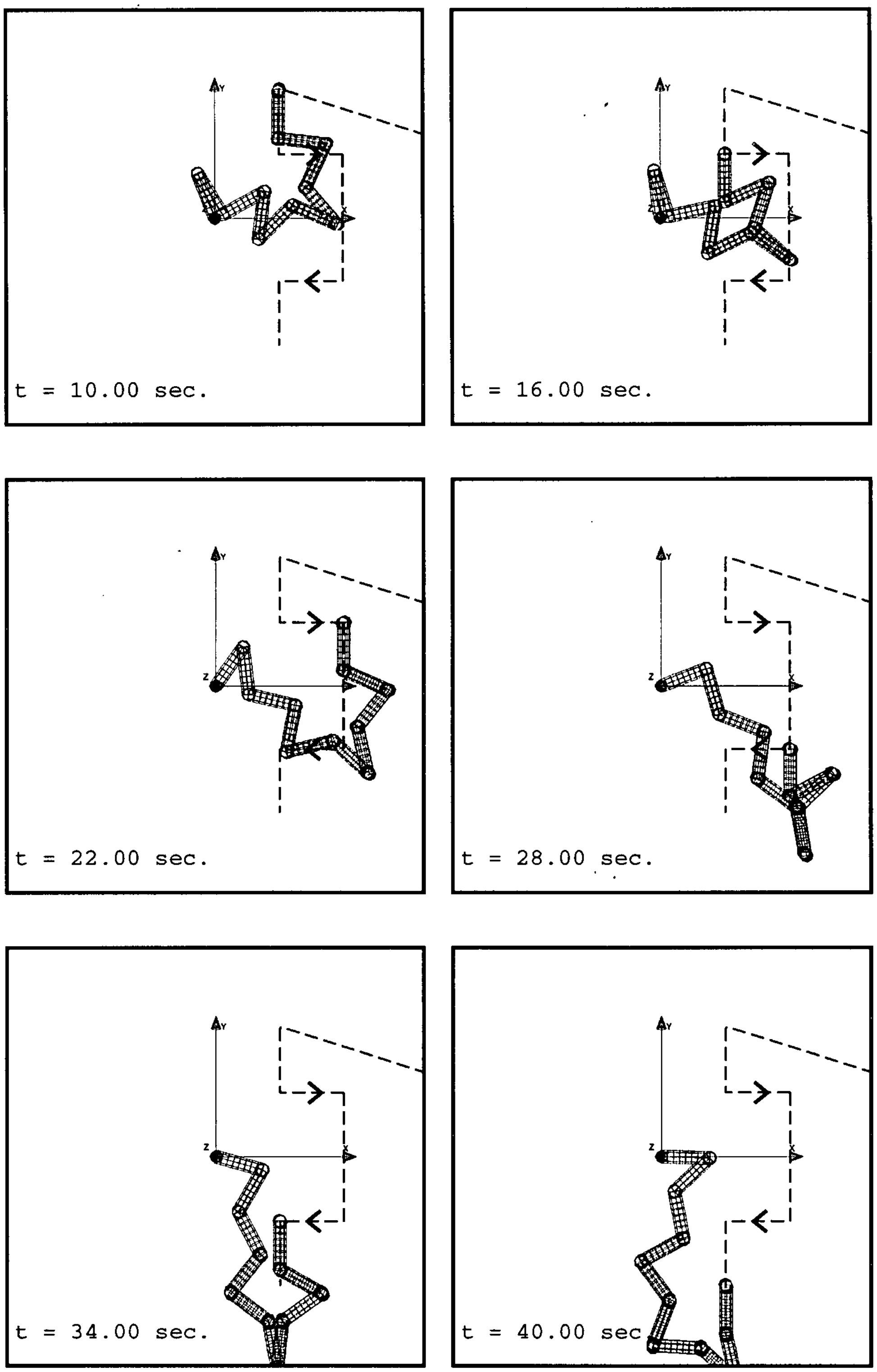


Figure 7.59: Both RMDAC end effector global and joint centering local goals with alternating setpoints are engaged.

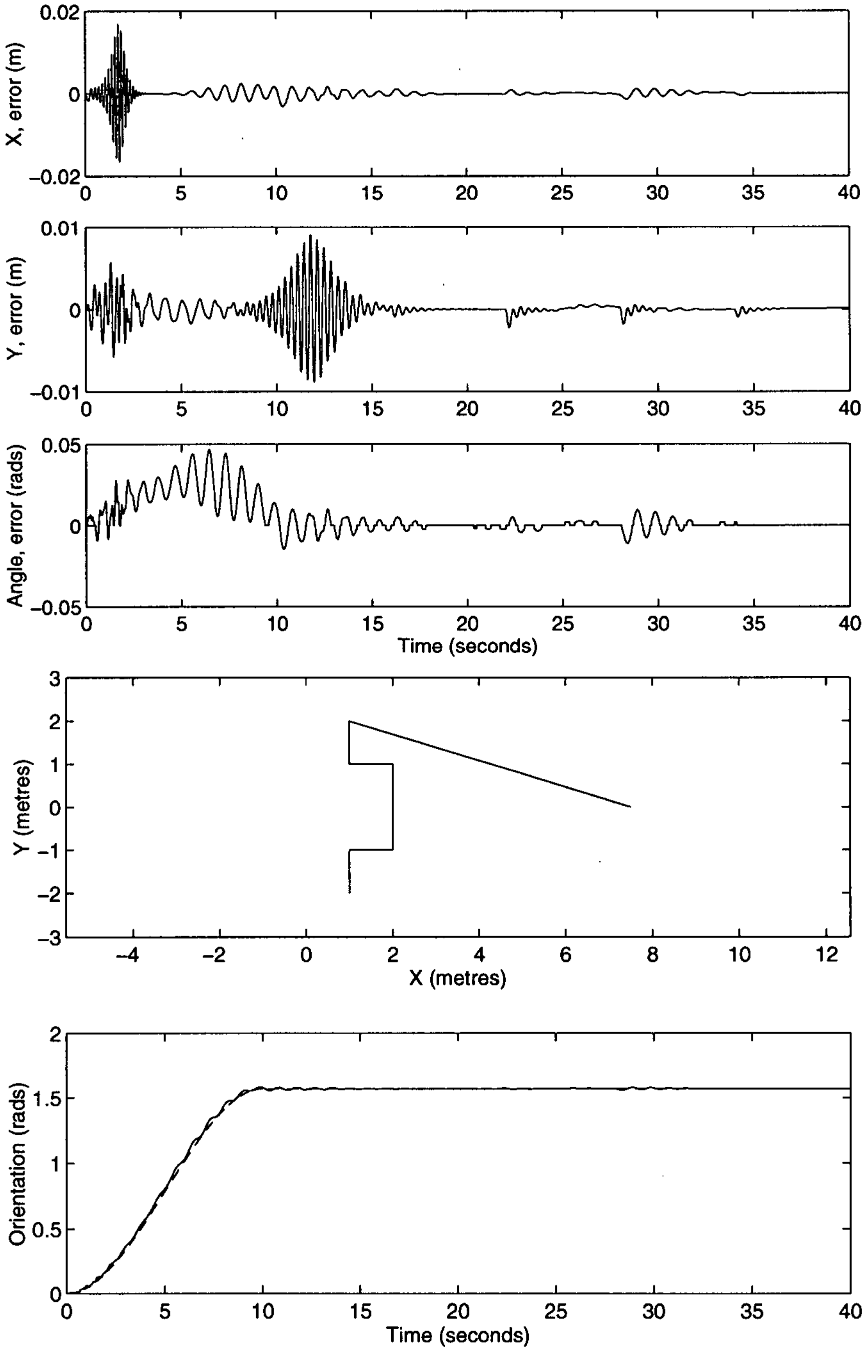


Figure 7.60: End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering 'retraction' local goals.

Agent	k_q	k_v	q_d	$\dot{q}_d$
01	1.0e+02	2.0e+01	+1.570e+00	0.00e+00
02	1.0e+02	2.0e+01	-1.570e+00	0.00e+00
03	1.0e+02	2.0e+01	+1.570e+00	0.00e+00
04	1.0e+02	2.0e+01	-1.570e+00	0.00e+00
05	1.0e+02	2.0e+01	+1.570e+00	0.00e+00
06	1.0e+02	2.0e+01	-1.570e+00	0.00e+00
07	1.0e+02	2.0e+01	+1.570e+00	0.00e+00
08	1.0e+02	2.0e+01	-1.570e+00	0.00e+00
09	1.0e+02	2.0e+01	+1.570e+00	0.00e+00
10	1.0e+02	2.0e+01	-1.570e+00	0.00e+00

Table 7.14: Tabulation of joint centering PD gains and setpoint for the constant compliance retraction local goal strategy.

Strategy	$\mathbf{e}_{\text{rms}}$	θ_{rms}
Retraction	2.244e-03	5.782e-03

Table 7.15: RMS error for the constant compliance retraction strategy.

7.6.4 Variable Compliance Centering

Another global shaping behaviour strategy is to vary the stiffness of each local PD goal generator. By differentiating agents through a stiffness strategy, an agent with a relatively soft PD controller will accept more displacement than stiffer controllers, giving some links 'preferential' treatment over others. For example, consider the following compliance strategy based on the following PD controller stiffness selection:

$$k_{q_j} > k_{q_{j-1}} \quad k = 1 \dots n \tag{7.314}$$

With this strategy, a 'variable compliance' multiagent system can be designed to becomes stiffer towards the end effector. The reverse of this strategy:

$$k_{q_j} < k_{q_{j-1}} \quad k = 1 \dots n \tag{7.315}$$

allows a 'variable compliance' multiagent system can be designed that becomes more flexible towards the end effector. For consistency both cases are are designed to be LHP stable and perfectly damped or:

$$k_{w_j} = 2\sqrt{k_{q_j}} \tag{7.316}$$

and the agent output is again:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1] \begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{centering}} \end{bmatrix} \tag{7.317}$$

Simulating the reference manipulator with the increasing gains documented in table 7.16 the familiar global 'leaf spring' behaviour emerges. However, the manipulator exhibits progressively smaller deflections from joint midrange in figure 7.62 from base to end effector. Comparing figure 7.61 with figure 6.30 it is clear that the addition of this behaviour tends to exaggerate transient end effector tracking errors, though steady state response is ultimately unaffected.

Similarly simulating the reverse strategy in figure 7.64 shows that decreasing gains from base to end effector permits progressively greater deflections at each link along the 'leaf spring', again magnifying the end effector transient response in figure 6.30. It is interesting to note the difference in transient response between increasing and decreasing gains cases. Clearly, the increasing gain strategy disrupts the end effector global goal less than the reversed, decreasing strategy. Given the Jacobian relationship between end effector and joint velocities, this should not be unexpected. Since the base gets 'preferential' treatment in the decreasing case (i.e. small deflections are desirable) with relatively large stiffnesses, restoration accelerations in the first link will be magnified at the end effector through equation (2.12). In the increasing gains case it is the end effector that retains relatively stiff gains and, as a consequence, generates smaller end effector disturbances.

Despite the performance differences, a comparison of the figures 7.61,7.63 and 7.52 indicates that reducing *total* stiffness of these variable series goal systems improves transient performance.

Remarks

To dynamicists, the observation that subtle, *structural* changes can lead to large changes in the response of a nonlinear system is not surprising. To the multiagent designer, however, this lesson shows that seemingly minor changes within a multiagent team can have significant impact on the system's global behaviour.

7.6.5 Combinations: Trajectory Tracking, Obstacle Avoidance, and Centering

Trajectory tracking and obstacle avoidance guarantee collision free motion if the tracking goal is reachable and the end effector trajectory is obstacle free. However, triggered obstacle avoidance tasks constrain the manipulator only while activated and leave the manipulator to free motion in $\mathbf{N}(\mathbf{J})$ otherwise. Maximizing

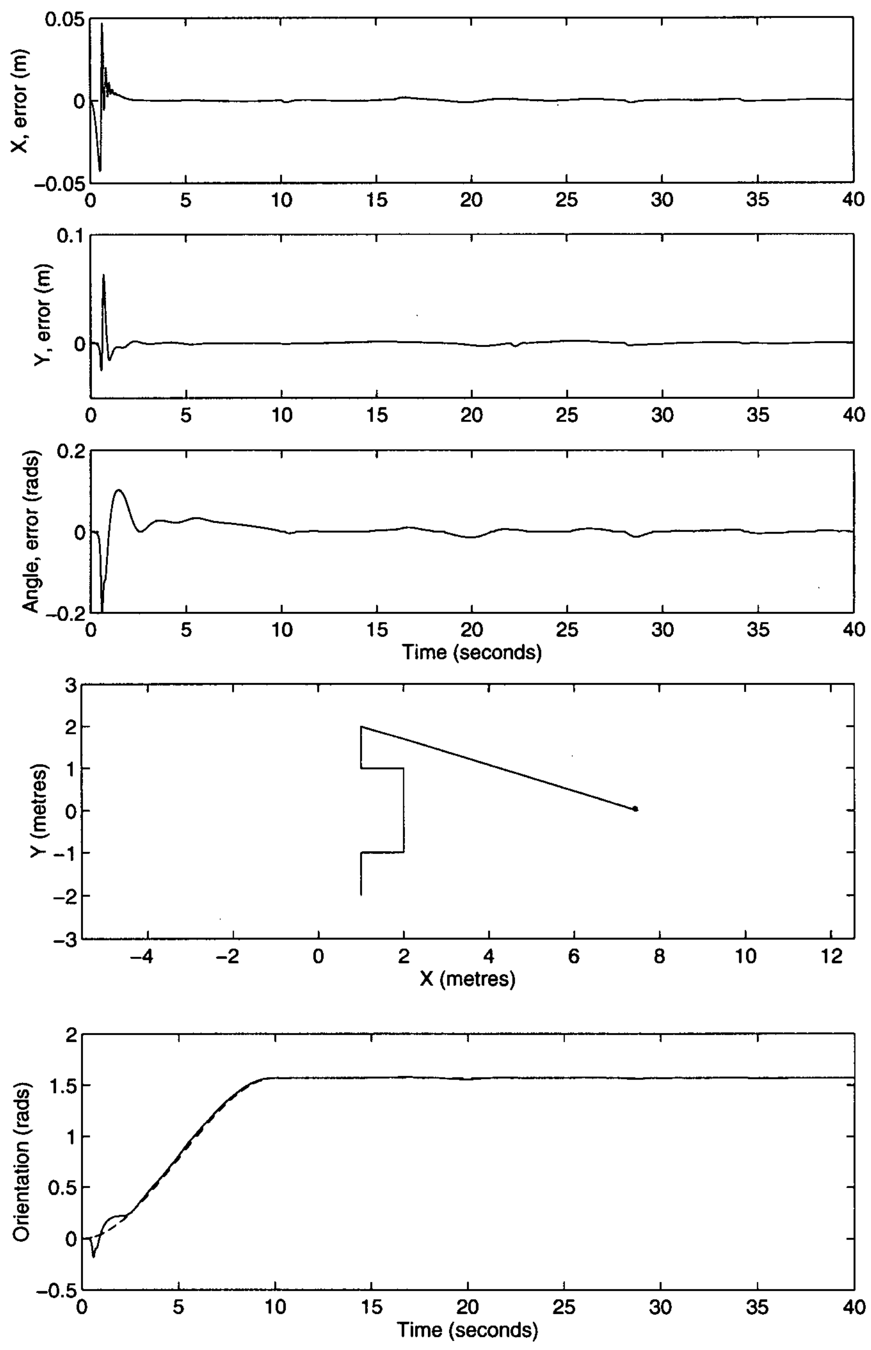


Figure 7.61: End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering local goals of linearly *increasing* stiffness.

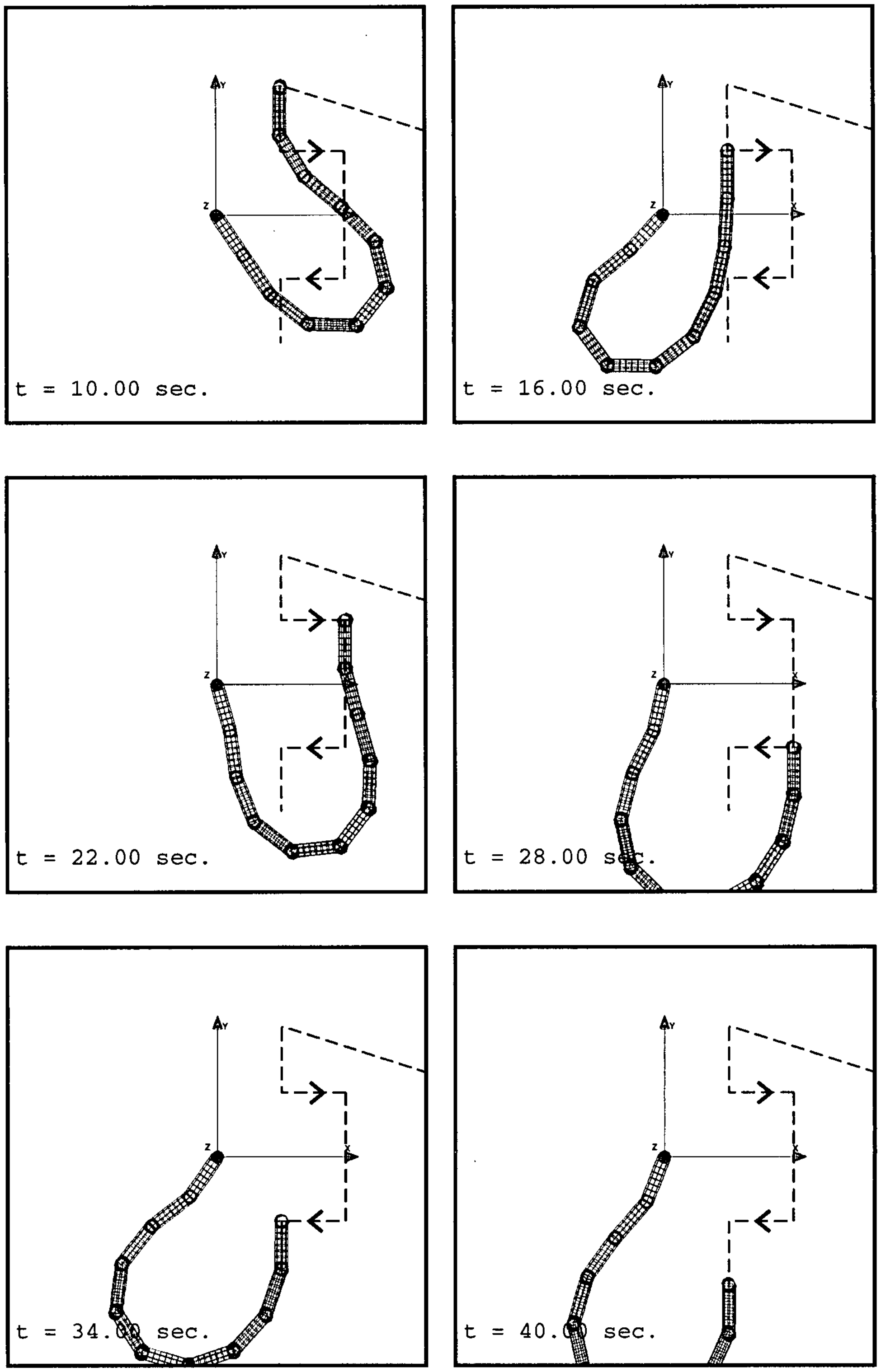


Figure 7.62: Both RMDAC end effector global and joint centering local goals with increasing stiffness are engaged.

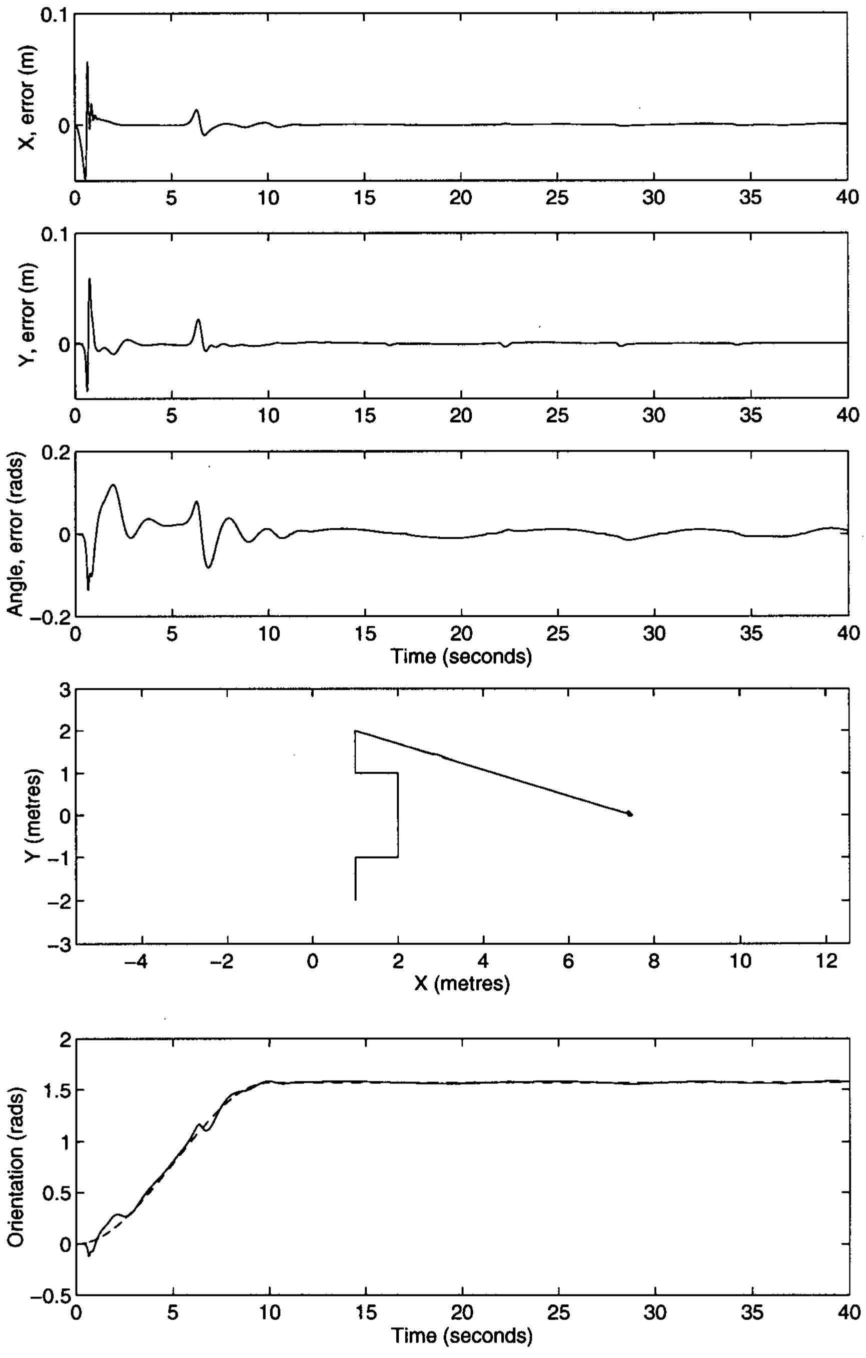


Figure 7.63: End effector trajectory performance for the reference manipulator and payload under RMDAC end effector and joint centering local goals of linearly *decreasing* stiffness.

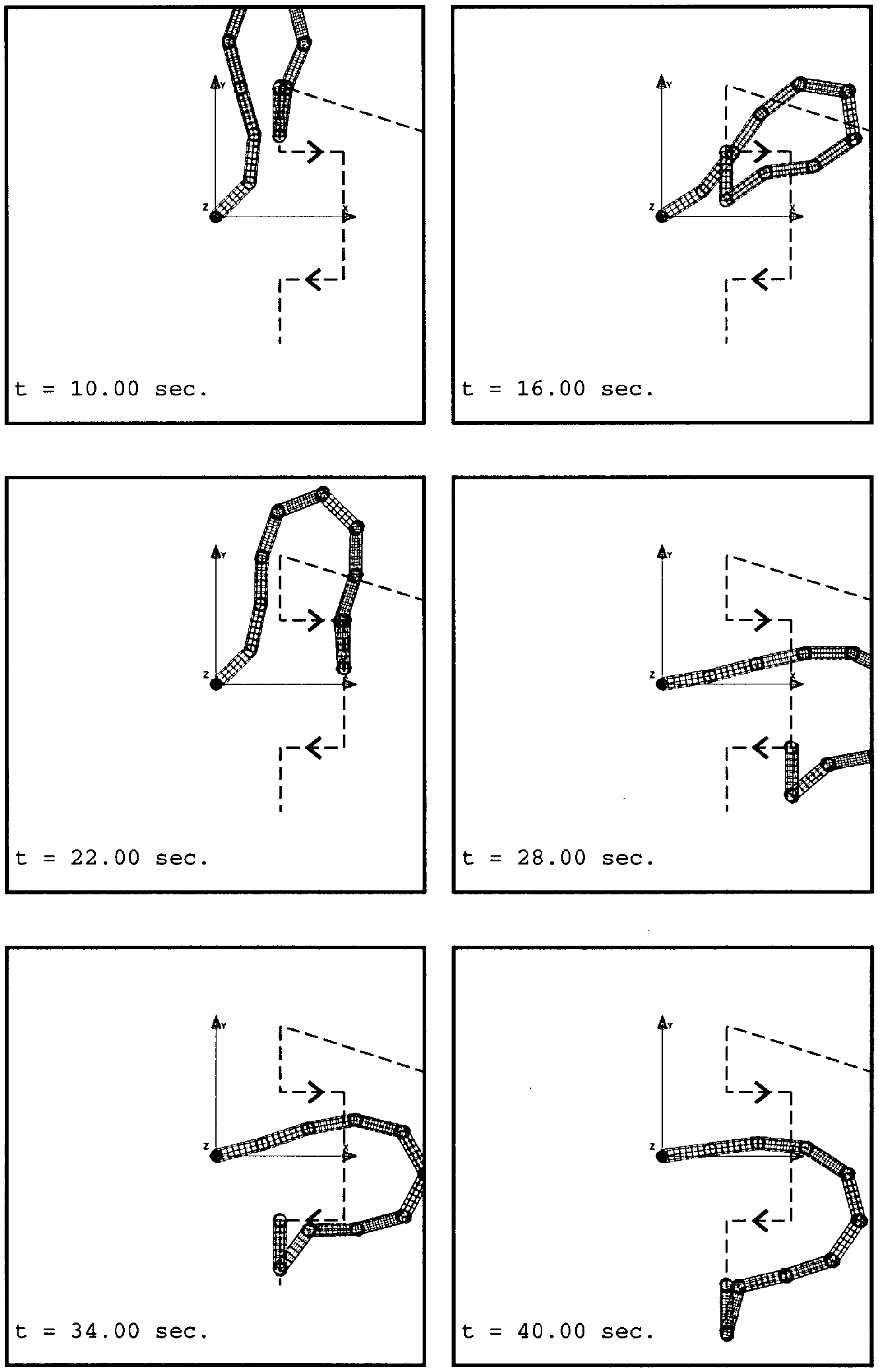


Figure 7.64: Both RMDAC end effector global and joint centering local goals with decreasing stiffness are engaged.

Agent	Decreasing		Increasing	
	k_q	k_v	k_q	k_v
01	1.0e+02	2.0e+01	1.0e+01	6.324e+00
02	9.0e+01	1.897e+01	2.0e+01	8.944e+00
03	8.0e+01	1.788e+01	3.0e+01	1.095e+01
04	7.0e+01	1.673e+01	4.0e+01	1.265e+01
05	6.0e+01	1.549e+01	5.0e+01	1.414e+01
06	5.0e+01	1.414e+01	6.0e+01	1.549e+01
07	4.0e+01	1.265e+01	7.0e+01	1.673e+01
08	3.0e+01	1.095e+01	8.0e+01	1.788e+01
09	2.0e+01	8.944e+00	9.0e+01	1.897e+01
10	1.0e+01	6.324e+00	1.0e+02	2.0e+01

Table 7.16: Tabulation of *increasing* and *decreasing* joint centering PD gains for the variable compliance local goal strategy.

Strategy	$\mathbf{e}_{\text{rms}}$	θ_{rms}
Increasing	4.972e-03	1.082e-02
Decreasing	6.045e-03	1.238e-02

Table 7.17: RMS error for "increasing" and "decreasing" variable centering gain strategies.

the maneuvering volume *and* avoiding obstacles frees the manipulator from conflicts with both configuration and cartesian space boundaries.

In the following demonstration, previewed in the introduction, three behaviours act simultaneously within each agent: an RMDAC trajectory tracking global goal, an RMOA obstacle avoidance global goal, and a PD joint centering local goal. The jth agent of the form:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1 \;\; 1] \begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{avoid}} \\ \mathbf{u}_{\text{centering}} \end{bmatrix} \tag{7.318}$$

Given that $\mathbf{u}_{\text{avoid}}$ is a triggered behaviour controller, the arbitration vector could be rewritten as:

$$\mathbf{k}_j = \begin{bmatrix} k_{\text{track}} \\ k_{\text{avoid}} \\ k_{\text{centering}} \end{bmatrix} \tag{7.319}$$

$$k_{\text{track}} = 1 \tag{7.320}$$

$$k_{\text{avoid}} = \begin{cases} 1 & \text{if } |\mathbf{x}_r| \le c \\ 0 & \text{if } |\mathbf{x}_r| > c \end{cases} \tag{7.321}$$

Behaviour Controllers	$\mathbf{k}_{\mathrm{A}_i}$	$\mathbf{e}_{\mathrm{rms}}$	θ_{rms}	$\mathbf{q}_{\mathrm{rms}}$
Tracking	[1.0]	2.779e-03	2.850e-02	7.848e+00
Tracking, Avoidance	[1.0 1.0]	2.537e-03	5.475e-03	9.904e+00
Tracking, Centering	[1.0 1.0]	5.963e-03	1.424e-02	2.700e+00
Tracking, Avoidance, Centering	[1.0 1.0 1.0]	4.679e-03	7.626e-03	2.239e+00
Tracking, Avoidance, Centering	[1.0 1.0 0.5]	3.999e-03	8.046e-03	2.292e+00

Table 7.18: Comparison of RMS errors for a tracking, obstacle avoidance ($\eta = 10.0$), and centering ($k_p = 100$ $k_d = 20$) combined strategies.

$$k_{\text{centering}} = 1 \tag{7.322}$$

where $|\mathbf{x}_r|$ and c are described are the surface and clearance ranges respectively.

Selecting the avoidance gain, $\eta = 10.0$, and centering gains uniformly as $k_q = 100.0$ and $k_w = 20.0$, a multiagent simulation can demonstrate the combined interaction between these multiple goal systems.

The resulting end effector performance depicted in figure 7.65 demonstrates that, as in previous centering tasks, the adaptive controller's performance does not degrade significantly. Indeed, the combined strategy shows marked improvement over the tracking strategy alone. When triggered, the obstacle avoidance behaviour stabilizes local oscillations and, as a result, improves RMS trajectory tracking.

The individual agent torque contributions shown in figures 7.67 to 7.71 clearly demonstrate the interaction and arbitration between behaviours as a function of end effector trajectory, link displacement, and range to obstacle surface. Again all local goal activity resides in $\mathbf{N}(\mathbf{J})$ through the strict enforcement of the trajectory tracking task.

As mentioned earlier a similar mixture of tasks was demonstrated by [Slotine, 1988] based on an offline version of RMRC. Using a null space selection operator, joint centering and obstacle avoidance were demonstrated in conjunction with a trajectory tracking task. The control law used in [Slotine, 1988]:

$$\dot{\mathbf{q}} = -\alpha \mathbf{S}\mathbf{J}^T\mathbf{K}_p\mathbf{x} - \beta(\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})(\mathbf{K}\mathbf{q} + \mathbf{K}_{\text{obs}}\sum_{j=1}^{j=\eta}\mathbf{J}_j^T\sum_{i=1}^{i=\kappa}(\frac{\mathbf{x}_{ij}}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}})) \tag{7.323}$$

where η and κ are the number of joints and obstacles respectively. Clearly, Slotine used similar, centralized, controllers. In multiagent control, the adoption of the global goal proxy enables the distribution of these controllers over the manipulator, while the adaptive global goal engages self organization in the Jacobian null space.

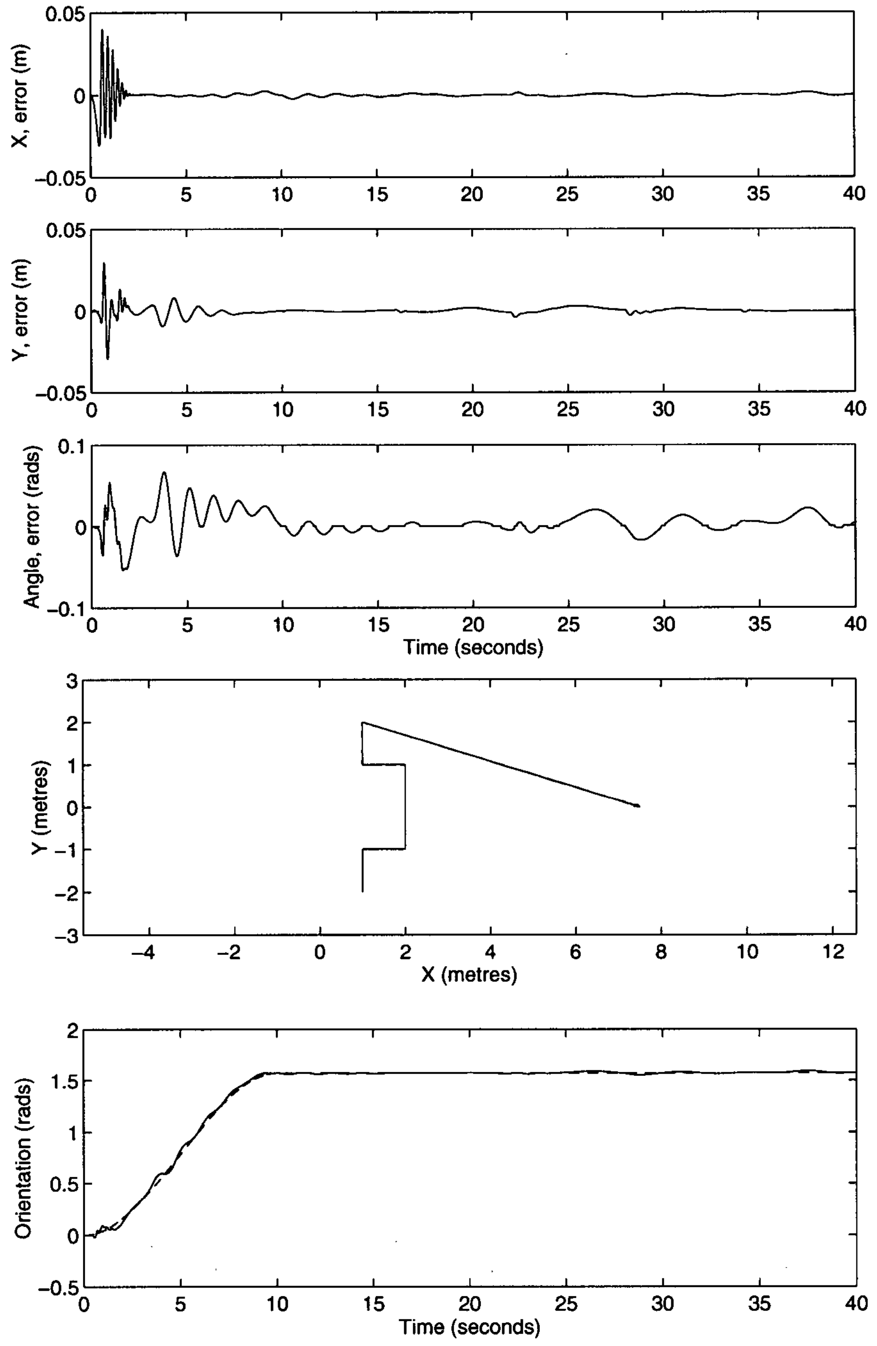


Figure 7.65: End effector trajectory error for the reference manipulator and payload under both RMDAC end effector global, RMOA obstacle avoidance, and joint centering local goals.

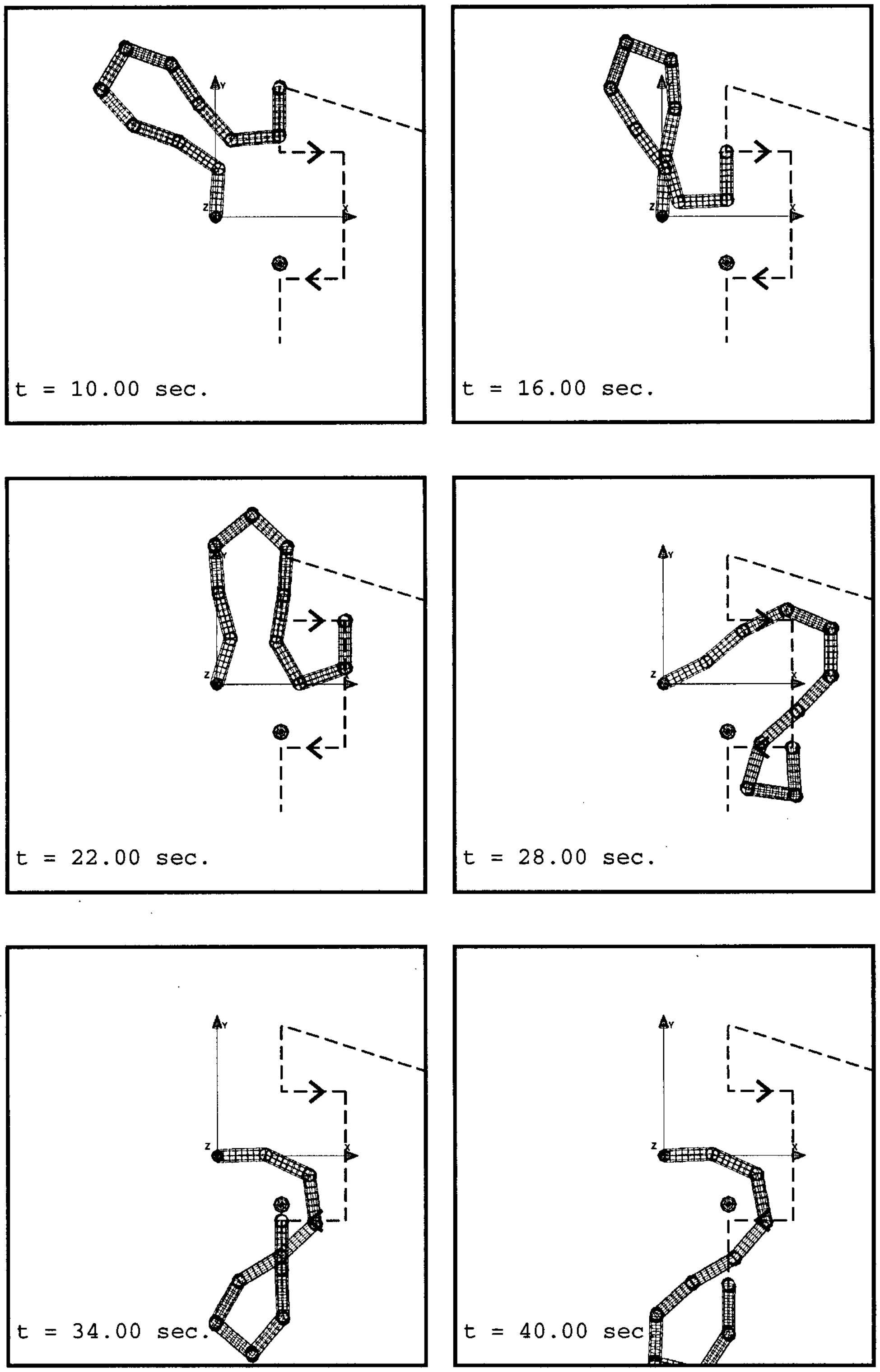
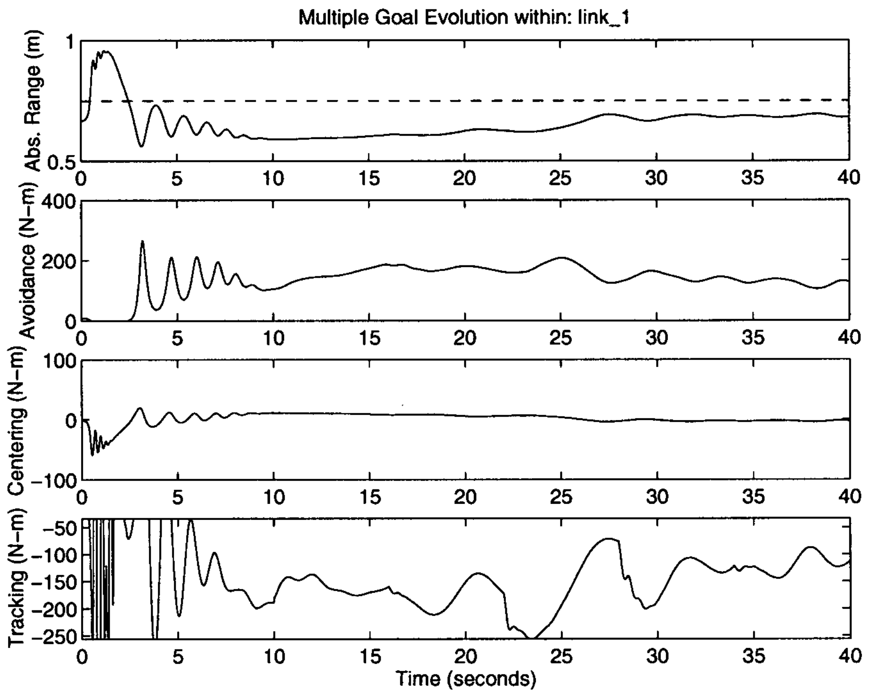


Figure 7.66: Reference manipulator configurations at reference trajectory knotpoints under RMDAC end effector and obstacle avoidance global goals ($\eta = 10.0$) and joint centering local goals($k_q = 100$, $k_w = 20$). Note 'leaf spring' configuration and avoidance reside in $\mathbf{N}(\mathbf{J})$.



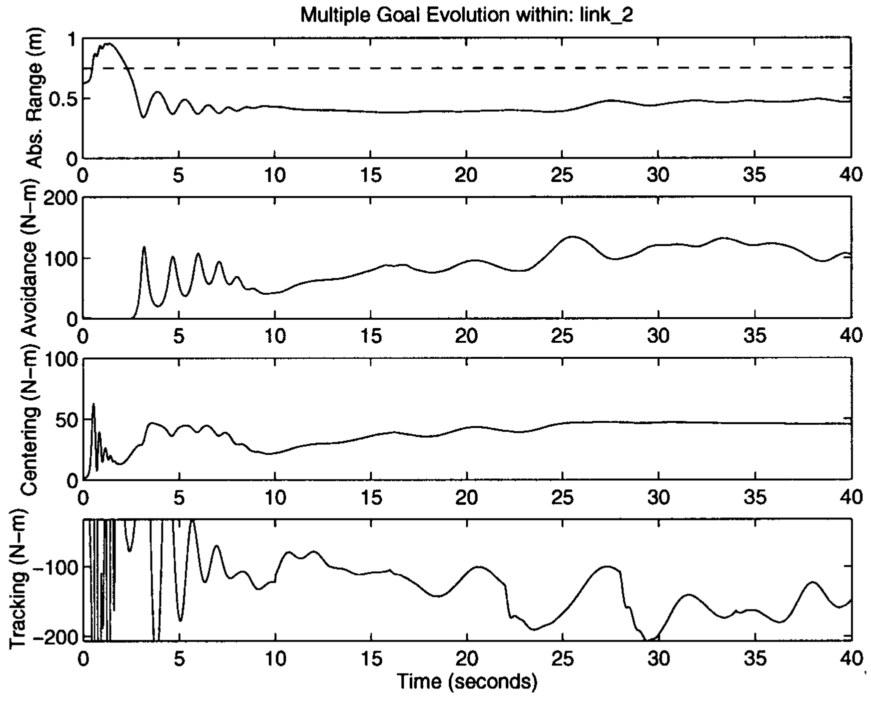
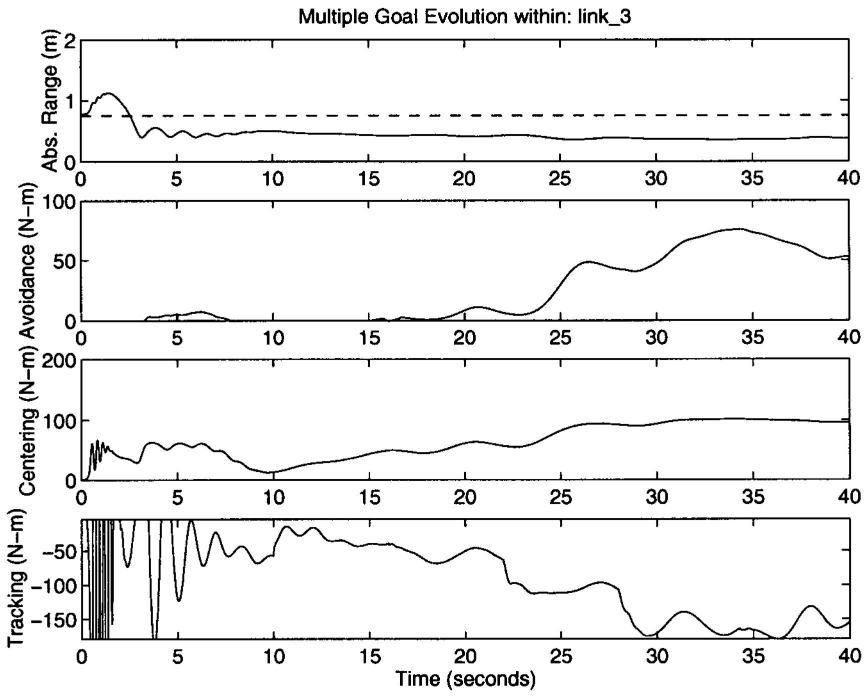


Figure 7.67: Range to surface, avoidance, centering, and tracking torques for links 1(top) and 2 (bottom) of the reference manipulator tracking the reference trajectory.



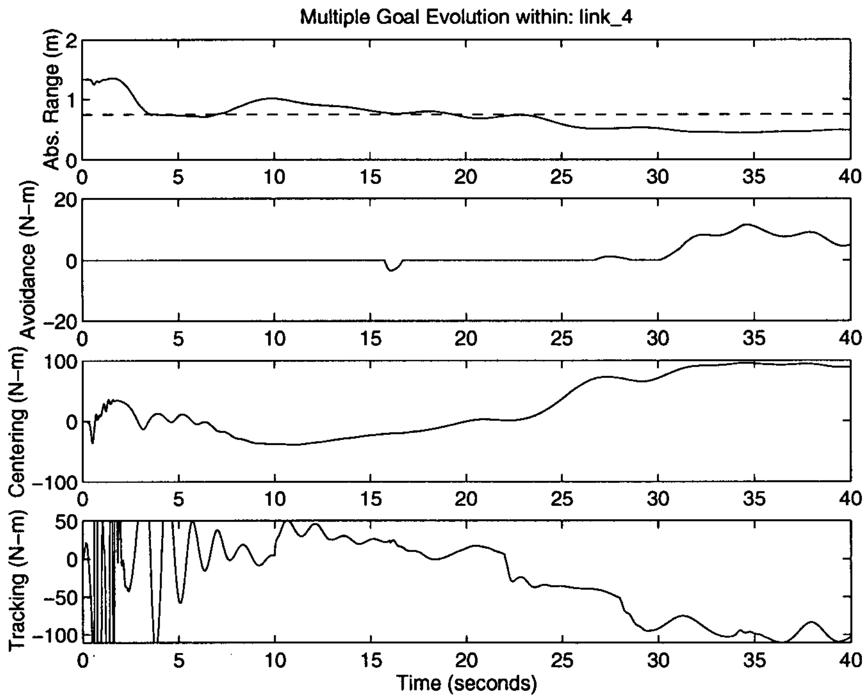


Figure 7.68: Range to surface, avoidance, centering, and tracking torques for links 3 (top) and 4 (bottom) of the reference manipulator tracking the reference trajectory.

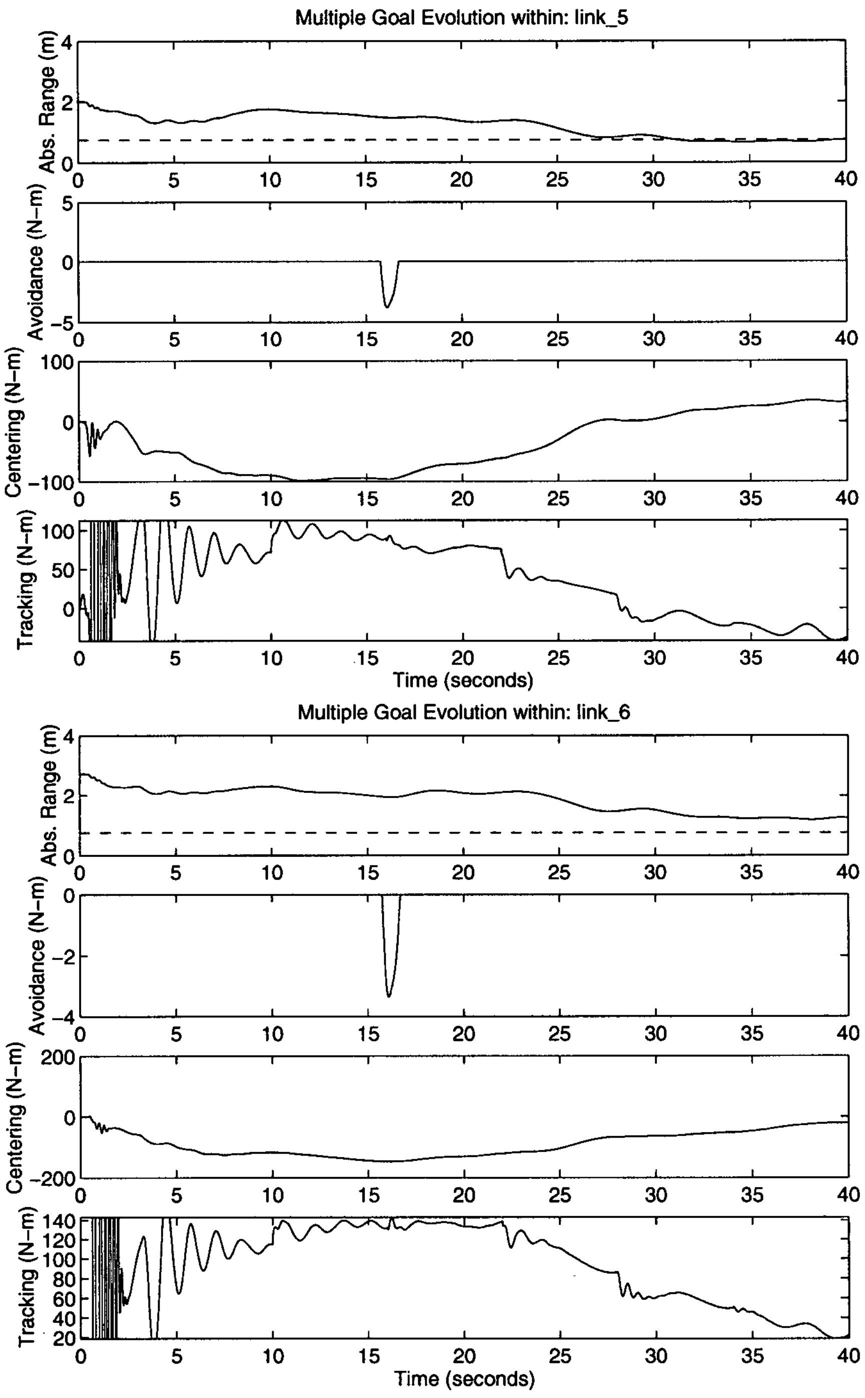
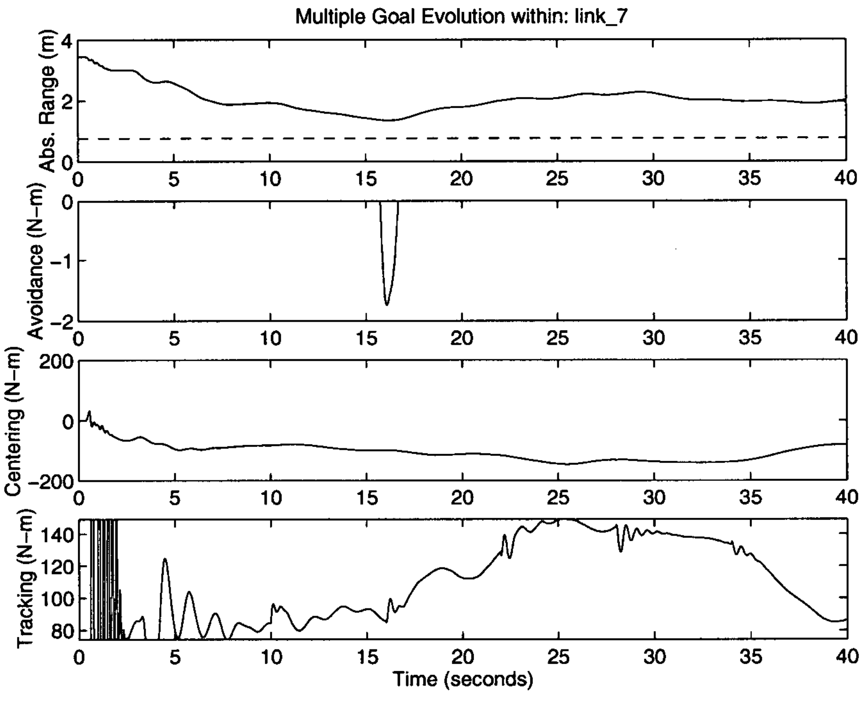


Figure 7.69: Range to surface, avoidance, centering, and tracking torques for links 5 (top) and 6 (bottom) of the reference manipulator tracking the reference trajectory.



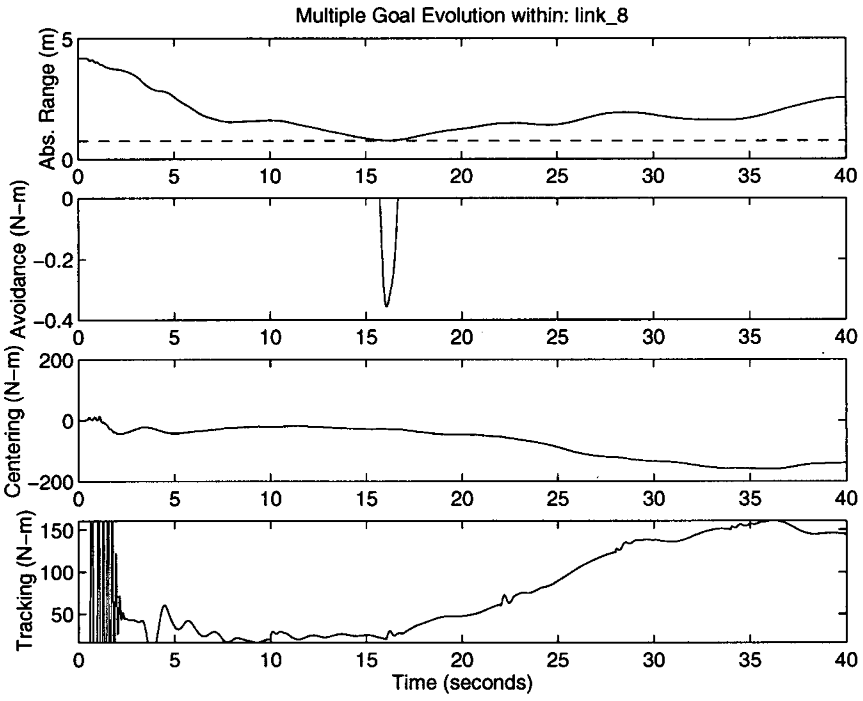


Figure 7.70: Range to surface, avoidance, centering, and tracking torques for links 7 (top) and 8 (bottom) of the reference manipulator tracking the reference trajectory.

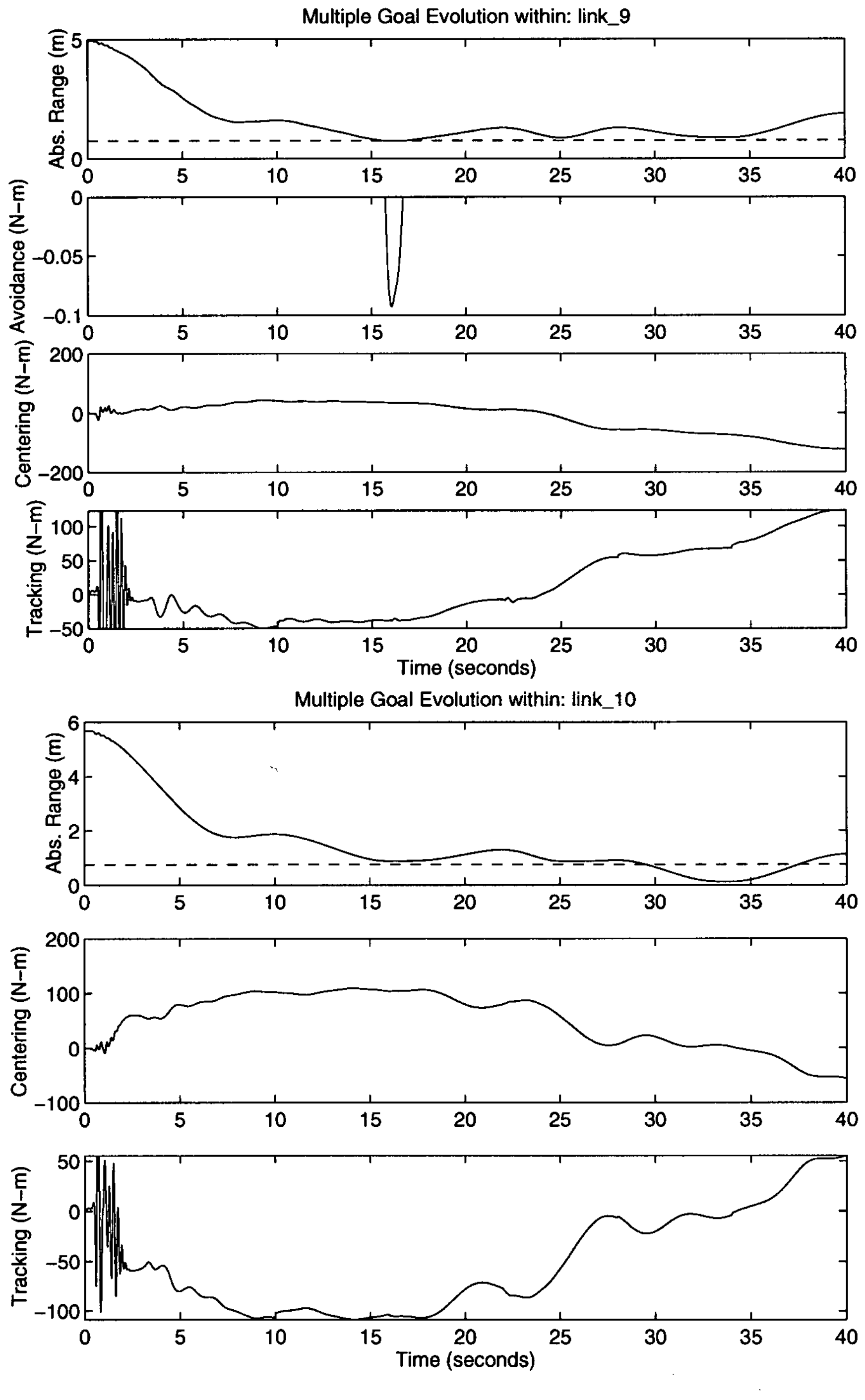


Figure 7.71: Range to surface, avoidance, centering, and tracking torques for links 9 (top) and 10 (bottom) of the reference manipulator tracking the reference trajectory.

7.7 Arbitration and Compliance

In keeping the elements of the linear agent model's arbitration gain vector, $\mathbf{k}_j$, unity, arbitration has been simplified to an unmoderated equilibrium between goal systems. However, in the previous experiments it was observed that changing the gains within the local PD goal system significantly altered the performance of the system. Clearly arbitration and compliance are related. If this is so, what is the impact of changing the relative magnitudes of the arbitration vector elements? To explore this further, consider the local centering goal system in isolation.

$$\mathbf{u}_{\text{centering}} = k_q q_{c_k} + k_w \dot{q}_{c_k} \tag{7.324}$$

Within the linear agent model, the output of the agent is defined as:

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}^T \mathbf{u}_{\mathsf{A}_j} = [k_{\text{track}} \;\; k_{\text{centering}}] \begin{bmatrix} \mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{centering}} \end{bmatrix} \tag{7.325}$$

Now suppose this expression was rewritten, defining new behaviour controllers, $\mathbf{u}'_{\text{track}}$ and $\mathbf{u}'_{\text{track}}$ in which the arbitration gains have been incorporated:

$$\mathbf{u}_{cj} = [1 \;\; 1] \begin{bmatrix} \mathbf{u}'_{\text{track}} \\ \mathbf{u}'_{\text{centering}} \end{bmatrix} \tag{7.326}$$

Where, for example, the centering behaviour becomes:

$$\mathbf{u}'_{\text{centering}} = k'_q q_{c_k} + k'_w \dot{q}_{c_k} \tag{7.327}$$

where

$$k'_q = k_{\text{centering}} k_q \quad k'_w = k_{\text{centering}} k_w \tag{7.328}$$

Reexamining the model of the PD controller with gains k'_q and k'_w the Laplace transform, $Q_{m_k}(s)$, becomes:

$$\begin{aligned} Q_{m_k}(s) &= \left[\frac{K_k k_{\text{centering}} k_q + K_k k_{\text{centering}} k_w s}{\Delta_k(s)}\right] Q_{d_k}(s) - r_k \frac{D_k(s)}{\Delta_k(s)} && (7.329) \\ \Delta_k(s) &= J_{\text{eff}_k} s^2 + (B_{\text{eff}_k} + K_k K_b + K_k k_{\text{centering}} k_w)s + K_k k_{\text{centering}} k_q && (7.330) \end{aligned}$$

If we assume, for argument, that B_{eff_k} and K_b are negligible and K_k is unity, then the characteristic reduces to:

$$\Delta_k(s) = J_{\text{eff}_k} s^2 + k_{\text{centering}} k_w s + k_{\text{centering}} k_q \tag{7.331}$$

It is apparent that arbitration gains other than unity can affect the performance properties of the simple PD controller. In figure 7.72 and table 7.18 this effect is demonstrated using identical behaviour controllers to the obstacle avoidance, centering behaviour example (i.e. $k_q = 100$, $k_w = 20$, $\mathbf{K}_p = \text{diag}(100)$, $eta = 10.0$ and $\mathbf{K}_w = \text{diag}(20)$) and with arbitration gains set to $k_{\text{centering}} = 0.5$ and $k_{\text{track}} = 1.0$. Though trajectory tracking is only marginally affected, the centering performance index drops as might be expected with a lower arbitration gain. Interestingly, applying a gain of $k_{\text{centering}} = 2.0$ renders this same system unstable, a confirmation that changing a lone arbitration gain can directly influence the stability of the entire system.

Now, suppose that the PD controller was divided into separate position and damping behaviour controllers. The linear agent model then becomes:

$$\mathbf{u}_{cj} = [1 \;\; 1 \;\; 1]^T \begin{bmatrix} \mathbf{u}'_{\text{track}} \\ \mathbf{u}'_{\text{position}} \\ \mathbf{u}'_{\text{damping}} \end{bmatrix} \tag{7.332}$$

Then it becomes clear that local the position and velocity gains,k_q and k_w are, effectively, arbitration gains or:

$$\mathbf{u}_{cj} = [1 \;\; k_q \;\; k_w] \begin{bmatrix} \mathbf{u}'_{\text{track}} \\ \mathbf{q}_e \\ \dot{\mathbf{q}}_e \end{bmatrix} \tag{7.333}$$

One conclusion to draw from this argument is that weighted combination arbitration strategies (such as the linear model) affect both the performance of the combined tasks and *the stable performance of individual tasks*. Another is that compliance and arbitration are equivalent in the linear agent model.

With the recognition that arbitration and compliance are equivalent, adaptive goal generators (such as RMDAC) can be examined in a slightly different light. Using the RMDAC goal generator as an example and applying the arbitration gain k_{track} into the global goal proxy:

$$\mathbf{u}_{\text{track}} = \mathbf{J}_j^T(\mathbf{p}_n, \mathbf{x}_{j-1})\mathbf{f}_d \tag{7.334}$$

Since k_{track} is scalar, the gain may be moved arbitrarily into $\mathbf{f}_d$ or:

$$\begin{aligned} f_i(t) \;\; &= \;\; k_{\text{track}}[d_i(t) + c_i(t)\mathbf{r}_i(t) + b_i(t)\dot{\mathbf{r}}_i(t) + a_i(t)\ddot{\mathbf{r}}_i(t) + \\ & \quad\;\; K_{pi}(t)\mathbf{x}_e(t) + K_{vi}(t)\dot{\mathbf{x}}_e(t)] \end{aligned} \tag{7.335}$$

which with further simplification, it can be shown that k_{track} ultimately affects the integral gains. Recalling the generic integral gain structure of equation (6.206):

$$k'_i(v(t), e(t), t) = k_{\text{track}}[k_i(0) + u_{1i}\int_0^t v_i(t)e_i(t)dt + u_{2i}v_i(t)e_i(t)] \tag{7.336}$$

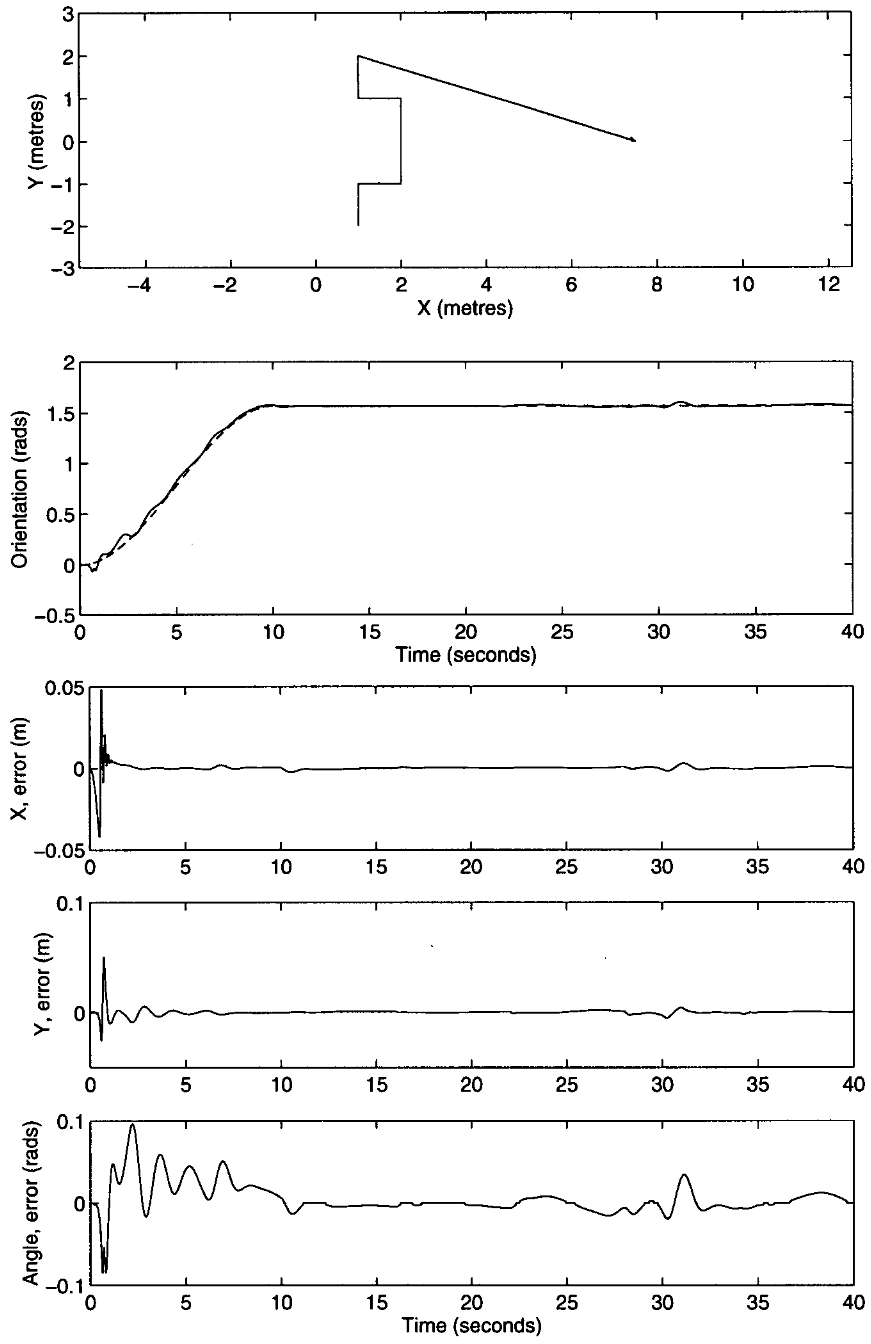


Figure 7.72: End effector tracking performance overconstrained by RMOA and N Centering controllers ($\mathbf{K}_q$ = diag(100) , $\mathbf{K}_w$ = diag(20)) and the agents arbitration vector $\mathbf{k}_{\mathsf{A}_j} = [k_{\text{track}}\ k_{\text{avoid}}\ k_{\text{centering}}] = [1.0\ 1.0\ 0.5]$.

Since $k_i(0)$ and u_{2i} are usually ignored in the cartesian case, as mentioned earlier, only the integral is affected:

$$k_i'(v(t), e(t), t) = k_{\text{track}} u_{1i} \int_0^t v_i(t) e_i(t) dt \tag{7.337}$$

In which case, k_{track} modifies either u_{1i}, $v_i(t)$ or $e_i(t)$ or possibly some multiplicative combination. So, even in the adaptive case, multiplicative arbitration gains will affect the performance of the individual controller as well as the combined performance of multiple goal systems.

Conversely, arbitration gains can be extracted from the existing goal generator. Just as in the PD case, the RMDAC global goal generator can be divided into six global goal generators and the Jacobian Transpose-integral coefficient products for each component elevated to an arbitration gain or:

$$\mathbf{u}_{cj} = \mathbf{k}^T_{\mathsf{A}_j} \; \mathbf{u}_{\mathsf{A}_j} \tag{7.338}$$

$$\mathbf{k}^T_{\mathsf{A}_j} = [\mathbf{J}_j^T \;\; \mathbf{J}_j^T\mathbf{C}(t) \;\; \mathbf{J}_j^T\mathbf{B}(t) \;\; \mathbf{J}_j^T\mathbf{A}(t) \;\; \mathbf{J}_j^T\mathbf{K}_p(t) \;\; \mathbf{J}_j^T\mathbf{K}_v(t) \;\; k_q \;\; k_w] \tag{7.339}$$

$$\mathbf{u}_{\mathsf{A}_j} = [\mathbf{d}(t) \;\; \dot{\mathbf{r}}(t) \;\; \ddot{\mathbf{r}}(t) \;\; \mathbf{x}_e(t) \;\; \dot{\mathbf{x}}_e(t) \;\; q_e \;\; \dot{q}_e \;] \tag{7.340}$$

Though the linear agent model *could* be expressed this way, this obscures the fact that the local and global generators have been designed as separate stable components. In other words, the behaviour controllers $\mathbf{u}_{\text{track}}$ and $\mathbf{u}_{\text{centering}}$ represent behaviour controller *groups* explicitly designed to work together to achieve a specific performance objective.

7.8 Summary

In this chapter, the design and interaction of combined behaviour systems has been explored in the context of multiagent manipulator control. These local and global behaviour controllers provide some important lessons on multiagent manipulator control in particular and multiagent system control in general.

In particular, the conclusions drawn from this chapter include:

- multiagency can be made robust to failure through simple, local measures.
- multiagency accommodates global goal generation by any agent.
- two design elements ensure the emergent coordination of an agent team towards multiple goals:
  - null space self organization
  - arbitration through compliance

- global and local goal systems can produce emergent global behaviour (i.e. manipulator shaping policy) in addition to desired global behaviours.

Fail safe methods discussed earlier demonstrate that even in the absence of an agent's global goal proxy, local strategies can be devised (such as a braking maneuver) that permit arbitrary global goal seeking to continue. Kinematically, this is not surprising. Disabling one agent's capacity to seek global goals removes a single degree of freedom from the team. If the team is redundant in the global goal, global behaviour should be unaffected. Architecturally, centralized, model based schemes such as RMPC, RMRC, and RMAC are not well suited to these simple low-level solutions to link failure principally due to the dependence on an explicit map between global and local goal spaces. Since these maps often require an accurate system model, precise fault descriptions are required to maintain acceptable global performance.

This chapter has shown that multiple goal systems are achievable for collections of independent link agents. Though investigated for some time, multiple tasks are usually incorporated into a centralized global process, disseminating primary and auxiliary tasks to each link according to centralized null space or task space augmentation models. Reexamined as a distributed, multiagent control problem, the previous chapter showed that centralized explicit manipulator control is not necessary for well posed global tasks. It has been shown in this chapter that any global goal conforming to Jacobian transpose methods may be generated and *broadcast* by any agent in a multiagent team. The addition of a simple PD local goal generator demonstrated not only that decentralized local and global goal systems can coexist in globally redundant systems but that this local generator can generate useful, even optimal, performance.

Significantly, this study identified two distinct mechanisms that together provide an automatic goal arbitration mechanism for agents pursuing multiple goals: *Null Space Self Organization* and *Arbitration through Compliance*

Null Space Self Organization

By adopting an adaptive cartesian controller, this study predicted and demonstrated that explicit identification of the null space and subsequent task insertion are unnecessary. As described in section 7.2, perfect regulation of the end effector position drives torque and displacement perturbations into the null space. This property has been used in JTC to implement mutually exclusive global goal systems. However, null space self organization has not previously been identified or exploited as a desirable byproduct of adaptive end effector task enforcement. Null space self organization ensures that local behaviours are automatically inserted of into the Jacobian null space of an adaptive global goal system. Furthermore, this technique shows that compliance can be used as a self arbitration mechanism in multiagent teams.

Arbitration through Compliance

Experiments in this chapter demonstrate that arbitration can be distributed amongst behaviour controllers through the selection of an appropriate compliance strategy for each behaviour. In particular, if the 'highest priority' task is enforced through a rigid goal regulator such as an adaptive controller and 'lower priority' behaviour controllers are enforced through less rigid strategies such as nonlinear and linear control laws, then task conflicts are automatically resolved between controllers and require neither an explicit arbitration strategy within each agent nor an explicit coordination strategy between agents.

Based on the Emergent Coordination definition D4.23, agents within a redundant multiagent manipulator under rigid global control and compliant local control exhibit an emergent coordination strategy and, therefore, emergent behaviour. Taking the centering local behaviour as an example, rigid end effector tracking drives local centering behaviour into the null space of the end effector Jacobian. In so doing, this interaction permits partial though maximal satisfaction of the local goal without coordination by a centralized coordination process. In effect, the interaction between the two goal systems alone acts as a coordination policy that enables desirable local behaviour if possible – an example of self arbitrating or emergent control.

Furthermore, if a team of similar agents produce a desirable, stable global behaviour without external coordination then the team exhibits emergent global behaviour, for example manipulator shaping policies arise from local compliant goal systems.

Chapter 8

Multiagent Control Compared

The application of multiagent control to manipulation was justified on the grounds that improved robustness, architectural simplicity, lower cost, greater extensibility, and improved real time performance observed in other multiagent systems might be realized in manipulator control. This chapter will show that this manipulator architecture delivers these benefits, and, furthermore, that these advantages are unique to multiagent control.

Multiagent manipulator control is not a single technique, but the careful reformulation of a number of fundamental techniques into a coherent control strategy. So while each component brings advantages in isolation, the combined performance is uniquely beneficial to redundant manipulator control. The key contributing components to multiagent manipulator control are:

- Redundant Manipulator Control
- Adaptive Control
- Jacobian Transpose Control
- Decentralized Control

The relationship between these foundations and multiagent control appear in figure 8.73. Guided by the agent and multiagent structures developed in chapter 4, these components can be combined, delivering two distinct advantages over traditional, centralized manipulator model based systems: *structure* that is simple, robust, and extensible, and *performance* that is consistent, guaranteed, and independent of a manipulator's dynamic structure.

8.0.1 Multiagent vs. Centralized Control

Manipulator model based systems (i.e. those relying on knowledge of the manipulator's structure and parameters) achieve consistent guaranteed performance, but at the cost of a centralized inverse kinematic solution (e.g. a geometric inverse or Jacobian pseudoinverse). For example, in resolved motion acceleration control, a desired acceleration is computed from the pseudoinverse of a task space trajectory:

$$\begin{aligned}\ddot{\mathbf{q}}_d(t) &= \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}_d(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v\dot{\mathbf{x}}_e + k_p\mathbf{x}_e\right] \\ \mathbf{u}_{\text{track}} &= \hat{\mathbf{D}}(\mathbf{q})\ddot{\mathbf{q}}_d + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q})\end{aligned}$$

This centralization extends to auxiliary tasks that must be assigned to regions of configuration space through a task assignment mechanism. For example, again in RMAC:

$$\mathbf{u} = \mathbf{u}_{\text{track}} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{u}_{\text{auxiliary}} \tag{8.341}$$

where auxiliary tasks, $\mathbf{u}_{\text{auxiliary}}$ are inserted into the Jacobian Null space by the $(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})$ operator.

By combining Adaptive Task Space Control with a decentralized Jacobian Transpose Controller an estimation of a tracking task model is maintained rather than a manipulator's structure and parameters. Alone this reduces computing costs from an at least $O(N^2)$ pseudoinversion and $O(N)$ dynamic model to simply $O(MN)$. Furthermore, by distributing the inverse solution amongst a set of link agents computational costs are further reduced from $O(MN)$ to $O(M)$. Finally, adaptive task space control ensures disturbances migrate to the Jacobian null space. This means multiple tasks can coexist within each agent, providing robust control alternatives to each link without the necessity of a centralized task assignment protocol.

Finally, multiagent systems permit either the removal, replacement, or addition of whole link agents at any time on or off line without costly software changes commonly required for traditional centralized control architectures. This has important implications on task or mission planning in which serial manipulators can be simply and easily "daisy chained" or "split" in real time, physically combining or dividing manipulators respectively.

Together these features of multiagent manipulator control provide performance comparable to the compute intensive, model based, centralized systems but with a substantially simpler structure, potentially lower implementation and maintenance cost, flexibility, and robustness to change or failure.

8.0.2 Multiagent Manipulator Control and the Multiagent Context

Though comparison of multiagent systems is difficult, the multiagent architecture presented here is unusual in that task assignment through explicit interagent bidding, negotiation, or competition, common to other systems, is unnecessary. Rather 'negotiation' emerges through the dynamics of both adaptive and compliant goal systems in a manner similar to Reynold's Boids. This is in contrast to Parker's Troops and

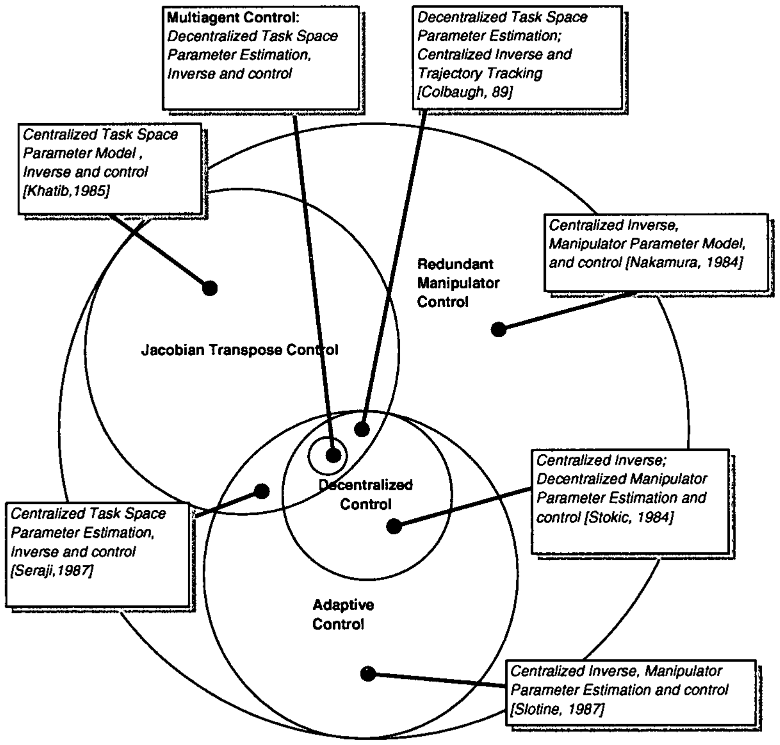

Multiagent Control: Decentralized Task Space Parameter Estimation, Inverse and control
Decentralized Task Space Parameter Estimation; Centralized Inverse and Trajectory Tracking [Colbaugh, 89]
Centralized Task Space Parameter Model, Inverse and control [Khatib,1985]
Centralized Inverse, Manipulator Parameter Model, and control [Nakamura, 1984]
Redundant Manipulator Control
Jacobian Transpose Control
Decentralized Control
Centralized Inverse; Decentralized Manipulator Parameter Estimation and control [Stokic, 1984]
Centralized Task Space Parameter Estimation, Inverse and control [Seraji,1987]
Adaptive Control
Centralized Inverse, Manipulator Parameter Estimation and control [Slotine, 1987]

Figure 8.73: Convergence of techniques that form Multiagent Manipulator Control.

Mataric's Nerd Herd that employ explicit negotiation protocols reminiscent of Smith's Contract Net Protocol [Smith, 1980].

8.1 Demonstrating The Advantages

To clearly demonstrate the advantages that multiagent manipulator control offers over traditional centralized model based control, the performance and structure must be compared. In the first experiment, traditional RMAC will be used to perform end effector trajectory tracking and joint centering tasks. This result should demonstrate the performance possible from a centralized model based end effector and task assignment system. In the second test, the task assignment system is removed and the joint centering tasks decentralized to each link. This result should demonstrate the structural sensitivity of RMAC to both manipulator modelling errors and/or decentralized auxiliary controllers, both unmodelled constraints within RMAC.

8.1.1 Performance

In the following experiment, the *independent variable* will be the required performance: simultaneous tracking of a desired end effector trajectory and joint displacement minimization. The *dependent variable* will be the structure required to achieve the desired behaviour, one the centralized model based RMAC, and the other the decentralized manipulator model-free Multiagent Manipulator Control system.

In resolved motion acceleration control the typical structures required to achieve these tasks are:

$$\ddot{\mathbf{q}}_d(t) = \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}_d(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v\dot{\mathbf{x}}_e + k_p\mathbf{x}_e\right] \tag{8.342}$$

$$\mathbf{u}_{\text{track}} = \hat{\mathbf{D}}(\mathbf{q})\ddot{\mathbf{q}}_d + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}) \tag{8.343}$$

$$\mathbf{u}_{\text{centering}} = \mathbf{K}_q\mathbf{q}_e + \mathbf{K}_w\dot{\mathbf{q}}_e \tag{8.344}$$

$$\mathbf{u} = \mathbf{u}_{\text{track}} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{u}_{\text{centering}} \tag{8.345}$$

and are very similar to [Slotine, 1988].Note the estimate of the manipulator's parameters and frequent use of the Jacobian pseudoinverse, both centralized models of the manipulator's kinematics and dynamics.

Adaptive task space control and decentralization make multiagent control's structure substantially simpler than the RMAC equivalent:

$$\begin{aligned}\mathbf{f}(t) &= \mathbf{d}(t) + \mathbf{C}(t)\mathbf{r}(t) + \mathbf{B}(t)\dot{\mathbf{r}}(t) + \mathbf{A}(t)\ddot{\mathbf{r}}(t) + \\ &\quad \mathbf{K}_p(t)\mathbf{x}_e(t) + \mathbf{K}_v(t)\dot{\mathbf{x}}_e(t)\end{aligned} \tag{8.346}$$

$$\mathbf{u}_{\text{track}} = \mathbf{J}_j^T(\mathbf{p}_n, \mathbf{x}_{j-1})\mathbf{f}_d \tag{8.347}$$

$$\mathbf{u}_{\text{centering}} = k_q q_{e_k} + k_w \dot{q}_{e_k} \tag{8.348}$$

$$\mathbf{u}_{cj} = \mathbf{k}_{\mathsf{A}_j}\mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1]\begin{bmatrix}\mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{centering}}\end{bmatrix} \tag{8.349}$$

This latter structure is the familiar link agent described in chapter 7 with both tracking and centering tasks. Again, note that there is no manipulator parameter description only a simple Newtonian dynamic model within the adaptive end effector goal generator. Furthermore computation of the cartesian states needed for the Jacobian transpose row is distributed over N agents. At no point in the multiagent system is it necessary to maintain a complete centralized representation of the manipulator geometry or physical parameters.

Simulation

Though the tracking and displacement minimization performance of the two systems are comparable, the computational cost is clearly less in the multiagent system. Comparing figures 8.74 and 7.52 in chapter 7 and results in table 8.19, it is clear that these two structures exhibit similar end effector tracking and displacement minimization performance. However, RMAC is substantially more compute intensive with both a feedforward dynamic model and a pseudoinverse Jacobian computation. Since the pseudoinverse structures cannot be decentralized over N joint processes, RMAC must be centralized on a single powerful CPU.

8.1.2 Structure

As remarked above, the structure of the multiagent system is considerably simpler than that of centralized RMAC. Suppose, however, that RMAC was treated as a global goal generating process like RMDAC, how would this centralized system perform in the computationally simpler agent-like structure:

$$\ddot{\mathbf{q}}_d(t) = \mathbf{J}^{-1}\left[\ddot{\mathbf{x}}_d(t) - \dot{\mathbf{J}}\dot{\mathbf{q}}(t) + k_v\dot{\mathbf{x}}_e + k_p\mathbf{x}_e\right] \tag{8.350}$$

$$\mathbf{u}_{\text{track}} = \hat{\mathbf{D}}(\mathbf{q})\ddot{\mathbf{q}}_d + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}) \tag{8.351}$$

$$\mathbf{u}_{\text{centering}} = k_q q_{e_k} + k_w \dot{q}_{e_k} \tag{8.352}$$

$$\mathbf{u}_{cj} = \mathbf{K}_{\mathsf{A}_j}\mathbf{u}_{\mathsf{A}_j} = [1 \;\; 1 \;\; 1]\begin{bmatrix}\mathbf{u}_{\text{track}} \\ \mathbf{u}_{\text{centering}}\end{bmatrix} \tag{8.353}$$

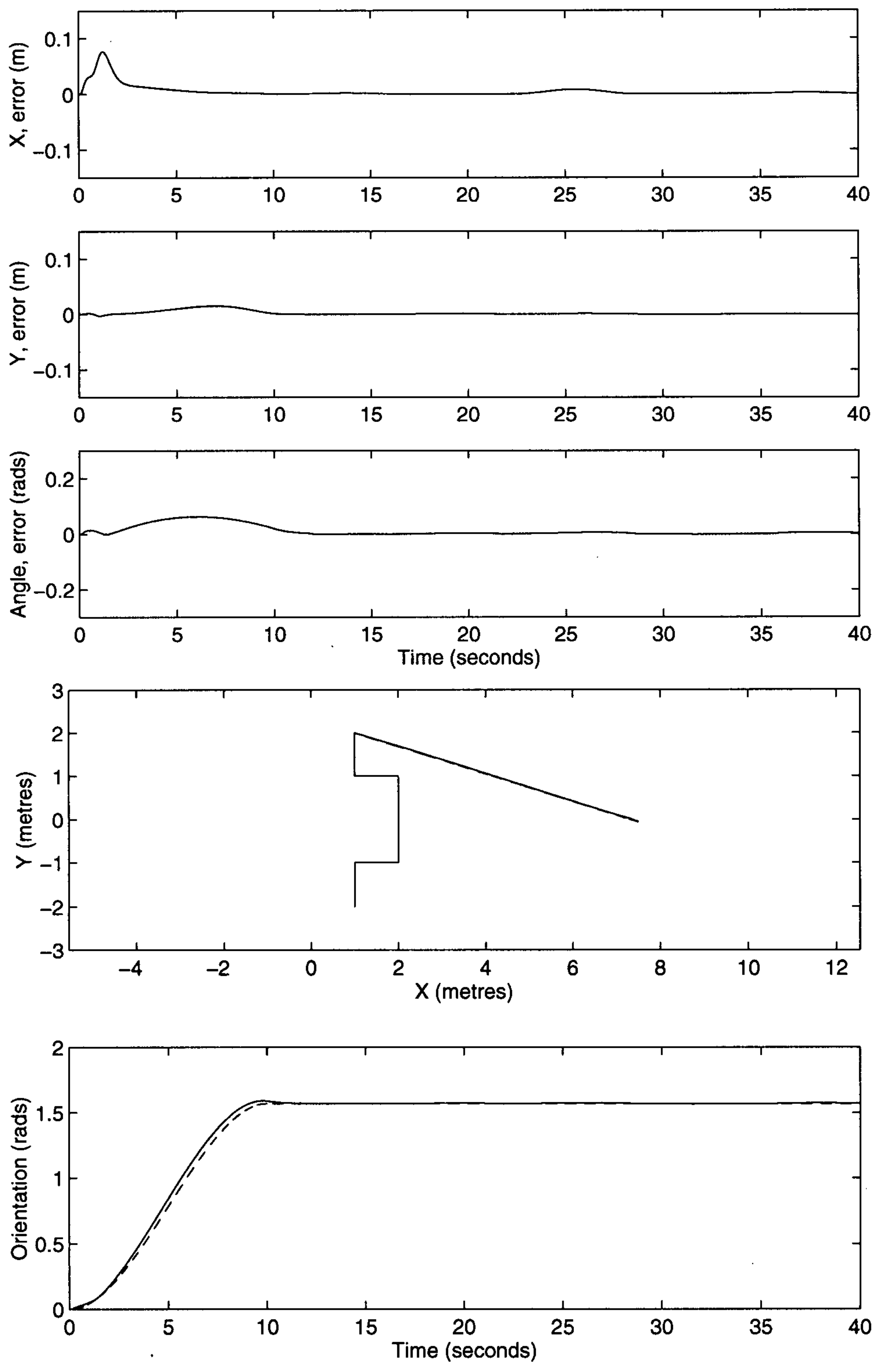


Figure 8.74: End effector trajectory for the reference manipulator and payload under centralized RMAC end effector and joint centering task assigned to the null space.

Simulation

In this experiment, the *independent variable* will be the structure: a distributed multiagent environment and decentralized control structure. The *dependent variable* will be performance: simultaneous tracking of a desired end effector trajectory and joint displacement minimization. This experiment may be viewed in two ways. On the one hand this experiment examines the performance of RMAC within a structure identical to multiagent control. On the other hand this experiment examines the performance of RMAC given an imperfect manipulator model (i.e. one in which joint controllers are biased by a linear PD controller).

Structurally, these two systems are now comparable. The computational cost of the agent-like RMAC is less than in the original RMAC system and local behaviours may act without centralized coordination.

With these changes, it is clear from figure 8.75 and table 8.19 that RMAC cannot match the performance of the multiagent system. Without centralized task assignment, the RMAC system exhibits considerably greater tracking performance error and somewhat poorer joint displacement minimization than the RMAC with null space task assignment. Interestingly, the multiagent control system is an order of magnitude more precise in trajectory tracking but exhibits poorer joint minimization performance than both RMAC cases. These results must be put in perspective.

The performance of the RMAC system without task assignment must be a compromise between end effector tracking and joint centering. Given the large local gains, end effector tracking performance suffers substantially, while centering performs well (indeed, very near RMAC with task assignment). In this version of Multiagent control, the adaptive end effector task enforces the highest priority and, therefore, exhibits small end effector errors. Unlike RMAC, RMDAC cannot compensate for manipulator dynamics in the Jacobian Null space, producing greater motion in $\mathbf{N}(\mathbf{J})$ and larger RMS error over the joint displacement history. Ultimately, however, figure 8.75 shows that without task assignment RMAC fails to provide acceptable steady state end effector performance.

Since RMAC is model based, arbitrary insertion of auxiliary tasks without altering the centralized model through a null space task insertion operator can only degrade end effector performance. In other words, model based systems rely on a complete description of all active constraints to achieve satisfactory control. To maintain this performance with added local behaviours requires that RMAC be altered through a task assignment mechanism.

The ability to arbitrarily add local behaviours without alterations to a centralized process combined with the lower computational costs of decentralized control suggest that the multiagent system more flexible,

Strategy	$\mathbf{e}_{\mathrm{rms}}$	θ_{rms}	$\mathbf{q}_{\mathrm{rms}}$
RMAC with task assignment	1.257e-02	1.109e-02	1.908e+00
RMAC without task assignment	5.612e-02	7.756e-02	1.942e+00
Multiagent Control	5.963e-03	1.424e-02	2.700e+00

Table 8.19: RMAC and Multiagent Control RMS end effector and joint centering performance.

extensible and robust than the model based counterpart. Clearly there are advantages to model based control if an accurate model is available. However, in a dynamic environment a static model is of little advantage and the costs associated with maintaining accurate dynamic models grows in proportion to the environment.

This comparison also provides an avenue to understanding the stability properties of a multiagent system with both local and global goals. In the next section stability will be discussed by first examining the localized stability properties of the combined 'compliant' goal system, RMAC (a task space PD controller) and 'compliant' joint centering local goals (joint space PD controllers). These arguments will clarify the properties of the 'rigid' global RMDAC (an adaptive task space controller) and 'compliant'joint centering goal system.

8.2 Stability

Qualitatively, some useful observations can be made about the combined performance of global and local goals by first considering the combination of compliant global and compliant local goals. If modelled as a pair of spring-damper systems, *compliant* local and global goals should exchange potential and kinetic energy, ultimately settling into equilibrium offset from both local and global desired trajectories.

Recall the case above of an adaptive global goal simulating a perfectly rigid (albeit, moving) task space constraint. Acting in combination with local *compliant* constraints, the adaptive controller corrected any end effector trajectory errors by adaptively changing the force setpoint. Since force setpoints are propagated to the agent team through the global goal, local control efforts that perturb the end effector position are, in effect, "reflected" off the global goal to the entire agent team.

Though the RMDAC global goal generator has been designed from the outset to be asymptotically stable, a global stability proof for a *combined* compliant local/RMDAC global system is difficult. Nevertheless, a valuable starting point is to first examine the stability of the combined locally and globally compliant system near equilibrium. With this simple analysis, the coupling mechanisms between the global and local

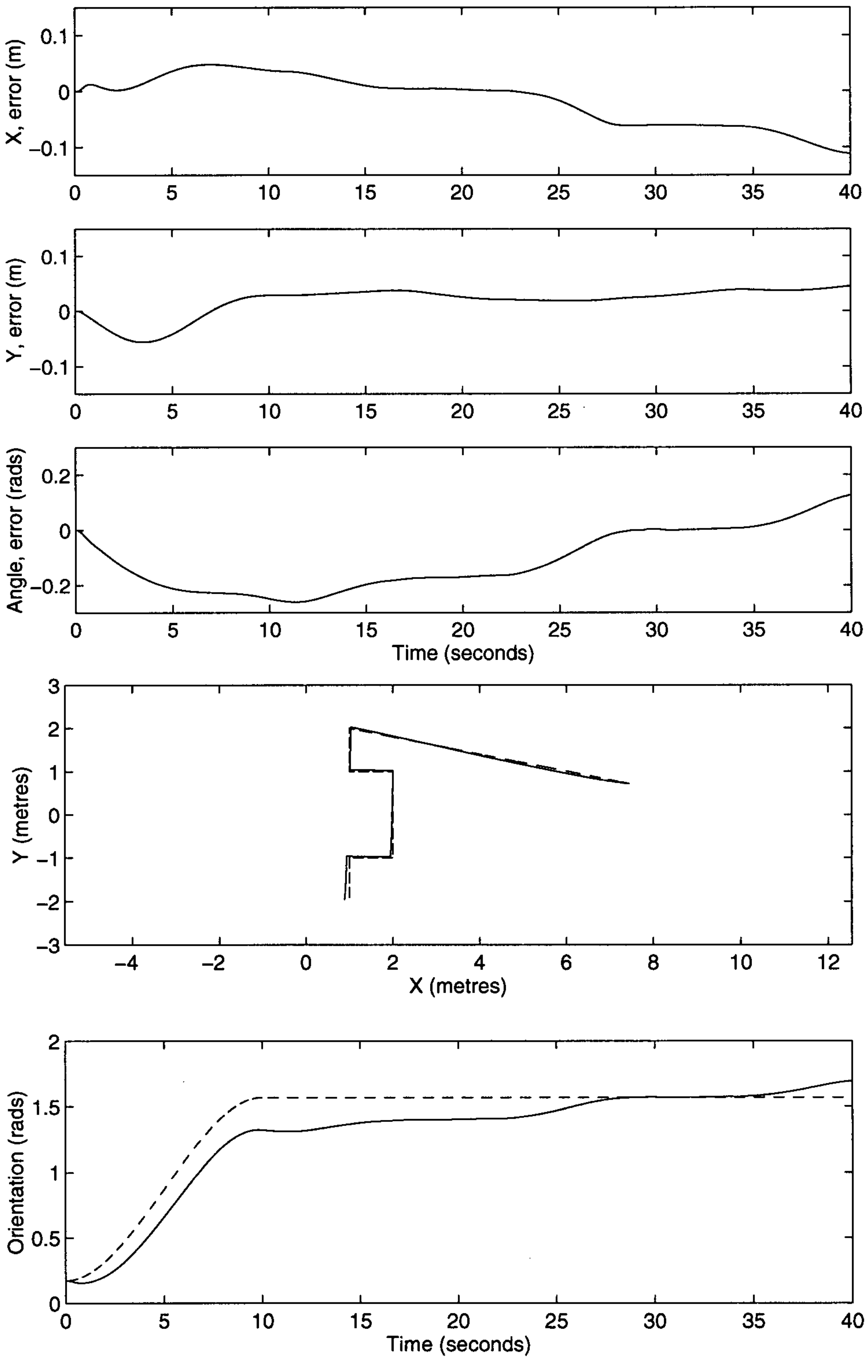


Figure 8.75: End effector trajectory for the reference manipulator and payload under centralized RMAC end effector and decentralized joint centering auxiliary tasks ($\mathbf{K}_q = \text{diag}(100.0)$, $\mathbf{K}_w = \text{diag}(20.0)$).

domains can be clearly observed and demonstrated. Furthermore, by extending these linear arguments the performance of locally compliant and globally adaptive systems can be clarified.

8.2.1 Combining Locally and Globally Compliant Goal Systems

A compliant global goal generator for task space manipulator control is best exemplified through a system such as Khatib's OSF, a cartesian computed torque controller. Now task space computed torque exploits a task space feedforward model in the construction of the required force vector $\mathbf{f}_d$:

$$\mathbf{f}_d = \hat{\mathbf{M}}(\mathbf{x})\mathbf{f}_m^* + \hat{\mathbf{N}}(\mathbf{x},\dot{\mathbf{x}}) + \hat{\mathbf{p}}(\mathbf{x}) \tag{8.354}$$

where the hat superscript indicates parameter estimates. If $\mathbf{f}_m^*$ is defined as the acceleration necessary to move a *unit mass*, identity matrix $\mathbf{I}_m$, along a prescribed trajectory,

$$\mathbf{f}_m^* = \mathbf{I}_m\ddot{\mathbf{x}}_d + \mathbf{K}_p(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_d(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) \tag{8.355}$$

Including the local controllers, the Robot Equation becomes:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{K}_q\left[\mathbf{q}_d - \mathbf{q}\right] + \mathbf{K}_w\left[\dot{\mathbf{q}}_d - \dot{\mathbf{q}}\right] + \mathbf{J}^T\left[\hat{\mathbf{M}}(\mathbf{x})\mathbf{f}_m^* + \hat{\mathbf{N}}(\mathbf{x},\dot{\mathbf{x}}) + \hat{\mathbf{p}}(\mathbf{x})\right] \tag{8.356}$$

where

$$\mathbf{M}(\mathbf{x}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{D}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}) \tag{8.357}$$

$$\mathbf{N}(\mathbf{x},\dot{\mathbf{x}}) = \mathbf{J}^{-T}\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{M}(\mathbf{x})\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} \tag{8.358}$$

$$\mathbf{p}(\mathbf{x}) = \mathbf{J}^{-T}\mathbf{g}(\mathbf{q}) \tag{8.359}$$

Now suppose $\hat{\mathbf{M}}(\mathbf{x}) = \mathbf{M}(\mathbf{x})$,$\hat{\mathbf{N}}(\mathbf{x},\dot{\mathbf{x}}) = \mathbf{N}(\mathbf{x},\dot{\mathbf{x}})$, and $\hat{\mathbf{p}}(\mathbf{x}) = \mathbf{p}(\mathbf{x})$ the simplified equation becomes:

$$\mathbf{K}_q\mathbf{q}_e + \mathbf{K}_w\dot{\mathbf{q}}_e + \mathbf{J}^T\hat{\mathbf{M}}(\mathbf{x})\left[\ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e\right] = 0 \tag{8.360}$$

where $\mathbf{x}_e = \mathbf{x}_d - \mathbf{x}$, $\mathbf{q}_e = \mathbf{q}_d - \mathbf{q}$ and, in general $\mathbf{x}_d \neq \mathbf{f}(\mathbf{q}_d)$.

This combined error system provides insight into the performance of combined goal operations. In particular, equation (8.360) indicates that if the parameter estimates are exact, the cartesian system will asymptotically converge to $\mathbf{x}_e = 0$ if and only if the first two terms vanish. This is true under two possible conditions:

1. $\mathbf{K}_q\mathbf{q}_e + \mathbf{K}_w\dot{\mathbf{q}}_e = \mathbf{0}$.

2. $\mathbf{K}_q\mathbf{q}_e + \mathbf{K}_w\dot{\mathbf{q}}_e \subset \mathbf{N}(\mathbf{J})$, the local goals reside in the null space of the Jacobian.

The first case is trivial, implying that either the setpoint error has vanished, position and velocity terms are equal and opposite, or the PD gains are precisely zero. The second possibility is that local and global systems reside in complementary regions of configuration space. If neither condition is true, then both systems reside in overlapping subspaces of $\mathbf{R}(\mathbf{J})$ and, though they may achieve a stable equilibrium, neither $\mathbf{q}_e$ nor $\mathbf{x}_e$ will converge to zero asymptotically.

Local Stability

Though equation (8.360) is a concise statement of the system's error dynamics, it does not completely show the coupling between global and local goal systems. By rewriting this equation entirely in task space a better understanding of these two systems can be established. Recalling that $\hat{\mathbf{M}} = \mathbf{J}^{\dagger T}\hat{\mathbf{D}}\mathbf{J}^{\dagger}$ where $\mathbf{J}^{\dagger}$ is the pseudoinverse :

$$\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{q}_e + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\dot{\mathbf{q}}_e + \ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e = 0 \qquad (8.361)$$

Since the term $\dot{\mathbf{q}}_e$ produces velocities at the end effector:

$$\dot{\mathbf{x}}_{qe} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}_e \qquad (8.362)$$

where $\dot{\mathbf{x}}_{qe}$ is the end effector velocity error induced by $\dot{\mathbf{q}}_e$. Inverting this relation the equation can be further rewritten:

$$\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{q}_e + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}\dot{\mathbf{x}}_{qe} + \ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e = 0 \qquad (8.363)$$

Now assuming the local displacement error $\mathbf{q}_e$ is *small*, the forward solution may be *linearized* about $\mathbf{q}$:

$$\mathbf{x}_{qe} = \mathbf{J}(\mathbf{q})\mathbf{q}_e \qquad (8.364)$$

A task space error expression can now be introduced: $\mathbf{x}_{qe} = \mathbf{x}_{qd} - \mathbf{x}$. Where $\mathbf{x}_{qe}$ is the displacement error between the end effector position at $\mathbf{q}$ and $\mathbf{q}_d$. Rewriting (8.363):

$$\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}\mathbf{x}_{qe} + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}\dot{\mathbf{x}}_{qe} + \ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e = 0 \qquad (8.365)$$

Now, referring to figure 8.76, $\mathbf{x}_{qe}$ and $\mathbf{x}_e$ can be related through:

$$\mathbf{e} = \mathbf{x}_d - \mathbf{x}_{qd} \qquad (8.366)$$

$$\mathbf{x}_{qe} = \mathbf{x}_{qd} - \mathbf{x} = \mathbf{x}_e - \mathbf{e} \qquad (8.367)$$

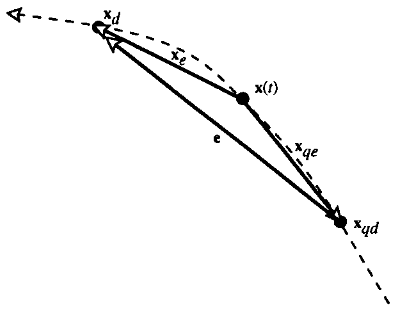


Figure 8.76: About the end effector position $\mathbf{x}(t)$, the desired global goal $\mathbf{x}_d$ is related to the desired local goal $\mathbf{x}_{qd}$ through the vectors $\mathbf{x}_e$, $\mathbf{x}_{qe}$ and $\mathbf{e}$.

substituting for $\mathbf{x}_{qe}$ and rewriting equation (8.365):

$$\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}(\mathbf{x}_e - \mathbf{e}) + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}(\dot{\mathbf{x}}_e - \dot{\mathbf{e}}) + \ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e = 0 \tag{8.368}$$

Applying the Laplace Transform:

$$\mathbf{X}_e(s) = \frac{\left[\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger} + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}s\right]}{\Delta(s)}\mathbf{E}(s) \tag{8.369}$$

$$\Delta(s) = s^2\mathbf{I} + [\mathbf{K}_d + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}]s + [\mathbf{K}_p + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}] \tag{8.370}$$

where the characteristic equation of the combined system is $\Delta(s)$. Through this simple linearized analysis, it is clear that local and global goal systems are closely coupled and that the performance of the combined system is a hybrid of both systems. In particular, task space damping and stiffness coefficients are clearly configuration dependent with the introduction of the local goal system.

If step inputs, $\mathbf{X}_d$ and $\mathbf{Q}_d$ (or simple $\mathbf{E}(s)$) , are applied to the cartesian and the local controllers respectively, the steady state error may be shown to be the equilibrium position between the two PD systems. Applying the final value theorem and recalling that $\mathbf{X}_e(s)$ is the end effector position error vector:

$$\begin{aligned}
\mathbf{x}_{e_{ss}} &= \lim_{s\to 0} s\mathbf{X}_e(s) \\
&= \lim_{s\to 0}\left[\left[s^2\mathbf{I} + [\mathbf{K}_d + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}]s + [\mathbf{K}_p + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}]\right]^{-1} s\left[\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger} + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_w\mathbf{J}^{\dagger}s\right]\frac{\mathbf{E}}{s}\right] \\
\mathbf{x}_{e_{ss}} &= [\mathbf{K}_p + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}]^{-1}\left[\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}\mathbf{E}\right]
\end{aligned} \tag{8.371}$$

Thus the intuitive conclusion that local goals residing in $\mathbf{R}(\mathbf{J}^T)$ cause steady state end effector offsets is borne out. As the 'stiffness' of the local goals, $\mathbf{K}_q$, increases, the cartesian steady state error, too, will increase. Clearly a system possessing both local and global compliant goals will experience steady state errors in both local and global domains if the local goals reside in $\mathbf{R}(\mathbf{J}^T)$.

Through further manipulation, the cartesian steady state error can be transformed into an expression of equilibrium between local and global systems. Rearranging equation (8.371) and substituting equation (8.367) produces:

$$\begin{aligned} \mathbf{K}_p\mathbf{x}_{e_{ss}} &= -\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}(\mathbf{E}-\mathbf{x}_{e_{ss}}) && (8.372)\\ &= -\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}\mathbf{x}_{qe_{ss}} && (8.373)\end{aligned}$$

Thus the steady state error is an equilibrium between the local and global systems:

$$\mathbf{J}^T\hat{\mathbf{M}}\mathbf{K}_p x_{e_{ss}} = -\mathbf{K}_q\mathbf{q}_{e_{ss}} \tag{8.374}$$

Of course if the parameter estimates are not exact:

$$\delta_D(\mathbf{q})\ddot{\mathbf{q}} + \delta_C(\mathbf{q},\dot{\mathbf{q}}) + \delta_g(\mathbf{q}) - \mathbf{K}_q\mathbf{q}_e - \mathbf{K}_w\dot{\mathbf{q}}_e = \mathbf{J}^T\hat{\mathbf{M}}(\mathbf{x})\left[\ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e\right] \tag{8.375}$$

the two linear systems will be driven by the nonlinear parameter errors defined as:

$$\begin{aligned} \delta_D(\mathbf{q}) &= \mathbf{D}(\mathbf{q}) - \hat{\mathbf{D}}(\mathbf{q}) && (8.376)\\ \delta_C(\mathbf{q},\dot{\mathbf{q}}) &= \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}}) && (8.377)\\ \delta_g(\mathbf{q}) &= \mathbf{g}(\mathbf{q}) - \hat{\mathbf{g}}(\mathbf{q}) && (8.378)\end{aligned}$$

Since gravitational forces are bounded [Lewis, 1993] and time invariant, parameter estimation errors reduce to errors in gravity compensation, $|\delta_g(\mathbf{q})| < \delta_G$, in the steady state. Such errors contribute to the steady state offset:

$$\mathbf{x}_{e_{ss}} = [\mathbf{K}_p + \mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}]^{-1}\left[\mathbf{J}\hat{\mathbf{D}}^{-1}\mathbf{K}_q\mathbf{J}^{\dagger}\mathbf{E} + \mathbf{J}\hat{\mathbf{D}}^{-1}\delta_G(\mathbf{q})\right] \tag{8.379}$$

It is important to reiterate that steady state offsets will arise only from local behaviours in $\mathbf{R}(\mathbf{J})$. However, in section 7.2 it was shown that local behaviours migrate to $\mathbf{N}(\mathbf{J})$ if $\mathbf{J}^{\dagger}\mathbf{J} \neq \mathbf{I}$, an intrinsic condition of redundant manipulation.

By *overconstraining* the system with both local and global goals, the combined performance of global and local goals can be demonstrated. Recalling that RMAC and OSF are functionally identical compliant goal systems, the results of the structural comparison in figure 8.75 show that, as predicted, a manipulator driven by a compliant goal system experiences steady state position and orientation offsets throughout the end effector trajectory if overconstrained by a compliant local goal system.

While it is true that end effector control of *underconstrained* serial manipulators ensures local behaviours migrate to $\mathbf{N}(\mathbf{J})$, it is also true that homogeneous teams of agents might inadvertently overconstrain the system with local goal activity. If this is so, the team should not have to rely on an omniscient explicit coordination system to arbitrate between local and global systems. Yet, both steady state analysis and simulation shows that centralized compliant methods (such as RMAC or OSF) *require* centralized coordination (e.g. equation (8.341)) to achieve convergence for *both* local and global goal systems under overconstraint. Now adaptive task space controllers (such as RMFC and RMDAC) are designed to overcome end effector errors and, as a consequence of the Null Space Self Organization theorem, drive local behaviours into $\mathbf{N}(\mathbf{J})$ without explicit coordination.

8.2.2 Combining Globally Adaptive and Locally Compliant Goal Systems

RMDAC was designed to ensure asymptotically stable trajectory tracking without an explicit manipulator model by adopting a model reference adaptive control strategy in cartesian space. The adaptive controller essentially adjusts task space estimates of the robots parameters to match the performance of an idealized linear tracking process. So, while RMDAC does establish a dynamic model, it is independent of a dynamic description of the robot.

Including the local PD controllers and the RMDAC global controller, the Robot Equation becomes:

$$\begin{aligned} \mathbf{D}(\mathbf{q})\ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})(t) + \mathbf{g}(\mathbf{q})(t) &= \mathbf{K}_q\mathbf{q}_e(t) + \mathbf{K}_w\dot{\mathbf{q}}_e(t) + \mathbf{J}^T\mathbf{f}_d(t) \\ \mathbf{f}_d(t) &= \mathbf{d}(t) + \mathbf{C}(t)\mathbf{x}_d(t) + \mathbf{B}(t)\dot{\mathbf{x}}_d(t) + \mathbf{A}(t)\ddot{\mathbf{x}}_d(t) \\ &\quad + \mathbf{K}_p(t)\mathbf{x}_e(t) + \mathbf{K}_v(t)\dot{\mathbf{x}}_e(t) \end{aligned} \tag{8.380}$$

Recall that RMDAC is the product of an MRAC Lyapunov design technique [Seraji, 1989a] converging not to a manipulator model but to an idealized trajectory tracking process. Therefore in general:

$$\begin{aligned} \mathbf{A}(t) &\neq \mathbf{M}(\mathbf{x}) \\ \mathbf{B}(t) &\neq \mathbf{N}(\mathbf{x},\dot{\mathbf{x}}) \\ \mathbf{C}(t) &\neq \mathbf{g}(\mathbf{x}) \end{aligned}$$

So while parameter convergence cannot, in general, be guaranteed, asymptotic convergence of the end effector with the desired trajectory is guaranteed under the assumption of slowly varying manipulator parameters. Therefore, so long as the slow variation assumption is maintained, local goals acting in $\mathbf{R}(\mathbf{J}^T)$ should appear as slow parameter changes and RMDAC should remain stable about the desired end effector trajectory. Assuming this is true, what performance can be expected from the combination of local and global control?

First consider, again, the combined response of locally and globally *compliant* controllers. If the system $\mathbf{N}(\mathbf{J}) = \emptyset$, the motion produced by an additional task in $\mathbf{R}(\mathbf{J}^T)$ inevitably disturbs the end effector trajectory by some offset (as in figure 8.75). Hence the necessity of explicitly inserting auxiliary tasks into $\mathbf{N}(\mathbf{J})$ and equation (3.70). On the other hand, if $\mathbf{N}(\mathbf{J}) \neq \emptyset$ (the robot is redundant), the resulting motion is, ultimately, forced into $\mathbf{N}(\mathbf{J})$ by the compliant control forces and producing no end effector disturbance.

Now consider the combined response of locally compliant and globally adaptive controllers. Since any disturbance at the end effector trajectory is adaptively removed, any local task (or portion thereof) in $\mathbf{R}(\mathbf{J}^T)$ is removed if $\mathbf{N}(\mathbf{J}) = \emptyset$. On the other hand, if $\mathbf{N}(\mathbf{J}) \neq \emptyset$ (the robot is redundant), a local task (or portion thereof) is, again, forced into $\mathbf{N}(\mathbf{J})$ by adaptive compensation.

Furthermore, since compensations in global behaviour are transmitted to the entire agent team. Adaptive global controllers can be interpreted as a natural goal assertion and broadcast mechanism. When local controllers act in $\mathbf{R}(\mathbf{J})$ and perturb the end effector, the compensation process 'reflects' or broadcasts this perturbation to the agent team. Thus even the end effector, fully constrained in task space by the end effector adaptive controller, can affect manipulator configuration simply by modulating global goal seeking with local activity.

8.3 Summary

Multiagent control offers comparable trajectory tracking performance to traditional systems at substantially lower computational cost, equal or better robustness to failure and parameter changes, and arbitrary extensibility thanks to a parallel agent computational model, carefully selected global and local controllers, and the innate properties of redundant manipulation.

Chapter 9

Conclusions, Contributions, and Recommendations

9.1 Conclusions

In summary, this thesis provided control analysis and simulation of an N d.o.f. redundant manipulator executing multiple tasks using N independent *agent* processes without a centralized manipulator parameter model or auxiliary task distribution mechanism. The control strategy combined an existing adaptive task space controller [Seraji, 1987b] a new decentralized Jacobian Transpose Control (JTC) [Monckton, 1995] method, and a carefully developed *agent* architecture to autonomously control each joint. In proving that task space adaptive control drives auxiliary tasks into the Jacobian Null space, this thesis developed a decentralized manipulation system permitting link agents to act on multiple local and global goals *without* manipulator model based setpoint derivation, auxiliary task assignment to the Jacobian Null Space [Nakamura, 1984] or task space augmentation. Thus each link process became an *agent* within a *multiagent* manipulation system that exhibited self organizing task assignment, priority through compliance, robustness to failure, and computationally parallelism – independent of team size, composition, or degree of redundancy.

9.1.1 Caveats

Though the feasibility and benefits of multiagent manipulator control were shown in both simulation and analysis, this controller has notable weaknesses that may compromise real world performance. Through Jacobian decentralization, multiagent manipulator control clearly depends on a distributed, reliable communications bus or shared memory structure. The failure or underperformance of this communications structure would lead to catastrophic failure of the control system as currently described.

Experimentation was restricted by the simulator's limited ability to model realistic manipulator and obstacle characteristics. For example: In adopting rigid link models, the effects of elastic deformation and collision were excluded and their impact on local and global goal systems remains unknown. Similarly, sensing and communications were idealized, ignoring sensor failure and/or noise along with unreliable communications (a significant feature of implemented systems such as Mataric's Nerd Herd). All of which have well known deleterious effects on the practicality and, often, stability of control systems.

Furthermore, the behaviours demonstrated here, while instructive, possess either limited or undetermined potential application. As implemented here, the obstacle avoidance simulation, for example, does not account for the manipulator itself (a common, if surprising, feature typical of such simulations). Though local PD behaviour controllers proved useful in revealing the interplay between goal systems they have undetermined practical application. Despite these caveats, this thesis provides fundamental contributions to the practice of multiagent control and manipulation that deserve review.

9.2 Contributions

Building on the considerable work of other investigators, this thesis contributes to both theory and practice of manipulation robotics and multiagent control. In particular:

Multiagent Manipulator Control A manipulator was controlled through a set of autonomous processes acting on a common force trajectory goal. The first of its kind, a multiprocess simulator simulated an agent *team* able to control a manipulator's end effector trajectory without centralized geometric or integral inverse kinematic solutions or a centralized manipulator parameter model. This parameter-free decentralization permitted the design of multiagent serial manipulation teams with constant time response regardless of team size or geometry.

Multiple Goal Interaction Traditionally manipulator redundancy is resolved through pseudoinverse or task space augmentation [Nakamura, 1984, Seraji, 1989b] methods that insert auxiliary tasks into the primary task's Jacobian null space thus guaranteeing multiple objectives. Through simultaneous action of both local compliant and adaptive global goals this study identified a number of significant properties and techniques:

Null Space Self Organization Drawing on a simple argument, a Null Space Self Organization Theorem was developed that proves rigid end effector control and explicit null space task assignment operators are functionally identical. This means that model based centralized task assignment operators (e.g. $(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})$) are unnecessary in redundant multitask systems under adaptive task space control.

Compliance as Priority If rigid global and compliant local goals occupy configuration subspace $\mathbf{R}(\mathbf{J}^T)$, global goals dominate and local goal activity is limited to the Jacobian null space. By assigning task priority through 'stiffness' or rigidity of auxiliary goal expressions, multiple tasks executed in overlapping domains can be biased according to priority.

Emergent Coordination Agents acting in the global goal's Jacobian range space were shown to disturb the global goal. In broadcasting corrections to these disturbances, global goal adaptation reflects local goal activity to the agent team. So while emergent global behaviour is not specified by a particular global goal expression, it arises from interaction between global and local constraints nevertheless.

Cartesian Decentralization Traditional inverse kinematic solutions (an inverse aggregate of end effector behaviour) uniformly require a complete manipulator description to relate end effector position to joint displacements. However, in this thesis, a simple, practical, distributed inversion strategy was developed out of the class of Jacobian Transpose Control. With cartesian sensing (or distributed Newton Euler kinematic computation), N processes can control N degree of freedom manipulators through the observation of a desired force trajectory goal.

Behaviours A number of new redundant manipulator null space configuration strategies were demonstrated. Arising from simple local and global goal systems, complex organized manipulator behaviours were generated with little additional computational effort. In particular:

- Analysis and simulation demonstrated that multiagent control is distinct from traditional centralized control. By overconstraining a redundant manipulator with an RMAC end effector goal and N PD joint centering local goals, it was shown that without explicit configuration space assignment of the centering goals, the RMAC end effector goal was disturbed.
- Analysis and simulation demonstrated that overconstraining a redundant manipulator with an RMDAC end effector goal and N PD joint centering local controllers, produced both near optimal joint displacement minimization without explicit configuration space assignment of the local goals and convergent end effector tracking performance.
- It was shown in simulation that by varying either setpoints or gains alone, local PD goal generators could produce qualitatively predictable, global manipulator shaping behaviours in conjunction with global end effector trajectory tracking without any centralized manipulator parameter model.
- In simulation it was shown that, given a redundant manipulator under an RMDAC end effector goal, N agents could each broadcast a nonlinear global obstacle avoidance goals to the agent team

to successfully avoid obstacles in the work volume without disturbing the end effector trajectory.

Agent and Multiagent Theoretical Foundations In placing agent and multiagent structures within a control theoretic context a simple agent control *architecture* was shown to be composed of a set of task specific *behaviour controllers*, an *arbitrator* that combined the output of these controllers to act on a *plant*. The *behaviour* of the agent was defined as the response of the plant to the arbitrated control efforts.

Similarly, a logical development of the multiagent model revealed basic structures and techniques that clarify the multiagent design problem:

Aggregate Relations An *aggregate relation* was identified that maps local to global behaviour. In doing so, this thesis recasts the problem of multiagent control as one of aggregate *inversion* and agent *coordination.*

Feasibility Samson's feasibility criteria (an application of the inverse function theorem) applied to a multiagent system's aggregate relation establishes the existence of the *inverse aggregate* and classifies these solutions into one-to-one, one-to-many, or insoluble.

Coordination *Explicit coordination* inverts the aggregate using a one-to-one map between global and local behaviour, clearly a model based process. *Implicit coordination* inverts the aggregate through a one-to-many projection. In manipulator control, it was shown that manipulator parameter-free, implicit coordination requires an adaptive global goal system. *Emergent Coordination* was shown to generate global behaviour as a byproduct of constraint resolution and, therefore, without a global goal description.

Linearized Stability Analysis Applied to both aggregate and inverse aggregates, linearized stability analysis linearization can be used to explore local stability criteria. In particular, it was shown that even with a nonlinear manipulation system, the stability (e.g. the steady state condition) of multiple goal systems could be determined locally through simple linearization.

In short, control theoretic expressions of agent and multiagent architectures represent an important first step in the practice of multiagent control design and clarifies critical issues of structure, performance, and stability of multiagent systems.

9.3 Recommendations

Though performing comparably to traditional centralized control architectures without centralized computation or coordination, the study of multiagent manipulator control remains immature. In particular, time delay, sensor noise, and flexible links or actuators are fundamentally unexplored issues and may present significant barriers to the adoption of this architecture in some applications.

While this thesis improves the understanding of agent and multiagent systems, there remain many outstanding issues that deserve further investigation:

- *Goal design methods.*Though partially explored here, the goal design process deserves more attention. For example: adaptation about a desired global behaviour clearly imposes order on the multiagent team. What other adaptive models exist for manipulation? How broadly applicable is this adaptive model to generic trajectory tracking tasks (i.e. mobile robotics)? How can local adaptive generators coexist with global adaptive systems? These are a few of many questions requiring attention in global and local goal design.

- *Arbitrator design.* It was shown here that arbitration influences team stability. The converse must also be true – that stability can govern arbitration design. It is conceivable that arbitrator design could be refined through further exploration of stability.

- *Team homogeneity and morphology.*Though it is a useful research model, team homogeneity is less likely in practice. Goal systems are likely to differ between agents by accident or by design (e.g. sensor suites, computational power, failures, etc.). What are the implications of heterogeneous teams?

Particular to manipulation, however, a number of potentially critical issues remain unresolved and deserve additional attention including:

- *Time delay.* The propagation of kinematic data along a shared memory bus structure invites time delay into end effector control. Though Seraji suggests that RMDAC will remain stable if control is faster than significant Jacobian rates of change (approx. 10^{-1} seconds), it is conceivable that slow sensing or high degrees of freedom may drive control rates into this region. What is the significance of slow or time delayed computation on end effector control?

- *Sensor noise.* Given appropriate filtering within each controller, sensor noise should not directly affect the practicality of multiagent control in general. However since filtering can introduce time delay, noise clearly can effect the stability of the multiagent manipulator controller described here. Given a particular set of behaviour controllers what is the sigificance of noise on the performance of a distributed, multiagent manipulation system?

- *multiple manipulators* under multiagent control. Perhaps the most intriguing. Broadcasting a single global goal to more than one multiagent team should lead to a competition between teams to provide the desired forces. Does global adaptation, as one might expect, resolve these conflicts automatically?
- *flexible links.* Space based applications in which payloads, links, actuators, and bases are no longer rigid bodies greatly complicates manipulator control. Can local or distributed goal systems be developed to mitigate these affects? Can flexibility couple goal systems (e.g. obstacle avoidance)?

Through the application of control theory to agent and multiagent systems in general and manipulation in particular, this thesis strives to outline and demonstrate an orderly method to the design and analysis of these new controllers. Yet with so many unanswered questions, this can only be one of many small steps on the road to a coherent multiagent control theory.

Bibliography

[Albus, 1990] J.S. Albus and R. Quintero. Toward a reference model architecture for real time intelligent control systems (arctics). In *ISRAM '90*, pages 243–250, July 1990.

[Ambler, 1975] A.P. Ambler and R.J. Popplestone. Inferring the positions of bodies from specified spatial relationships. *Artificial Intelligence*, 6, 1975.

[Ambler, 1980] A.P. Ambler and R.J. Popplestone. An interpreter language for describing assemblies. *Artificial Intelligence*, 14, 1980.

[An, 1989] C.H. An, C.G. Atkeson, and J.M. Hollerbach. *Model Based Control of a Robot Manipulator*. The MIT Press, 1989.

[Anderson, 1988] R.J. Anderson and M.W. Spong. Hybrid impedance control of manipulators. *IEEE Transactions of Robotics and Automation*, 4(5), 1988.

[Andersson, 1989] R.L. Andersson. Understanding and applying a robot ping-pong players expert controller. In *Proceedings 1989 IEEE Int. Conf. on Robotics and Automation*, pages 1284–1289, 1989.

[Arkin, 1987] R.C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4), August 1987.

[Arkin, 1991] R.C. Arkin. Reactive control as a substrate for telerobotic systems. *IEEE Transactions on Robotics and Automation*, 8(4), June 1991.

[Asada, 1986] H. Asada and J.J. Slotine. *Robot Analysis and Control.* John Wiley and Sons, 1986.

[Ballieul, 1985] J. Ballieul. Kinematic programming alternatives for redundant manipulators. In *Proceedings 1985 IEEE Int. Conf. on Robotics and Automation*, pages 722–728, 1985.

[Brooks, 1989a] R.A. Brooks. *A Robot that Walks: Emergent Behaviours from a Carefully Evolved Network*, chapter 24, pages 28–39. Artificial Intelligence at MIT. The MIT Press, 1989.

[Brooks, 1989b] R.A. Brooks. *Robotic Science*, chapter 11, The Whole Iguana, pages 432–456. The MIT Press, 1989.

[Brooks, 1989c] R.A. Brooks. *A Robust Layered Control System for a Mobile Robot*, chapter 24, pages 2–27. Artificial Intelligence at MIT. MIT Press, 1989.

[Brooks, 1991a] R.A. Brooks. *AI Memo No. 1293: Intelligence without Reason.* Massachusetts Institute of Technology, 1991.

[Brooks, 1991b] R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–159, 1991.

[Colbaugh, 1989] R. Colbaugh, H. Seraji, and K.L. Glass. Obstacle avoidance for redundant robots using configuration control. *Journal of Robotic Systems*, 6(6):722–744, 1989.

[Colombetti, 1996] M. Colombetti, M. Dorigo, and G. Borghi. Behaviour analysis and training - a methodology for behaviour engineering. *IEEE Transactions on Systems Man and Cybernetics*, 26(3):365–380, 1996.

[Connell, 1990] J.H. Connell. *Minimalist Mobile Robotics.* Academic Press, 1990.

[Cormen, 1990] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* McGraw-Hill., 1990.

[de Silva, 1989] C.W. de Silva. *Controls, Sensors, and Actuators.* Prentice-Hall, 1989.

[Denavit, 1955] J. Denavit and R.S. Hartenberg. A kinematic notation for lower pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, pages 215–221, June 1955.

[EL-Rayyes, 1990] L. EL-Rayyes, R.W. Toogood, I. Kermack, and D. McKay. Symbolic generation of robot dynamics. Software Documentation., December 1990.

[Ferrell, 1993] C. Ferrell. *Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators.* PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1993.

[Fichter, 1996] E.F. Fichter. A stewart platform-based manipulator: General theory and practical construction. *International Journal of Robotics Research*, 5(2):157–182, Summer 1996.

[Fisher, 1992] W.D. Fisher and S. Mujtaba. Hybrid position/force control: A correct formulation. *International Journal of Robotics Research*, 11(4):299–311, August 1992.

[Fu, 1987] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee. *Robotics Control, Sensing, Vision, and Intelligence.* McGraw-Hill, 1987.

[Fukunaga, 1990] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[Gat, 1994] E. Gat. Behavior control for robotic exploration of planetary surfaces. *IEEE Transactions of Robotics and Automation*, 10(4):490–503, August 1994.

[Glass, 1993] K. Glass. On-line collision avoidance for redundant manipulators. *Proceedings 1993 IEEE Int. Conf. on Robotics and Automation*, pages 36–43, 1993.

[Glass, 1995] K. Glass, R. Colbaugh, D. Lim, and H. Seraji. Real-time collision avoidance for redundant manipulators. *IEEE Transactions of Robotics and Automation*, 11(3):448–457, 1995.

[Goldstein, 1981] H. Goldstein. *Classical Mechanics, Second Edition.* Addison Wesley, 1981.

[Hartley, 1991] R. Hartley and F. Pipitone. Experiments with the subsumption architecture. In *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pages 1652–1658, April 1991.

[Hogan, 1985a] N. Hogan. Impedance control: An approach to manipulation: Part i theory. *Journal of Dynamic Systems, Measurement, and Control*, 107:1–7, 1985.

[Hogan, 1985b] N. Hogan. Impedance control: An approach to manipulation: Part ii implementation. *Journal of Dynamic Systems, Measurement, and Control*, 107:8–16, 1985.

[Hogan, 1985c] N. Hogan. Impedance control: An approach to manipulation: Part iii applications. *Journal of Dynamic Systems, Measurement, and Control*, 107:17–24, 1985.

[Hollerbach, 1987] J.M. Hollerbach and K.C. Suh. Redundancy resolution of manipulators through torque optimization. *IEEE Transactions of Robotics and Automation*, RA-3(4):308–316, August 1987.

[Jones, 1993] J.L. Jones and A.M. Flynn. *Mobile Robots: Inspiriation to Implementation.* A.K. Peters Ltd., Wellesly MA, 1993.

[Kazerounian, 1988] K. Kazerounian and Z. Wang. Global versus local optimization in redundancy resolution of robotic manipulators. *International Journal of Robotics Research*, 7(5):3–12, October 1988.

[Khatib, 1985] O. Khatib. Real time obstacle avoidance for manipulators and mobile robots. In *Proceedings 1985 IEEE Int. Conf. on Robotics and Automation*, 1985.

[Khatib, 1987] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Transactions of Robotics and Automation*, RA-3(1):43–53, February 1987.

[Kosecka, 1994] J. Kosecka and R. Bajcsy. Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12:187–198, 1994.

[Leahy Jr, 1994] M.B. Leahy Jr and G.N.Saridis. A behaviour based system for off road navigation. *IEEE Transactions of Robotics and Automation*, 10(6):776–782, December 1994.

[Lewis, 1993] F. Lewis, C.T. Abdallah, and D.M. Dawson. *Control of Robot Manipulators.* MacMillan Publishing Company, 1993.

[Lippman, 1991] S. B. Lippman. *C++ Primer.* Addison Wesley Company, 1991.

[Luh, 1980a] J.Y.S. Luh, M.W. Walker, and R.P. Paul. On line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102:69–76, 1980.

[Luh, 1980b] J.Y.S. Luh, M.W. Walker, and R.P. Paul. Resolved motion acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Controls*, AC-25(3), 1980.

[Lumia, 1994] R. Lumia. Using nasrem for real-time sensory interactive robot control. *Robotica*, 12:127–135, 1994.

[MacKenzie, 1995] D.C. MacKenzie, J.M. Cameron, and R.C. Arkin. Specification and execution of multiagent missions. In *Proceedings 1995 IROS Conference*, 1995.

[Mahadevan, 1992] S. Mahadevan and J.H. Connell. Automatic programming of behaviour based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[Mataric, 1992] M.J. Mataric. Minimizing complexity in controlling a mobile robot population. In *Proceedings 1992 IEEE Int. Conf. on Robotics and Automation*, May 1992.

[Mataric, 1994] M.J. Mataric. *Interaction and Intelligent Behaviour.* PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994.

[Mel, 1990] B. W. Mel. *Connectionist Robot Motion Planning.* Academic Press, 1990.

[Minsky, 1990] M. Minsky. *Excerpts from The Society of Mind*, volume 1, chapter 10, pages 245–269. The MIT Press, 1990.

[Monckton, 1995] S. P. Monckton and D. Cherchas. Jacobian transpose control: A foundation for multiagent manipulator control. In *1995 IEEE Int. Conf. on Systems, Man, and Cybernetics*, Oct. 22-25 1995.

[Nakamura, 1984] Y. Nakamura and H. Hanafusa. Task priority based redundancy control of robot manipulators. In *The Second International Symposium on Robotics Research.* The MIT Press, 1984.

[Nakamura, 1991] Y. Nakamura. *Advanced Robotics Redundancy and Optimization.* Addison Wesley Publishing Co., 1991.

[NeXT, 1993] NeXT. *NEXTSTEP Release 3.2.* NeXT Software Inc., 1993.

[Parker, 1992a] L.E. Parker. Adaptive selection for cooperative agent teams. In *Proceedings of the Second International Conference on Simulation of Adaptive Behaviour*, December 1992.

[Parker, 1992b] L.E. Parker. *AI Memo No. 1357: Local Versus Global Control Laws for Cooperative Agent Teams.* Massachusetts Institute of Technology, 1992.

[Parker, 1994] L.E. Parker. An experiment in robotic cooperation. In *Proceedings of the ASCE Specialty Conference on Robotics and Automation for Challenging Environments*, February 1994.

[Paul, 1981] R.P. Paul. *Robot Manipulators-Mathematics, Programming and Control.* MIT Press, 1981.

[Pratt, 1995] J.E. Pratt. Virtual model of a biped walking robot. Master's thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 1995.

[Press, 1988] W.H. Press, B.P. Flannery, S.A. Teulkolsky, W.T. Vetterling, and D.M. Dawson. *Numerical Recipes in C.* Cambridge University Press, 1988.

[Raibert, 1981] M.H. Raibert and J.J. Craig. Hybrid position/force control of manipulators. *Transactions of the ASME*, 102:126–133, June 1981.

[Raibert, 1984] M.H. Raibert and H.B.Brown Jr. Experiments in balance with a 2d one legged hopping machine. *Transactions of the ASME, Journal of Dynamic Systems and Control*, 106:75–81, March 1984.

[Ramadge, 1989] P.J.G. Ramadge and W.M. Wonham. The control of discrete events. *Proceedings of the IEEE*, 77(1):81–97, January 1989.

[Reynolds, 1987] C. W. Reynolds. Flocks, herds, and schools: a distributed behavioural model. *Computer Graphics*, 21(4):25–34, July 1987.

[Rosenblatt, 1995a] J.K. Rosenblatt. DAMN: A distributed architecture for mobile navigation. In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents , H. Hexmoor and D. Kortenkamp (Eds.).* AAAI Press, Menlo Park, CA., March 1995.

[Rosenblatt, 1995b] J.K. Rosenblatt and C.Thorpe. Combining multiple goals in a behavior-based architecture. In *Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS), Pittsburgh, PA*, August 1995.

[RTI, 1996] RTI. *Control Shell Object Oriented Framework for Real Time System Software - User's Manual.* Real Time Innovations, 1996.

[Samson, 1991] C. Samson, M. LeBorne, and B. Espiau. *Robot Control: The Task Function Approach.* Oxford University Press, 1991.

[Seraji, 1987a] H. Seraji. Adaptive force and position control of manipulators. *Journal of Robotic Systems*, 4(4):551–578, June 1987.

[Seraji, 1987b] H. Seraji. Direct adaptive control of manipulators in cartesian space. *Journal of Robotic Systems*, 4(1):157–158, April 1987.

[Seraji, 1989a] H. Seraji. Configuration control of redundant manipulators:theory and implementation. *IEEE Transactions of Robotics and Automation*, 5(4):472–490, August 1989.

[Seraji, 1989b] H. Seraji. Decentralized adaptive control of manipulators: Theory, simulation and experimentation. *IEEE Transactions of Robotics and Automation*, 5(2):183–201, April 1989.

[Seraji, 1989c] H. Seraji. Simple method for model reference adaptive control. *International Journal of Control*, 49(1):367–371, 1989.

[Seraji, 1993] H. Seraji. An on-line approach to coordinated mobilitiy and manipulation. In *Proceedings 1993 IEEE Int. Conf. on Robotics and Automation*, pages 28–35, 1993.

[Seraji, 1996a] H. Seraji. Sensor based collision avoidance: Theory and experiments. Email consultation, Mon, 3 Jun 1996 17:04:38, 1996.

[Seraji, 1996b] H. Seraji, R. Steele, and R. Ivlev. Sensor based collision avoidance: Theory and experiments. *Journal of Robotic Systems*, 13(9):571–586, 1996.

[Shoham, 1993] Yoav Shoham. Agent oriented programming. *Artificial Intelligence*, (60):51–92, October 1993.

[Simmons, 1991] R. Simmons and E. Kroktov. An integrated walking system for the ambler planetary rover. In *1991 IEEE International Conference on Robotics and Automation, Sacremento California*, April 1991.

[Simmons, 1994] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

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

[Slotine, 1988] H.Das, J.J.E. Slotine and T.B. Sheridan. Inverse kinematic algorithms for redundant systems. *IEEE Transactions on Robotics and Automation*, pages 43–48, 1988.

[Smith, 1980] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Commmunications*, Vol. C-29(12):1104–1103, 1980.

[Spong, 1989] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons, 1989.

[Steels, 1991] L. Steels. Towards a theory of emergent functionality. In *From Animals to Animats - Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, September 1991.

[Steels, 1994] L. Steels. Mathematical analysis of behaviour systems. In *Proceedings of Prearc Conference Lausanne*, 1994.

[Stokić, 1984] D. Stokić and M. Vukobratović. Practical stabilization of robotic systems by decentralized control. *Automatica*, 20(3):353–358, 1984.

[Sung, 1996] W.Y. Sung, K.D. Cho, and M.J. Chung. A constrained optimization approach to resolving manipulator redundancy. In *Journal of Robotic Systems*, pages 275–288, 1996.

[Thomas, 1988] F. Thomas and C. Torras. A group theoretic approach to the computation of symbolic parts relations. *IEEE Journal of Robotics and Automation*, 1988.

[Tu, 1994] D. Terzopoulos , X. Tu and R. Grzeszczuk. Artificial fishes with autonomous locomotion, perception, behavior, and learning in a simulated physical world. In *Proceedings of Artificial Life IV Workshop*, July 1994.

[Upstill, 1990] S. Upstill. *The RenderMan Companion.* Addison-Wesley, 1990.

[van de Panne, 1992] M. van de Panne and E. Fiume. A controller for the dynamic walk of a biped across variable terrain. In *Proceedings of the 31st IEEE conference on Decision and Control*, 1992.

[Verschure, 1992] P.F.M.J. Verschure, B.J.A Krose, and R. Pfeifer. Distributed adaptive control: The self organization of structured behaviour. *Robotics and Autonomous Systems*, 9:181–196, 1992.

[Vidyasagar, 1993] M. Vidyasagar. *Nonlinear Systems Analysis.* Prentice Hall, 2nd edition, 1993.

[Walker, 1982] M.W. Walker and D.E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement and Control*, 104:205–211, September 1982.

[Walter, 1950a] W.G. Walter. An imitation of life. *Scientific American*, 182(5):42–45, May 1950.

[Walter, 1950b] W.G. Walter. A machine that learns. *Scientific American*, 185(2):60–63, August 1950.

[Weir, 1984] M. Weir. *Goal Directed Behaviour.* Gordon and Breach Science Publishers, 1984.

[Weiss, 1987] L.E. Weiss, A.C. Sanderson, and C.P. Neuman. Dynamic sensor based control of robots with visual feedback. *IEEE Transactions of Robotics and Automation*, RA-3(5), October 1987.

[Whitney, 1969] D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man, Machines and Systems*, MMS-10(2), 1969.

[Winograd, 1972] T. Winograd. *Understanding Natural Language.* Academic Press, New York, 1972.

[Wu, 1982] C.H. Wu and R.P. Paul. Resolved motion force control of robot manipulator. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-12(3), 1982.

[Yuan, 1988] J.S.C. Yuan. Closed loop manipulator control using quaternion feedback. *IEEE Transactions of Robotics and Automation*, 4(4):434–440, 1988.

[Zhang, 1995] Y. Zhang and A.K. Mackworth. Constraint nets: a semantic model for dynamic systems. *Theoretical Computer Science*, (138):211–239, 1995.

Appendix A

Quaternions

Quaternions are the $\mathbb{R}^3$ equivalent of complex numbers, having *real* and imaginary *pure* components. The quaternion is composed of a scalar magnitude and a vector. Often these are combined into a single vector representation to create a four parameter representation of orientation.

A.1 Definition

A quaternion may be expressed as a couple $\{\eta, \vec{\mathbf{q}}\}$ where η is an element of $\mathbb{R}$ while $\vec{\mathbf{q}}$ is an element of $\mathbb{R}^3$. Suppose two coordinate systems exist seperated by a rotation φ about the *unit* vector $\vec{\mathbf{r}}$ [1]. The components η and $\vec{\mathbf{q}}$ are described by

$$\eta = \cos\frac{\varphi}{2} \tag{A.381}$$

$$\vec{\mathbf{q}} = \sin\frac{\varphi}{2}\vec{\mathbf{r}} \tag{A.382}$$

$$\lambda = \{\eta, \vec{\mathbf{q}}\} \tag{A.383}$$

where λ is a notational convenience representing the quaternion $(\eta, \vec{\mathbf{q}})$. Extending this notation:

$$\mathcal{R}(\lambda) = \eta \tag{A.384}$$

$$\mathcal{P}(\lambda) = \vec{\mathbf{q}} \tag{A.385}$$

where $\mathcal{R}()$ is the *real* quaternion component and $\mathcal{P}()$ is the *pure* quaternion component.

A.2 Properties

Uniqueness Both $\{\eta, \vec{\mathbf{q}}\}$ and $\{-\eta, -\vec{\mathbf{q}}\}$ describe the same orientation. If the rotation φ is limited to $-\pi \leq \varphi \leq \pi$ then η is nonnegative and the quaternion is unique.

Normality From the definition it is clear that:

$$\eta^2 + \vec{\mathbf{q}}^{\mathrm{T}}\vec{\mathbf{q}} = 1 \tag{A.386}$$

[1] Thus this becomes a *unitary* quaternion, commonly called Euler Parameters

Addition

$$\{\eta_1, \vec{\mathbf{q}}_1\} + \{\eta_2, \vec{\mathbf{q}}_2\} = \{\eta_1 + \eta_2, \vec{\mathbf{q}}_1 + \vec{\mathbf{q}}_2\} \tag{A.387}$$

Conjugate

$$\bar{\lambda} = \mathcal{R}(\lambda) - \mathcal{P}(\lambda) \tag{A.388}$$

Inner Product

$$< \lambda, \mu >= \frac{1}{2}(\bar{\lambda}\mu + \bar{\mu}\lambda) \tag{A.389}$$

Multiplication

$$\{\eta_1, \vec{\mathbf{q}}_1\}\{\eta_2, \vec{\mathbf{q}}_2\} = \{\eta_1\eta_2 - \vec{\mathbf{q}}_1 \cdot \vec{\mathbf{q}}_2, \vec{\mathbf{q}}_1 \times \vec{\mathbf{q}}_2 + \eta_1\vec{\mathbf{q}}_2 + \eta_2\vec{\mathbf{q}}_1\} \tag{A.390}$$

Norm

$$\| \lambda \| = \sqrt{\lambda\bar{\lambda}} = \sqrt{\eta^2 + \| \mathbf{q} \|^2} \tag{A.391}$$

Inverse and Equalities

$$\begin{aligned} \lambda^{-1} &= \frac{\bar{\lambda}}{\| \mathbf{q} \|^2} && \text{(A.392)} \\ \| \lambda\mu \| &= \| \lambda \| \, \| \mu \| && \text{(A.393)} \end{aligned}$$

Propagation If $\mathcal{F}_1$ rotates about $\mathcal{F}_0$ at an instantaneous angular velocity of ω. Then the quaternion $\{\eta, \vec{\mathbf{q}}\}$ evolves in time according to the following differential equation:

$$\begin{bmatrix} \dot{\eta} \\ \dot{\vec{\mathbf{q}}} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega^T \\ \omega & -\omega\times \end{bmatrix} \begin{bmatrix} \eta \\ \vec{\mathbf{q}} \end{bmatrix} \tag{A.394}$$

where ω is expressed in the coordinates of $\mathcal{F}_1$ and $\omega\times$ is the skew matrix of ω.

A.3 Relation to the Rotation Matrix

A rotation matrix may be generated from a quaternion through either the arbitrary rotation expression outlined earlier or through one of the two expressions below. In this instance the quaternion represents a rotation from frame 0 to frame 1.

$${}^0\mathbf{R}_1 = \cos\varphi\mathbf{I} + (1 - \cos\varphi)\vec{\mathbf{r}}^T\vec{\mathbf{r}} - \sin\varphi\vec{\mathbf{r}}\times \tag{A.395}$$

where $\vec{\mathbf{r}}\times$ is the skew matrix of $\vec{\mathbf{r}}$. Using the definition the above expression may be rewritten

$${}^0\mathbf{R}_1 = (\eta^2 - \vec{\mathbf{q}}^T\vec{\mathbf{q}})\mathbf{I} + 2\vec{\mathbf{q}}\vec{\mathbf{q}}^T - 2\eta\vec{\mathbf{q}}\times \tag{A.396}$$

where $\vec{\mathbf{q}}\times$ is the skew matrix of $\vec{\mathbf{q}}$.

Appendix B

Orientation Error

Orientation error for the purposes of cartesian control requires compact representation and therefore, limitation of redundant terms. Two possible approaches have been developed in literature. The differential motion vector is based upon homogeneous transformation algebra and assumes that the orientation error may be expressed through a small angle approximation [Paul, 1981]. Quaternion orientation formulations, adapted from spacecraft attitude control, have been used for cartesian robotic control [Yuan, 1988].

B.1 Differential Motion Vector or Euler Rotations

A simple form of cartesian error, discussed in [Fu, 1987] in reference to [Wu, 1982], is the differential motion vector derived in [Paul, 1981]. This technique, [Fu, 1987] (pg. 177-181), requires the development of a transformation between actual and desired end effector orientations. This transformation employs the small angle approximation of the expressions for rotation about each principle axis.

The expression for cartesian error, the *differential motion vector*:

$$\vec{x}_d - \vec{x}_a = \vec{x}_e = \begin{bmatrix} dx \\ dy \\ dz \\ \delta x \\ \delta y \\ \delta z \end{bmatrix} \tag{B.397}$$

where δx,δx, and δz are incremental rotations about the x, y, and z axes respectively. This translates into the following homogeneous transformation:

$$T_{\mathrm{nd}} = T_{\mathrm{na}} \begin{bmatrix} 1 & -\delta z & \delta y & dx \\ \delta z & 1 & -\delta x & dy \\ -\delta y & \delta x & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.398}$$

where T_{nd} and T_{na} are the desired and actual end effector locations of an n degree of freedom manipulator respectively. With the above relation, [Paul, 1981] reduces $\vec{x}_e$ into the following form:

$$\vec{x}_e = \begin{bmatrix} dx \\ dy \\ dz \\ \delta x \\ \delta y \\ \delta z \end{bmatrix} = \begin{bmatrix} \vec{n_a} \cdot (\vec{p_d} - \vec{p_a}) \\ \vec{o_a} \cdot (\vec{p_d} - \vec{p_a}) \\ \vec{a_a} \cdot (\vec{p_d} - \vec{p_a}) \\ \frac{1}{2}(\vec{a_a} \cdot \vec{o_d} - \vec{a_d} \cdot \vec{o_a}) \\ \frac{1}{2}(\vec{n_a} \cdot \vec{a_d} - \vec{n_d} \cdot \vec{a_a}) \\ \frac{1}{2}(\vec{o_a} \cdot \vec{n_d} - \vec{o_d} \cdot \vec{n_a}) \end{bmatrix} \tag{B.399}$$

While this is relatively conservative of space compared to the full 3×3 rotation matrix, the small angle approximation, [Paul, 1981], might prove to be a problem over large sampling periods. [Yuan, 1988] points out that this scheme is indeed stable over large orientation errors, but after linearization over small intervals is stable only over small intervals. Splitting the vector $\vec{x}_e$ into its position and orientation components $\vec{e}_p$ and $\vec{e}_o$. The latter is called the expression for *Euler Rotation.* According to [Luh, 1980b]:

$$\vec{e}_o = \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} = \sin\varphi\,\vec{\mathbf{r}} \tag{B.400}$$

employing the half angle formula:

$$\begin{aligned} \vec{e}_o &= \sin\varphi\,\vec{\mathbf{r}} && \text{(B.401)} \\ &= 2\cos\frac{\varphi}{2}\sin\frac{\varphi}{2}\,\vec{\mathbf{r}} && \text{(B.402)} \\ &= 2\delta\eta\,\delta\vec{\mathbf{q}} && \text{(B.403)} \end{aligned}$$

the latter being the Quaternion expression for Euler Rotations. See the appendix A for a discussion of Quaternions.

B.2 Quaternion Orientation Error

[Yuan, 1988] introduces Euler Parameter (Unitary or Normalized Quaternion) techniques from spacecraft attitude control to manipulator control.The quaternion expression for orientation error between two coordinate frames $\mathcal{F}_1$ and $\mathcal{F}_2$ is

$$\begin{aligned} \delta\eta &= \eta_1\eta_2 + \vec{\mathbf{q}}_1^T\vec{\mathbf{q}}_2 && \text{(B.404)} \\ \delta\vec{\mathbf{q}} &= \eta_1\vec{\mathbf{q}}_2 - \eta_2\vec{\mathbf{q}}_1 - \vec{\mathbf{q}}_1 \times \vec{\mathbf{q}}_2 && \text{(B.405)} \end{aligned}$$

[Yuan, 1988] determines that two coordinate systems coincide iff $\delta\mathbf{q} = 0$.

If $\mathcal{F}_1$ and $\mathcal{F}_2$ are the desired and actual hand coordinates relative to the base, then (B.405) is the orientation error. Now when two frames coincide:

$$\eta_1 = \eta_2 \tag{B.406}$$

$$\vec{\mathbf{q}}_1 = \vec{\mathbf{q}}_2 \tag{B.407}$$

or using the normality condition A.386:

$$\delta\eta = 1 \tag{B.408}$$

$$\delta\vec{\mathbf{q}} = 0 \tag{B.409}$$

Now at $\delta\vec{\mathbf{q}} = 0$

$$\eta_1\vec{\mathbf{q}}_2 - \eta_2\vec{\mathbf{q}}_1 = \vec{\mathbf{q}}_1 \times \vec{\mathbf{q}}_2 \tag{B.410}$$

Plainly $\eta_1\vec{\mathbf{q}}_2 - \eta_2\vec{\mathbf{q}}_1$, $\vec{\mathbf{q}}_1$, and $\vec{\mathbf{q}}_2$ share the same plane. $\vec{\mathbf{q}}_1 \times \vec{\mathbf{q}}_2$, therefore, is perpendicular to this plane. For (B.409) to be true, then both sides are zero or:

$$\eta_1\vec{\mathbf{q}}_2 = \eta_2\vec{\mathbf{q}}_1 \tag{B.411}$$

Then $\vec{\mathbf{q}}_1$ is a multiple of $\vec{\mathbf{q}}_2$. Since both vectors are of unit length and both vectors are aligned, the normality (A.386) condition may again be used:

$$\delta\eta = \eta_1\eta_2 + \vec{\mathbf{q}}_1^T\vec{\mathbf{q}}_2 = \pm 1 \tag{B.412}$$

Therefore:

$$\{\eta_1, \vec{\mathbf{q}}_1\} = \{\pm\eta_2, \pm\vec{\mathbf{q}}_2\} \tag{B.413}$$

both of which have the same orientation (see section A.2). Therefore B.405 is the orientation error between two coordinate systems.

Appendix C

Scenario Specification Database

The Multiprocess Manipulator Simulator uses a flexible group of database files to specifiy a particular manipulation scenario. A scenario is composed of a manipulator configuration, a global goal generation system for the manipulator, the global goal (a trajectory specification) for the end effector, and an obstacle specification.

To implement a 'database' that is legible to both man and machine a simple data formatting syntax for 'scenarios' has been adopted and will be discussed in this appendix. There are four very similar format types:

- Manipulator Description files
- Global Goal Description files
- Obstacle Description files
- End Effector Trajectory Description files

all contained within a single directory suffixed by "`.scenario`" (e.g. `GainSeries_01.scenario`. Each format type relies on a common set of basic elements. In particular, the scoping operators `define{}` and `end{}` limit the scope of defined variables to particular objects. Current objects include `link`, `controller`, `globalgoal`, `obstacle`, and `knotpoint` definitions. Formal parameters may be assigned within a given `define{}`- `end{}` pair through the familiar variable assignment syntax: `varname = parameter`. Formal parameters are those permanently defined within a specific objects. Informal parameters are those specified for a particular instance of objects (e.g. the radius of a ball obstacle) and have an 'escape' syntax: `@ varname = parameter`. Regardless of parameter formality only four variable types are supported:

double in exponential format.

vector whitespace delimited doubles enclosed in angle brackets `< >`.

matrix whitespace delimited vectors enclosed in square brackets `[ ]`.

strings whitespace delimited ASCII characters.

The following sections will detail the use of object instances within each scenario element.

C.1 Manipulator Description Files (MDF): `theRobot.rbt`

The Manipulator Description File documents all of the physical and local control parameters that constitute a link agent. The format is designed to be self documenting. Informal parameters to be added to the syntax without requiring a (time consuming) language redefinition.

Link objects are used to define individual links in the manipulator. The order of the link objects in the file represents the order of the links from base to end effector. There is no limit on the number of link objects. Formal parameters include mass, principle moments of inertia, and Denavit Hartenberg parameters.

To implement the jth link's local controllers $\mathbf{H}_{ij}$, the `controller` object may be instantiated. The order of instantiation is of no consequence *except where communication between local and global goals requires allocation of IOstreams.* In this case the order of local and global controller instantiation should be identical. Local controllers have only one formal parameter, a name identifier (a string e.g. "resolved_motion_force_control") followed by a list of informal parameters. There is an arbitrary limit of ten local controllers that may be instantiated for a given link. Not surprisingly, failure to provide an MDF within a scenario is an error.

The data file segment in figure C.77 demonstrates the manipulator description language employed in this simulator.

C.2 Global Goal Description Files (GDF): `theGoals.dat`

Global goal description files, as in figure C.78, are identical to the controller descriptions within the MDF described earler and carry the same caveats for order of instantiation. If the global goal file is omitted, simulation proceeds entirely using local goals alone.

C.3 Trajectory Description Files: `theTrajectory.spt`

Trajectory Description Files are simple lists of knotpoints in global coordinates. Each knotpoint is composed of a time interval over which the knotpoint is attained, an homogeneous transformation describing the knotpoint position [1], and a final velocity vector. The starting point is function of the moving entities starting position and is assumed to be unknown at startup (as in the case of a manipulator end effector).

[1] Homogeneous transformations are generally more man readable than quaternion-cartesian hybrids.

```
gravity = <+0.000e+00 +0.000e+00 +0.000e+00>
define{link}
        name = link_1
        type = revolute
        theta = +0.000e+00
        offset = +0.000e+00
        alpha = +0.000e+00
        length = +0.750e+00
        mass = +1.000e-01
        saturation = +1.000e+06
        velocity = +0.000e+00
        inertia =  <+5.000e-02 +4.938e-01 +4.938e-01>
        centroid =  <-0.375e+00 +0.000e+00 +0.000e+00>
        define{controller}
                name = resolved_motion_force_control
                @ ControlRate = 1.200e+02
        end{controller}
        .
        .
        .
end{link}
define{link}
.
.
.
end{link}
```

Figure C.77: A robot Specification Database flat (ASCII) file. Note that the `link` list order is from manipulator base (frame 1) to end effector (frame N).

```
define{globalgoal}
     name = resolved_motion_adaptive_control
     @ ControlRate = 1.200e+02
     @ Wp =  <+1.800e+03 +1.800e+03 +1.800e+03 +5.000e+01 +5.000e+01 +5.000e+01>
     @ Wv =  <+8.000e+02 +8.000e+02 +8.000e+02 +1.000e+01 +1.000e+01 +1.000e+01>
     @ f_zero = +0.000e+00
     @ f_coeff  =  <+0.000e+00 +6.000e+00 +0.000e+00>

     @ k0_zero = +0.000e+00
     @ k0_coeff =  <+0.000e+00 +4.000e+00 +0.000e+00>

     @ k1_zero = +0.000e+00
     @ k1_coeff =  <+0.000e+00 +4.000e+00 +0.000e+00>

     @ q0_zero = +0.000e+00
     @ q0_coeff =  <+0.000e+00 +1.000e+00 +0.000e+00>

     @ q1_zero = +0.000e+00
     @ q1_coeff =  <+0.000e+00 +1.000e+00 +0.000e+00>

     @ q2_zero = +0.000e+00
     @ q2_coeff =  <+0.000e+00 +2.000e+00 +0.000e+00>
end{globalgoal}
define{globalgoal}
     name = resolved_motion_obstacle_avoidance
     @ ControlRate = 1.200e+02
end{globalgoal}
```

Figure C.78: A Global Goal Database flat (ASCII) file.

```
define{knotpoint}
        interval = 0.00e+00 10.00e+00
        position = [<0.00e+00 -1.00e+00 0.00e+00 1.00e+00>
                    <1.00e+00  0.00e+00 0.00e+00 2.00e+00>
                    <0.00e+00  0.00e+00 1.00e+00 0.00e+00>
                    <0.00e+00  0.00e+00 0.00e+00 1.00e+00>]
        velocity = <0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00>
end{knotpoint}
.
.
.
define{knotpoint}
        interval = 34.00e+00 40.00e+00
        position = [<0.00e+00 -1.00e+00 0.00e+00  1.00e+00>
                    <1.00e+00  0.00e+00 0.00e+00 -2.00e+00>
                    <0.00e+00  0.00e+00 1.00e+00  0.00e+00>
                    <0.00e+00  0.00e+00 0.00e+00  1.00e+00>]
        velocity = <0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00>
end{knotpoint
```

Figure C.79: An Trajectory Database flat (ASCII) file.

The default profile is cubic linear (a parabolic velocity profile). Step trajectories can also be specified by inserting `@ type = step_trajectory` into a knotpoint entry. A trajectory segment example appears in figure C.79. Failure to provide a trajectory file is an error if a global goal generation file is present, otherwise the trajectory file is ignored.

C.4 Obstacle Description Files: `theObstacle.obs`

Obstacle description files document a list of obstacles in the scenario. Each obstacle entry incorporates a brief shape description within an informal parameter list. This is followed by the objects current position and an obstacle trajectory (see figure C.80). Again there is no limit on the number of obstacles present. The inclusion of an obstacle file is optional.

```
define{obstacle}
        name = ball
        @ radius = 0.1250e+00
        position = [<1.00e+00    0.00e+00    0.00e+00    1.00e+00>
                    <0.00e+00    1.00e+00    0.00e+00   -0.50e+00>
                    <0.00e+00    0.00e+00    1.00e+00    0.00e+00>
                    <0.00e+00    0.00e+00    0.00e+00    1.00e+00>]
        velocity = <0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00>
        define{knotpoint}
                interval = 0.00e+00 40.00e+00
                position = [<1.00e+00    0.00e+00    0.00e+00    1.00e+00>
                            <0.00e+00    1.00e+00    0.00e+00   -0.50e+00>
                            <0.00e+00    0.00e+00    1.00e+00    0.00e+00>
                            <0.00e+00    0.00e+00    0.00e+00    1.00e+00>]
                velocity = <0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00>
        end{knotpoint}
end{obstacle}
define{obstacle}
.
.
.
end{obstacle}
```

Figure C.80: An Obstacle Database flat (ASCII) file.

Appendix D

RMDAC Gain Selection

Though stability of Seraji's *Direct Adaptive Control* [Seraji, 1989c] (here known as resolved motion decentralized adaptive control, RMDAC) may be assured under appropriate operating conditions, adequate performance remains a function of the free variables of the adaptive control algorithm. The selection of these values determines the controller's rate and quality of convergence to the reference model. The next section will describe the process used to determine the free variables to achieve satisfactory end effector trajectory tracking.

Unlike OSF or RMFC in which gain selection is simplified through linear system stability rules, Configuration Control has a large number of *integral* gains and initial values to assign, including 7 initial values, 14 integral gains, and 12 weighting gains for each of n degrees of freedom or $33n$ values. The recommended method of determining these values is through trial and error [Seraji, 1989c, Glass, 1993], initial values can be left at zero, and the auxiliary term, $\mathbf{d}_i$, merely speeds convergence and is not crucial for stability. The only guideline is to ensure that gains remain positive.

This appendix describes a simple gain selection process that explores the significance of some of these values. The process is not an exhaustive and is merely designed to find a stable region possessing good performance. This experiment employs the reference manipulator configuration, trajectory, and payload. The following sections will explain the experimental method.

The control rate is set at 120 Hz, double the standard 'real time' sampling rate and cosiderably more than the fixed integration period of 0.001 seconds or 1000Hz. In this study weights are initially assigned as if they are LHP stable position and velocity gains for a linear system. [1] and all integral gains have been zeroed except α_{1i} and β_{1i}, set to 1.0. Recalling that each term in the RMDAC force expression is the product of an integration, at least one set of gains must have assigned values in order to track the trajectory.

A velocity weighting experiment retains a constant $\mathbf{w}_{pi}$ while successively doubling the magnitude of $\mathbf{w}_{vi}$. By selecting the greatest stable $[\mathbf{w}_{pi}, \mathbf{w}_{vi}]$ pair a series of integral tests can then be performed. In particular:

[1]Though there is no foundation for this approach it has been found to be a safe first step in selecting position and velocity gains

No.	w_{pi}	w_{vi}	f	α_{1i}	β_{1i}	ν_{1i}	γ_{1i}	λ_{1i}
01	100.0	20.0	0.0	1.0	1.0	0.0	0.0	0.0
02	225.0	30.0	0.0	1.0	1.0	0.0	0.0	0.0
03	400.0	40.0	0.0	1.0	1.0	0.0	0.0	0.0
04	625.0	50.0	0.0	1.0	1.0	0.0	0.0	0.0
05	900.0	60.0	0.0	1.0	1.0	0.0	0.0	0.0
06	1600.0	80.0	0.0	1.0	1.0	0.0	0.0	0.0
07	900.0	120.0	0.0	1.0	1.0	0.0	0.0	0.0
08	900.0	240.0	0.0	1.0	1.0	0.0	0.0	0.0
09	900.0	480.0	0.0	1.0	1.0	0.0	0.0	0.0
10	900.0	480.0	0.0	2.0	1.0	0.0	0.0	0.0
11	900.0	480.0	0.0	4.0	1.0	0.0	0.0	0.0
12	900.0	480.0	0.0	6.0	1.0	0.0	0.0	0.0
13	900.0	480.0	0.0	1.0	2.0	0.0	0.0	0.0
14	900.0	480.0	0.0	1.0	4.0	0.0	0.0	0.0
15	900.0	480.0	0.0	1.0	6.0	0.0	0.0	0.0
16	900.0	480.0	0.0	2.0	2.0	0.0	0.0	2.0
17	900.0	480.0	0.0	2.0	2.0	0.0	0.0	4.0
18	900.0	480.0	0.0	2.0	2.0	0.0	0.0	6.0
19	900.0	480.0	0.0	2.0	2.0	0.0	2.0	2.0
20	900.0	480.0	0.0	2.0	2.0	0.0	4.0	2.0
21	900.0	480.0	0.0	2.0	2.0	0.0	6.0	2.0
22	900.0	480.0	0.0	2.0	2.0	2.0	2.0	2.0
23	900.0	480.0	0.0	2.0	2.0	4.0	2.0	2.0
24	900.0	480.0	0.0	2.0	2.0	6.0	2.0	2.0
25	900.0	480.0	2.0	2.0	2.0	2.0	2.0	2.0
26	900.0	480.0	4.0	2.0	2.0	2.0	2.0	2.0
27	900.0	480.0	6.0	2.0	2.0	2.0	2.0	2.0

Table D.20: A tabulation of initial values (italic) and integral gains (bold) for a set of scenarios. Each integral can use up to three gains, u_0, u_1, and u_2. In this study u_0 is omitted while u_2 never varies, thus the listed gains represent the value of u_1 in each integral.

1. feedback position and velocity gains α_{1i} and β_{1i}.

2. Performance as a function of feedforward gains ν_{1i}, γ_{1i} and λ_{1i}.

3. Performance as a function of auxiliary signal gains d_i.

Since these cartesian controllers are orthonormal no coupling gains (the first term in the coefficient vectors) were required [Seraji, 1989c].The integration (the second term in the coefficient vectors) gains again are somewhat arbitrary, though early trial and error suggested unit gains for the feedback terms and 2.0 for the feedforward mass gain. Offset gains were also excluded (the term u_2 in the adaptive integrals).

The results are divided into four sections. Weights, feedback gains, feedforward gains, and auxiliary gains. The results uniformly show that using all of the integral terms improves trajectory tracking.

As observed in tables D.21 and D.22 as the weights are increased maximum and RMS errors diminish. There is a limit to this tactic, however. As the weights are increased, it eventually becomes necessary to increase the local control rates, possibly because of the ever increasing magnitude of accelerations.

Greater performance improvement may be afforded through the selection of larger velocity weights. The results here indicate that the velocity and position weights are of the same order of magnitude unlike the position and velocity gains in a linear PD-like controller.

It is interesting to observe that the maximum error is slightly higher in case 9 than in case 6. This is some early evidence that the control rate acts as a barrier to large gains. Apparently the initial error is larger in this latter case and the RMS error indicates poorer steady state performance, possibly hinting at an underdamped system. Convergence data from these runs, indicating that case 9 is slower to converge than case 6, confirms the perception that the system is less stable with the larger velocity weight.

As expected, tables D.24 and D.23 indicate that feedback gains must be present to produce reasonable tracking, and that as these gains increase tracking improves. Caution must be exercised with these and other gains, however, in that excessively large gains may result in integrator wind-up and poorer R.M.S. performance.

Tables D.25, D.26 and D.27 indicate that increasing feedforward gains improves tracking. While the basic trend is towards improved maximum and RMS performance, an inconsistent increase in both demonstrated in series 24 indicates that it is possible to apply excess feedforward position gains - leading to a decrease in maximum and RMS performance. It should be noted that convergence times were better for this run, however. This may indicate that as gains increase the system becomes underdamped, oscillating about the desired trajectory and converging rapidly.

Enabling the auxiliary gains improves performance as predicted by Seraji and shown in table D.28. It is interesting to note, however, that initially the Maximum and RMS errors are elevated with the introduction of another integrating term. This could be attributed to additional integrator wind up in the first seconds of execution. Ultimately, as more information is used to augment the auxiliary signal, error is substantially reduced.

Even with carefully selected gains, the reference trajectory presents some difficulties for the Jacobian Transpose Algorithm. The first segment's velocity and acceleration vectors have small magnitudes and are inclined only 15deg to the x-axis. Since the manipulator is initially homed along this axis, the algorithm produces small cartesian forces and even smaller torques. The effect of integrator windup is compounded by the six integrations - all of which depend on cartesian error. The overshoot serves to drive the system, making any instability more noticeable. Particularly in these low aspect segments, RMDAC showed considerable sensitivity to the feedforward term, presumably because this term would results in larger force requirements

w_{pi}	No.	Step Response		Reference Trajectory	
		$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$	$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$
100	01	2.610e-01	1.026e-01	2.248e-01	9.577e-02
225	02	2.302e-01	9.160e-02	1.839e-01	7.698e-02
400	03	2.103e-01	8.628e-02	1.489e-01	6.113e-02
625	04	1.970e-01	8.438e-02	1.315e-01	5.670e-02
900	05	1.868e-01	8.335e-02	1.167e-01	4.411e-02
1600	06	1.713e-01	8.671e-02	9.759e-02	3.594e-02

Table D.21: Tabulation of position and orientation RMS L_2 error with variation in error weights,$\mathbf{W}_p$.

earlier in the trajectory. The auxiliary signal, too, provided a significant improvement in performance.

While earlier studies indicated that using initial values could improve performance. Some of these values, particularly the feedback gains, would be difficult to estimate. Indeed, only $\mathbf{q}_2$, the manipulator inertia parameter, represents an easily estimated initial value.

Without more rigorous methods for establishing integral gains, we are left with a trial and error procedure. The procedure shown here, while not exhaustive, represents a reasonable method for searching the integral gain space. This process has been used to establish a 'best gains' RMDAC gain file for the trajectory, the 'step' response for which appears in figure D.81.

D.1 Control Rates and RMDAC

Given that RMDAC is designed with the assumption that the manipulator parameters vary slowly through time, control rate seems to play a role in determining the stability of a system that operates at the edge or outside of this assumption. Over the course of the trial and error process, large integral gains (or large position and velocity weights) sensitized the controller to large trajectory errors, often leading to instability. Increased control rates were observed to counter large integral gains, in effect reducing integrator wind up through smaller time steps.

w_{vi}	No.	Step Response		Reference Trajectory	
		$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$	$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$
60.0	03	2.103e-01	8.628e-02	1.489e-01	6.113e-02
120.0	07	1.572e-01	8.449e-02	1.161e-01	4.733e-02
240.0	08	1.391e-01	8.971e-02	1.112e-01	3.876e-02
480.0	09	1.520e-01	1.028e-01	1.070e-01	4.362e-02

Table D.22: Tabulation of position and orientation RMS L_2 error as a function of variation of the weight $\mathbf{W}_v$ given $w_{pi} = 900$.

α_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$	$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$
1.0	09	1.520e-01	1.028e-01	1.070e-01	4.362e-02
2.0	10	1.437e-01	1.006e-01	8.876e-02	3.614e-02
4.0	11	1.336e-01	9.827e-02	7.128e-02	2.950e-02
6.0	12	1.291e-01	9.600e-02	6.286e-02	2.668e-02

Table D.23: Tabulation of position and orientation RMS L_2 error as a function of the Feedback integral gain, α_{1i} given $\beta_{1i} = 1.0$.

β_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$	$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$
1.0	09	1.520e-01	1.028e-01	1.070e-01	4.362e-02
2.0	13	1.579e-01	1.024e-01	1.045e-01	4.767e-02
4.0	14	1.629e-01	1.053e-01	1.038e-01	5.246e-02
6.0	15	1.633e-01	1.073e-01	1.012e-01	5.003e-02

Table D.24: Tabulation of position and orientation RMS L_2 error as a function of the Feedback integral gain, β_{1i} given $\alpha_{1i} = 1.0$.

λ_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$	$\sqrt{\lvert\mathbf{x}_e\rvert^2}$	$\sqrt{\theta^2}$
2.0	16	1.496e-01	9.837e-02	7.166e-02	3.558e-02
4.0	17	1.496e-01	9.837e-02	6.509e-02	3.311e-02
6.0	18	1.496e-01	9.837e-02	6.004e-02	3.152e-02

Table D.25: Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, λ_{1i} given $\nu_{1i} = 0.0$, $\gamma_{1i} = 0.0$.

γ_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$	$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$
0.0	16	1.496e-01	9.837e-02	7.166e-02	3.558e-02
2.0	19	1.496e-01	9.837e-02	5.403e-02	3.483e-02
4.0	20	1.496e-01	9.837e-02	4.631e-02	3.217e-02
6.0	21	1.496e-01	9.837e-02	4.155e-02	3.124e-02

Table D.26: Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, γ_{1i}, given $\nu_{1i} = 0.0$, $\lambda_{1i} = 2.0$.

ν_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$	$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$
0.0	19	1.496e-01	9.837e-02	5.403e-02	3.483e-02
2.0	22	1.243e-01	1.057e-01	9.186e-03	1.632e-02
4.0	23	5.775e-01	3.390e-01	9.670e-03	1.602e-02
6.0	24	5.355e-01	2.853e-01	6.714e-03	1.634e-02

Table D.27: Tabulation of position and orientation RMS L_2 error as a function of the Feedforward integral gain, ν_{1i} given $\gamma_{1i} = 2.0$, $\lambda_{1i} = 2.0$.Note that cases 23 and 24 were unstable in the step response.

δ_{1i}	No.	Step Response		Reference Trajectory	
		$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$	$\sqrt{\overline{\lvert \mathbf{x}_e \rvert^2}}$	$\sqrt{\overline{\theta^2}}$
0.0	22	1.243e-01	1.057e-01	9.186e-03	1.632e-02
2.0	25	1.222e-01	9.658e-02	1.112e-02	1.337e-02
4.0	26	1.211e-01	9.373e-02	6.908e-03	1.311e-02
6.0	27	1.203e-01	9.234e-02	5.525e-03	1.306e-02

Table D.28: Tabulation of position and orientation RMS L_2 error as a function of the auxiliary integral gain, δ_{1i}, given $\gamma_{1i} = 2.0$, $\lambda_{1i} = 2.0$.

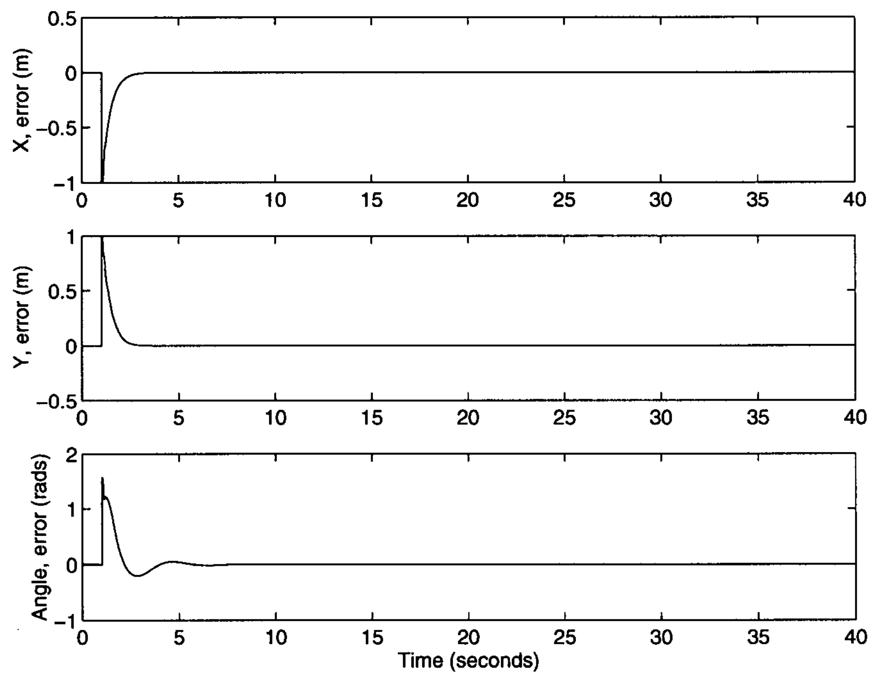


Figure D.81: Absolute end effector step response under 'Best Gains'.

Appendix E

Global Goal Generators Compared

In chapter 6, Configuration Control or Resolved Motion Decentralized Adaptive Control was proposed as an ideal global goal generator for a multiagent system. RMDAC exhibited attractive characteristics such as manipulator model independence, low computational requirements, and predictably stable end effector behaviour. Yet any one of a number of *possible* global goal generators based on the Jacobian Transpose methods described in 4 could have been chosen. Determining the most suitable goal generator required a detailed review and comparison of these candidate global goal generation methods. This appendix describes the process through which this generator was selected and provides a performance comparison between two systems: the Operational Space Formulation and Resolved Motion Force Control.

E.1 The Operational Space Formulation

In 1985, O. Khatib introduced a novel method of manipulator control, the Operational Space Formulation (OSF), to simplify the obstacle avoidance problem [Khatib, 1985]. In this method, exact end effector trajectory tracking was guaranteed by computing the precise cartesian forces required to drive the end effector along a desired trajectory. This was achieved through the use of a feedforward cartesian (operational space) manipulator model in which estimates of the manipulator's parameters were used in conjunction with a twice differentiable goal trajectory, $\mathbf{x}_d(t)$ to construct the desired force vector, $\mathbf{f}_d$ or:

$$\mathbf{f}_d = \hat{\mathbf{M}}(\mathbf{x})\mathbf{f}^*_m + \hat{\mathbf{N}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \hat{\mathbf{P}}(\mathbf{x}) \tag{E.414}$$

The hat superscript indicates parameter estimates and $\mathbf{f}^*_m$ is defined as the acceleration necessary to move a *unit mass*, the identity matrix $\mathbf{I}_m$, along a prescribed trajectory,$\mathbf{x}_d(t)$:

$$\mathbf{f}^*_m = \mathbf{I}_m\ddot{\mathbf{x}}_d + \mathbf{K}_p(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_d(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) \tag{E.415}$$

Now, if the parameter estimates are exact, equation (E.417) reduces to:

$$\ddot{\mathbf{x}}_e(t) + \mathbf{K}_v\dot{\mathbf{x}}_e(t) + \mathbf{K}_p\mathbf{x}_e(t) = 0 \tag{E.416}$$

where $\mathbf{x}_e = \mathbf{x}_d - \mathbf{x}$. Conversely, if the parameter estimates are not exact it can be shown that this linear system is driven by parameter errors:

$$\hat{\mathbf{M}}(\mathbf{x}_d)^{-1}\left[\mathbf{M}_e(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{N}_e(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{P}_e(\mathbf{x})\right] = \ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e \tag{E.417}$$

where $\mathbf{M}_e$,$\mathbf{N}_e$, and $\mathbf{P}_e$ are the parameter errors:

$$\mathbf{M}_e(\mathbf{x}) = \mathbf{M}(\mathbf{x}) - \hat{\mathbf{M}}(\mathbf{x}) \tag{E.418}$$

$$\mathbf{N}_e(\mathbf{x},\dot{\mathbf{x}}) = \mathbf{N}(\mathbf{x},\dot{\mathbf{x}}) - \hat{\mathbf{N}}(\mathbf{x},\dot{\mathbf{x}}) \tag{E.419}$$

$$\mathbf{P}_e(\mathbf{x}) = \mathbf{P}(\mathbf{x}) - \hat{\mathbf{P}}(\mathbf{x}) \tag{E.420}$$

Though parameter estimation is difficult particularly with time varying payloads, the chief disadvantage of OSF for multiagent control centers on the centralized computation of the feedforward model, from [Khatib, 1985]:

$$\mathbf{M}(\mathbf{x}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{D}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}) \tag{E.421}$$

$$\mathbf{N}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} = \mathbf{J}^{-T}\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{M}(\mathbf{x})\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} \tag{E.422}$$

$$\mathbf{p}(\mathbf{x}) = \mathbf{J}^{-T}\mathbf{g}(\mathbf{q}) \tag{E.423}$$

Further computation is required to determine the jacobian pseudoinverse for redundant manipulators and the avoidance of kinematic singularities. Indeed, substituting a precise dynamic model into the configuration space robot equation it can be shown that OSF is equivalent to RMAC:

$$\mathbf{D}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g} = \mathbf{J}^T\left[\mathbf{J}^{-T}\mathbf{D}\mathbf{J}^{-1}\mathbf{f}_m^* + \mathbf{J}^{-T}\mathbf{C}\dot{\mathbf{q}} - \mathbf{J}^{-T}\mathbf{D}\mathbf{J}^{-1}\dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}^{-T}\mathbf{g}\right] \tag{E.424}$$

$$\ddot{\mathbf{q}} = \mathbf{J}^{-1}\left[\mathbf{f}_m^* - \dot{\mathbf{J}}\dot{\mathbf{q}}\right] \tag{E.425}$$

$$\ddot{\mathbf{q}} = \mathbf{J}^{-1}\left[(\ddot{\mathbf{x}}_d + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e) - \dot{\mathbf{J}}\dot{\mathbf{q}}\right] \tag{E.426}$$

a result identical to equation (3.67) [An, 1989]. Thus OSF relies on both accurate manipulator geometric and mass parameters to achieve precise trajectory tracking. Such centralized models defeat the purpose of the decentralized, multiagent structure discussed thus far.

E.1.1 Theoretical Performance

Selection of the feedback gains $\mathbf{K}_p$ and $\mathbf{K}_d$ such that the roots of the characteristic equation, (E.416), reside in the left half plane of frequency space ensures exponential asymptotic stability. As we have discussed, imprecise parameter estimates corrupt the stability properties of this system.

Furthermore, it is important to distinguish between the stability of the cartesian and configuration state spaces. Within fully constrained manipulation systems where $\mathbf{J}^T$ is full rank, exponential stability in task space is equivalent to exponential stability in configuration space. However, the same is not true for redundant systems in which $\mathbf{J}^T$ is usually not square. Recalling the definition of redundancy from chapter 3, end effector stability gives no indication of stability in configuration space. Consider the configuration space expression of the robot equation and OSF:

$$(\mathbf{D} - \hat{\mathbf{D}})\ddot{\mathbf{q}} + (\mathbf{C} - \hat{\mathbf{C}})\dot{\mathbf{q}} + (\mathbf{g} - \hat{\mathbf{g}}) = \mathbf{J}^T\hat{\mathbf{M}}\left[\ddot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e + \mathbf{K}_d\dot{\mathbf{x}}_e\right] \tag{E.427}$$

Given inaccurate parameter estimates, this expression relates two distinct dynamic systems: one in configuration space and the other in task space. If these parameter errors reside in the null space of the Jacobian $N(\mathbf{J})$ then the right hand side represents a stable system equivalent to equation (E.416) while the left hand side exhibits dynamics equivalent to unforced or *free dynamics* in $N(\mathbf{J})$:

$$(\mathbf{D} - \hat{\mathbf{D}})\ddot{\mathbf{q}} + (\mathbf{C} - \hat{\mathbf{C}})\dot{\mathbf{q}} + (\mathbf{g} - \hat{\mathbf{g}}) = 0. \tag{E.428}$$

If isolated in $N(\mathbf{J})$, there is no requirement that these dynamics exhibit any form of stability. Realistically, however, joint friction would act to ultimately dampen motion in $N(\mathbf{J})$.

As mentioned above, redundant manipulators present identical difficulties to OSF as in RMAC. Without special measures such as singularity avoidance subtasks applied to the Null space of the jacobian [Ballieul, 1985], even the pseudoinverse becomes numerically unstable near kinematic singularities.

E.1.2 Simulation

Since OSF is provably identical to RMAC and computationally less burdensome, it is sufficient to demonstrate RMAC. In doing so, this simulation not only demonstrates the performance of both cartesian control methods, but also provides a common benchmark against traditional redundant manipulator control systems.

Of course, from the information flow standpoint, RMAC differs from OSF in the context of multiagent control. RMAC transmits uniquely specified joint torques for each actuator rather than broadcasting a common global goal couplet as would be used to implement OSF. RMAC is, therefore, an explicit coordination mechanism for multiagent control. This is significant, for while OSF *appears* to be an implicit coordination technique, broadcasting global goal couplets to the agent team, the feedforward dynamic model defeats the value of this decentralized approach. So, in truth, RMAC represents a more precise description of coordination within OSF.

Over the reference trajectory, RMAC demonstrates excellent trajectory tracking performance, depicted in figureE.82 with exactly known manipulator parameters . In figures E.83, the manipulator configuration at each knotpoint of the reference trajectory is shown. Since the desired trajectory consumes only three degrees of freedom, the manipulator exhibits free motion in $N(\mathbf{J})$, the remaining seven degrees of freedom. Unconstrained motion in $N(\mathbf{J})$ is usually characterized by jumbled or convoluted manipulator configurations as exemplified in this figure.

E.1.3 Discussion

Though OSF is an effective goal generation mechanism, the centralized computation required defeats the decentralized structure of multiagent control. The computation of the jacobian inverse or pseudoinverse requires a centralized model of the manipulator and thus suffers from the same drawbacks as RMAC. This problem can be solved only through the removal of any reliance upon a centralized manipulator dynamic model in the production of $\mathbf{f}_d$. Indeed, the separability property of the cartesian and configuration dynamics implies that dynamic modelling in cartesian space need not model manipulator dynamics at all, but should models some *projection* of the manipulator in task space. The following sections will review a simple solution, Resolved Motion Force Control, that removes the manipulator, but not the payload from the goal generation process.

E.2 Resolved Motion Force Control

Coincident with the introduction of OSF, Wu and Paul suggested a simple form of Jacobian Transpose control **Resolved Motion Force Control** (RMFC) [Wu, 1982]. In its simplest form RMFC applies a computed acceleration, $\ddot{\mathbf{x}}_a(t)$, to a known payload $\mathbf{M}_p$. Newtons second law applied to the payload:

$$\mathbf{f}_d(t) = \mathbf{M}_p\ddot{\mathbf{x}}_a(t) \tag{E.429}$$

where the acceleration, $\ddot{\mathbf{x}}_a(t)$, is regulated over a desired trajectory, $\mathbf{x}_d(t)$:

$$\ddot{\mathbf{x}}_a(t) = \mathbf{K}_v\dot{\mathbf{x}}_e(t) + \mathbf{K}_p\mathbf{x}_e(t) + \ddot{\mathbf{x}}_d(t) \tag{E.430}$$

and $\mathbf{x}_e = \mathbf{x}_d - \mathbf{x}$. Given that the manipulator inertia includes the payload mass or $\mathbf{M}'(\mathbf{x}) = \mathbf{M}(\mathbf{x}) + \mathbf{M}_p$, the error system for RMFC becomes:

$$\mathbf{M}_p^{-1}\left[\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{N}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{P}(\mathbf{x})\right] = \ddot{\mathbf{x}}_e + \mathbf{K}_v\dot{\mathbf{x}}_e + \mathbf{K}_p\mathbf{x}_e \tag{E.431}$$

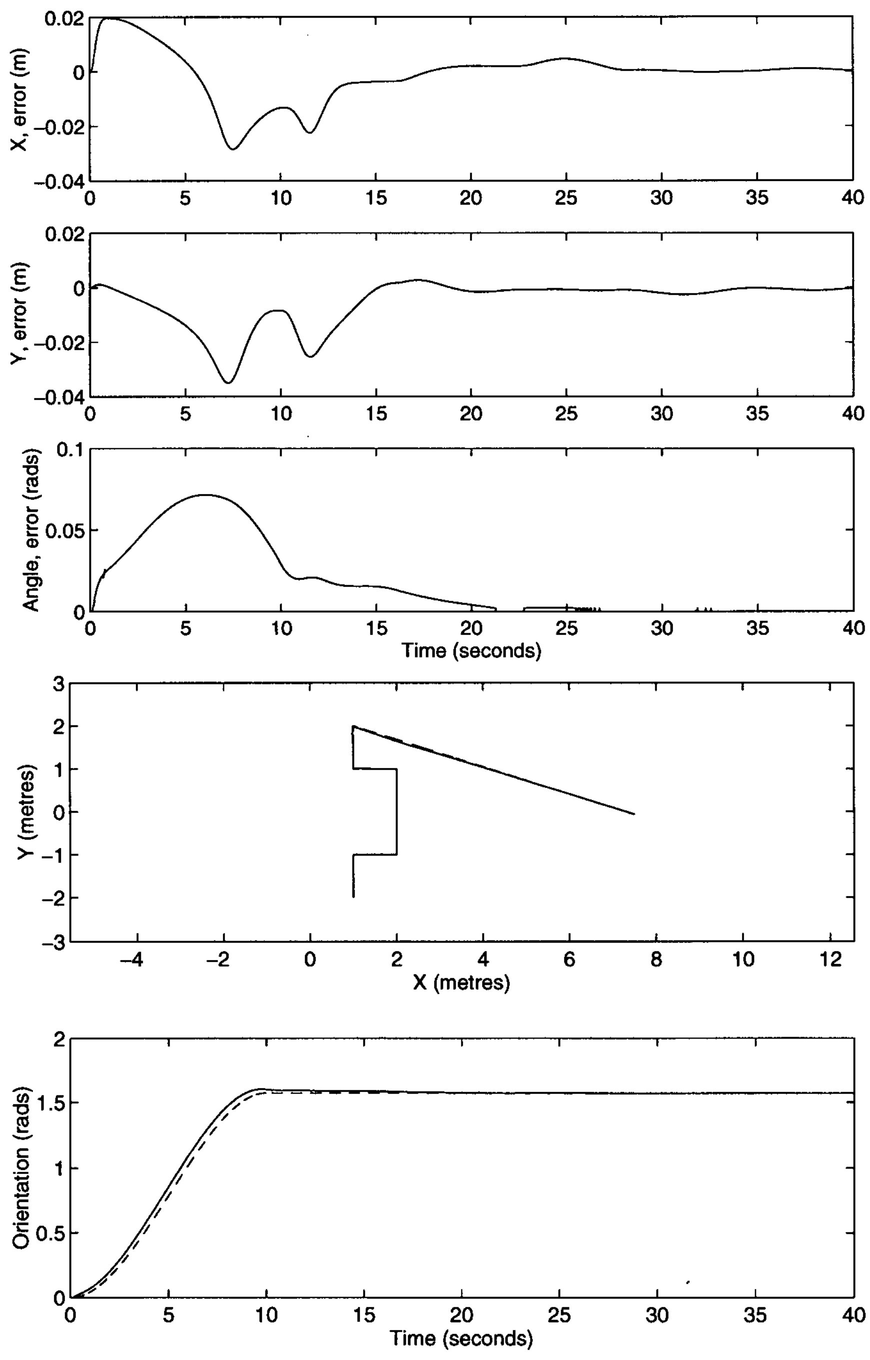


Figure E.82: RMAC end effector trajectory tracking performance of the reference trajectory with exact manipulator parameter estimates.

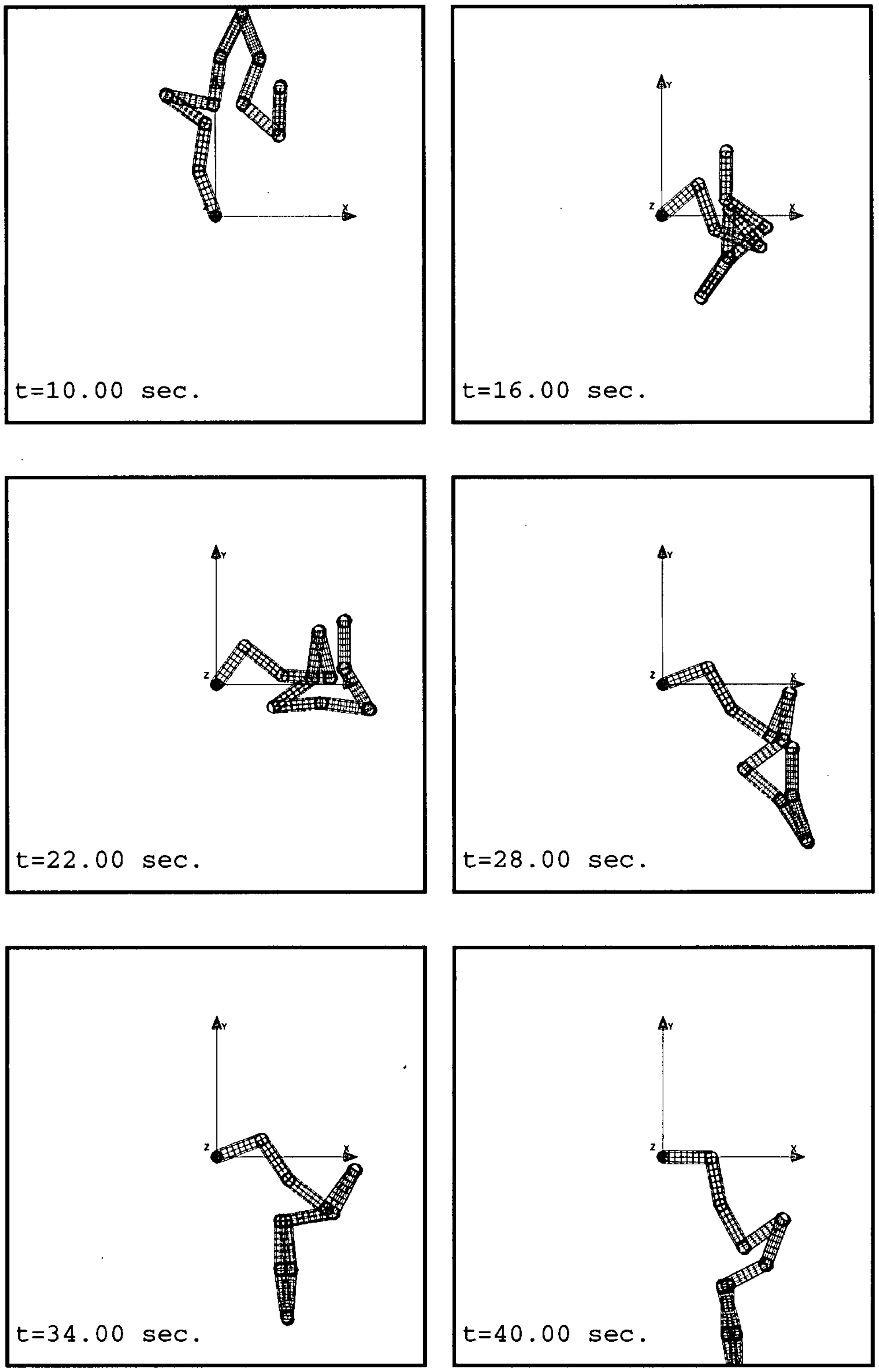


Figure E.83: RMAC (equivalent to OSF) configuration history of the reference trajectory for the reference manipulator and payload. Note that despite precise trajectory tracking, the manipulator adopts convoluted configurations, a clear indication of free motion in $N(\mathbf{J})$.

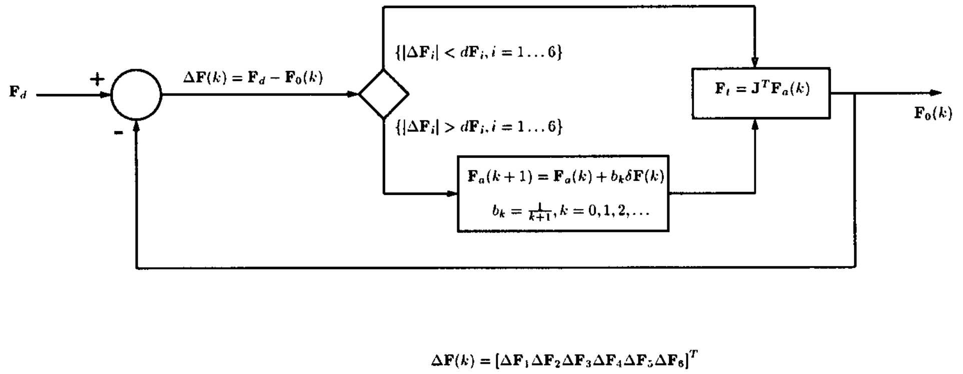

$\mathbf{F}_d$
+
-
$\Delta\mathbf{F}(k) = \mathbf{F}_d - \mathbf{F}_0(k)$
$\{|\Delta\mathbf{F}_i| < d\mathbf{F}_i, i = 1\ldots6\}$
$\{|\Delta\mathbf{F}_i| > d\mathbf{F}_i, i = 1\ldots6\}$
$\mathbf{F}_t = \mathbf{J}^T\mathbf{F}_a(k)$
$\mathbf{F}_a(k+1) = \mathbf{F}_a(k) + b_k\delta\mathbf{F}(k)$
$b_k = \frac{1}{k+1}, k = 0, 1, 2, \ldots$
$\mathbf{F}_0(k)$
$\Delta\mathbf{F}(k) = [\Delta\mathbf{F}_1\Delta\mathbf{F}_2\Delta\mathbf{F}_3\Delta\mathbf{F}_4\Delta\mathbf{F}_5\Delta\mathbf{F}_6]^T$

Figure E.84: The Force Convergent Controller

A *massless* manipulator reduces the system to the familiar:

$$\ddot{\mathbf{x}}_e(t) + \mathbf{K}_v\dot{\mathbf{x}}_e(t) + \mathbf{K}_p\mathbf{x}_e(t) = 0 \tag{E.432}$$

Now tracking error will converge asymptotically to zero if $\mathbf{K}_v$ and $\mathbf{K}_p$ are chosen such that the characteristic has negative real parts. As the manipulator mass matrix, $\mathbf{M}(\mathbf{x})$, approaches and exceeds $\mathbf{M}_p$ the error system (E.432) becomes driven by manipulator and payload dynamics.

While the manipulator will track the desired trajectory if $\mathbf{M}_p$ is of sufficient magnitude, typical manipulator-payload mass ratios (approximately 10:1) require dynamic compensation to guarantee convergence. Rather than employing an explicit manipulator dynamic model, Wu treated the manipulator as a 'black box' of unknown structure. By adopting a *Robbins-Monro stochastic estimator* and a wrist force measurement through a force sensor, the desired end effector forces, $\mathbf{f}_d$, could be modulated to produce the necessary the forces to drive the payload along $\mathbf{x}_d$.

The Stochastic Estimator Applied to RMFC

The arm is modelled as an unknown function

$$\mathbf{f}_o = f_{\text{arm}}(\mathbf{f}_a) \tag{E.433}$$

where $\mathbf{f}_o$, the observed force in base coordinates, is sampled through a wrist sensor.

Initially, the applied force, $\mathbf{f}_a$, is set to compensate for gravitational payload forces. At the kth timestep the required end effector forces are transformed into joint torques through the familiar expression:

$$\tau(k) = \mathbf{J}^T\mathbf{f}_a(k) \tag{E.434}$$

Once these torques have been applied, a sample from the force sensor, $\mathbf{f}_o(k)$, is taken and a cartesian force error computed:

$$\Delta\mathbf{f}(k) = \mathbf{f}_d - \mathbf{f}_o(k) \tag{E.435}$$

where $\Delta\mathbf{f} = [\Delta \mathrm{f}_1 \; \Delta \mathrm{f}_2 \; \ldots \; \Delta \mathrm{f}_n]^T$.

If $|\Delta\mathbf{f}_i|$ becomes greater than a prescribed threshold df_i, for any $i : 1, \ldots, n$ then the applied force vector is modified according to a simple *Force Convergence Controller* (FCC):

$$\mathbf{f}_a(k+1) = \mathbf{f}_a(k) + b_k \Delta\mathbf{f}(k) \tag{E.436}$$

$$b_k = \frac{1}{k+1} \tag{E.437}$$

Theoretically the Robbins Monroe stochastic estimator is an infinite series ($k \rightarrow \inf$). Practically, however, as long as $|\Delta\mathbf{f}_i > df_i|$, FCC increments k from zero up to a finite limit or until $|\Delta\mathbf{f}_i| < df_i$ at which point no further correction is applied for the rest of the control period. In practice [Wu, 1982] found that as little as two correction cycles, k, were sufficient to guarantee good trajectory tracking.

Through FCC, RMFC effectively dispenses with a central geometric representation of the manipulator and opens the possibility of nearly model-free goal regulation.

E.2.1 Theoretical Performance

The convergence properties of stochastic estimation methods are best summarized in Fukunaga [Fukunaga, 1990]:

> Stochastic approximation can be used for parameter estimation in pattern recognition, and convergence is guaranteed under very general circumstances. It is usually difficult, however, to discuss the rate of convergence.

So while convergence is generally guaranteed in RMFC, the rate of convergence is cannot be easily predicted. This is particularly true if there are payload inertia errors. An, Atkeson, and Hollerbach [An, 1989] report that stiffness controllers (of which OSF is one) can sustain modelling errors of 50

E.2.2 Simulation

In the following studies uncompensated and compensated RMFC will be demonstrated. An additional issue is that given a nominal control rate, the adoption of FCC effectively multiplies the applied control rate. Thus a nominal control rate of R with an FCC iteration limit of k generates an applied rate, $R' = (k+1)R$ (e.g. $R = 120Hz$, $k = 3$, $R' = 480Hz$). Conversely, if the applied control rate is fixed (typical of real controllers), then the nominal control rate must be reduced. Since the desired end effector forces are evaluated at the nominal rate, real world RMFC controllers might operate on the edge of stability despite FCC.

Simulated Uncompensated Response

In figure E.85 the reference manipulator carries the reference payload using the basic RMFC controller of equation (E.429) at a control rate of 120 Hz. Despite the small relative mass of the payload to the manipulator and the lack of compensation, the end effector tracks the trajectory reasonably well without force convergent control. Figure E.85 clearly shows the severe initial error due in large part to the difficulties of the first trajectory segment outlined earlier.

Simulated Compensated Response

In figure E.86 the reference manipulator again carries the reference payload using employing RMFC with Force Convergent control at a nominal control rate of 120 Hz and an FCC iteration limit of 3 (forming an actual control rate of 480Hz). FCC compensates for severe initial errors successfully as depicted in figure E.86.

Control Rates and FCC

Given unbounded control rates, RMFC and FCC demonstrate excellent trajectory tracking. However, in practice control rates are generally bounded by the computational load on the control system. A better comparison between compensated and uncompensated strategies assumes a maximum control rate of 120Hz, in which case an FCC limit of 3 iterations fixes the nominal RMFC control rate to 30Hz. The results from this run are depicted in figures E.87,E.88.

Not surprisingly, trajectory tracking performance is poor relative to the *nominal* 120Hz run, but remains tolerable. Orientation appears more sensitive to control rate than position control for the same gains.

E.2.3 Discussion

RMFC is simple and exhibits good end effector trajectory tracking. However, both the MMS and Wu's original implementation of RMFC are only simulations using 'perfect' noise-free force sensors. The performance of RMFC in real world applications is unknown. Practically speaking, the controller's reliance upon force sensor data and the necessity of an accurately known payload represent significant barriers to comparable

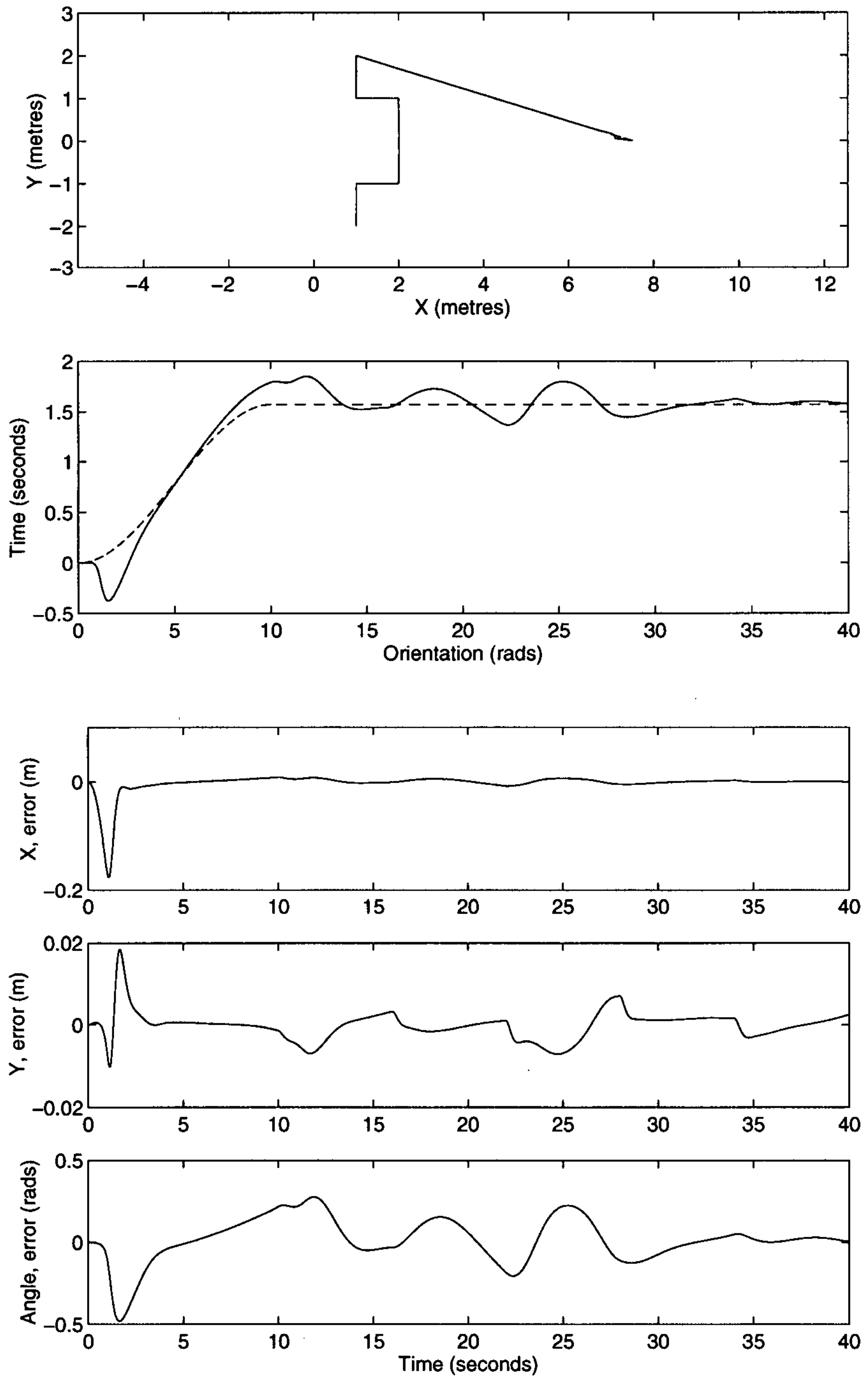


Figure E.85: End effector trajectory error performance during the reference trajectory *without* Force Convergent Control at 120hz.The manipulator carries the reference payload and the controller employs PD gains of $\mathbf{K}_p = 100$ and $\mathbf{K}_d = 20$.

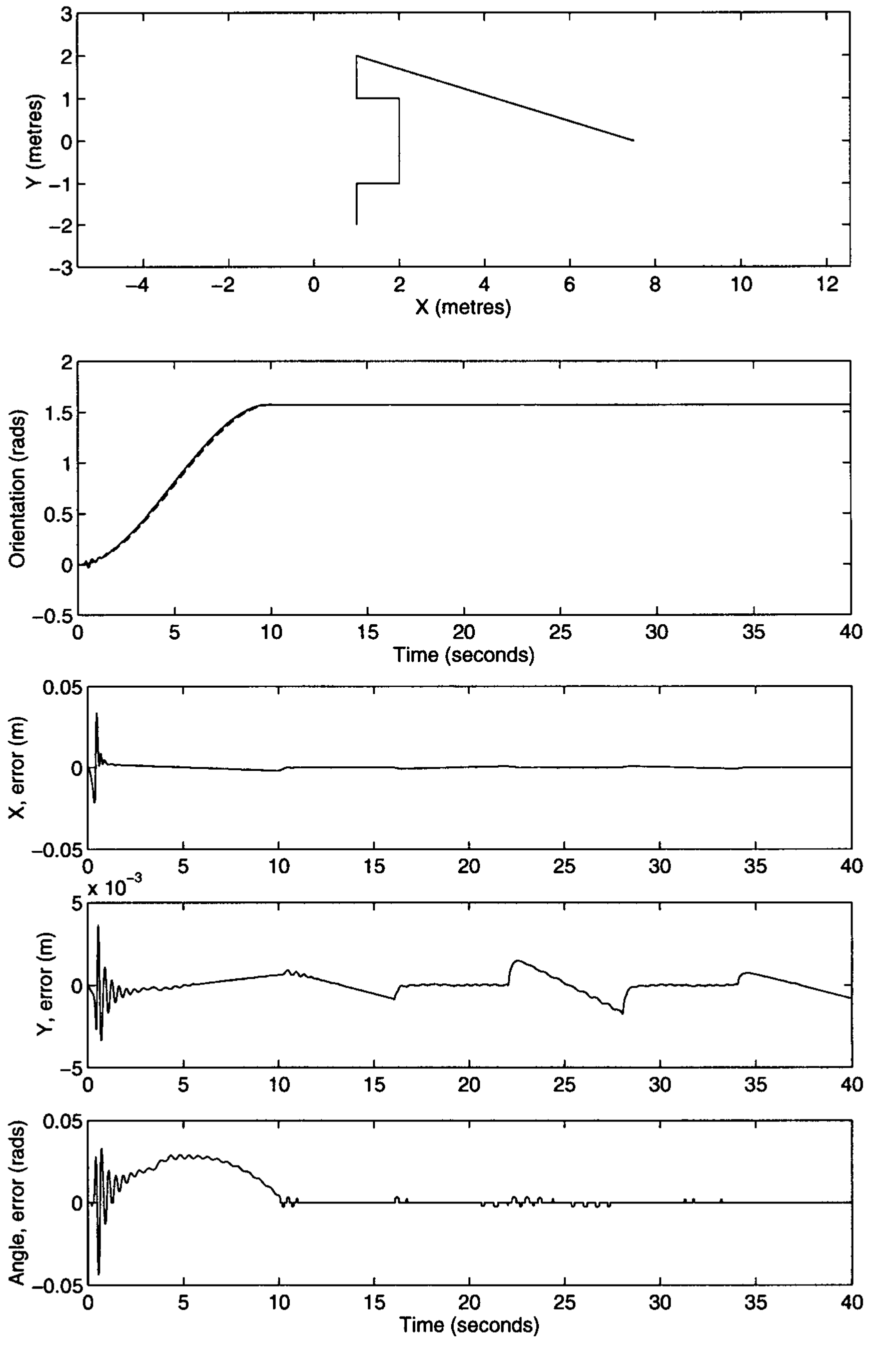


Figure E.86: End effector trajectory tracking performance during the reference trajectory under Force Convergent Control (RMFC nominal rate of 120hz, actual control rate 480Hz).

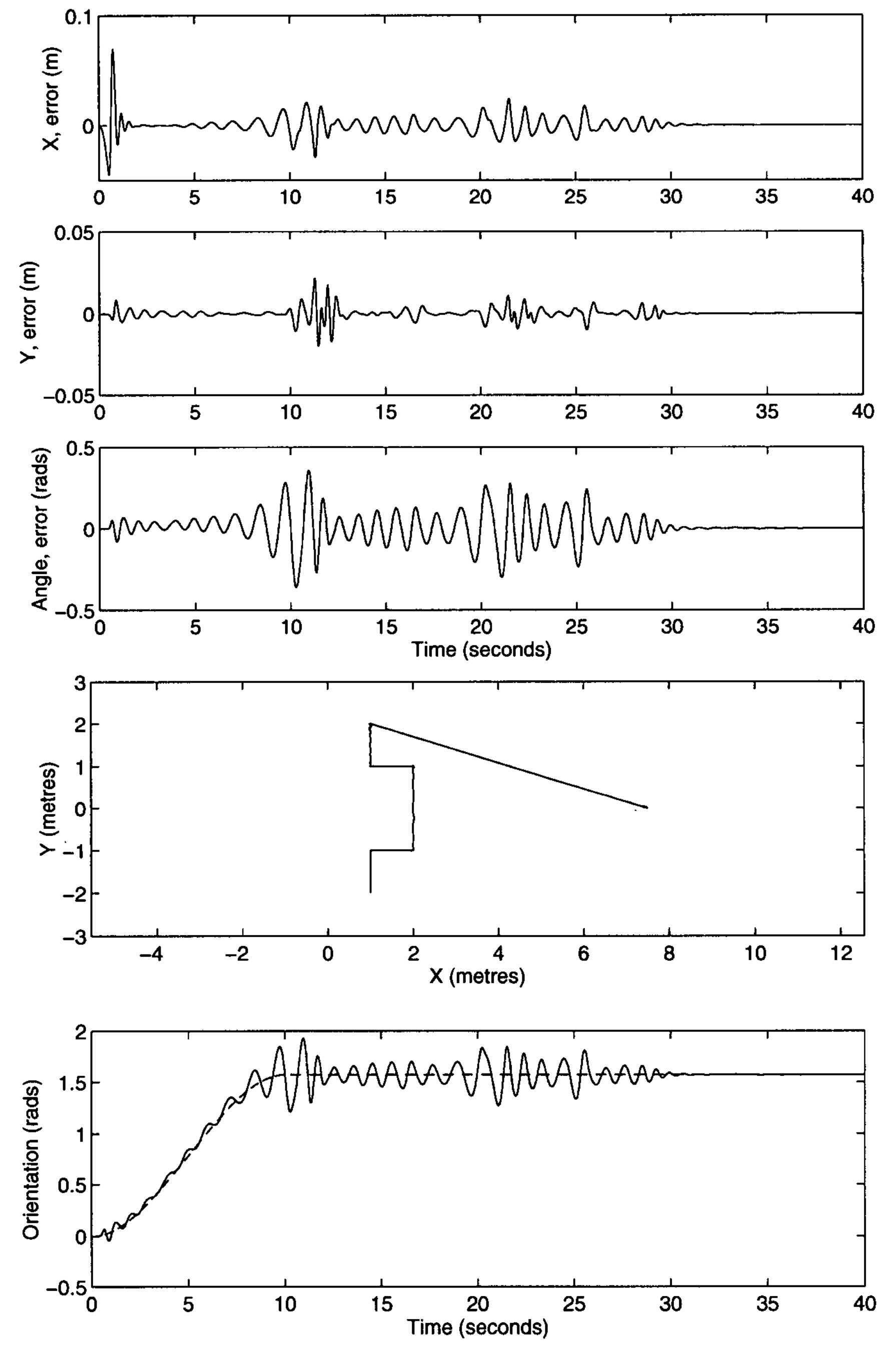


Figure E.87: End effector trajectory performance during the reference trajectory under Force Convergent Control(RMFC nominal rate of 30hz, actual control rate 120Hz).

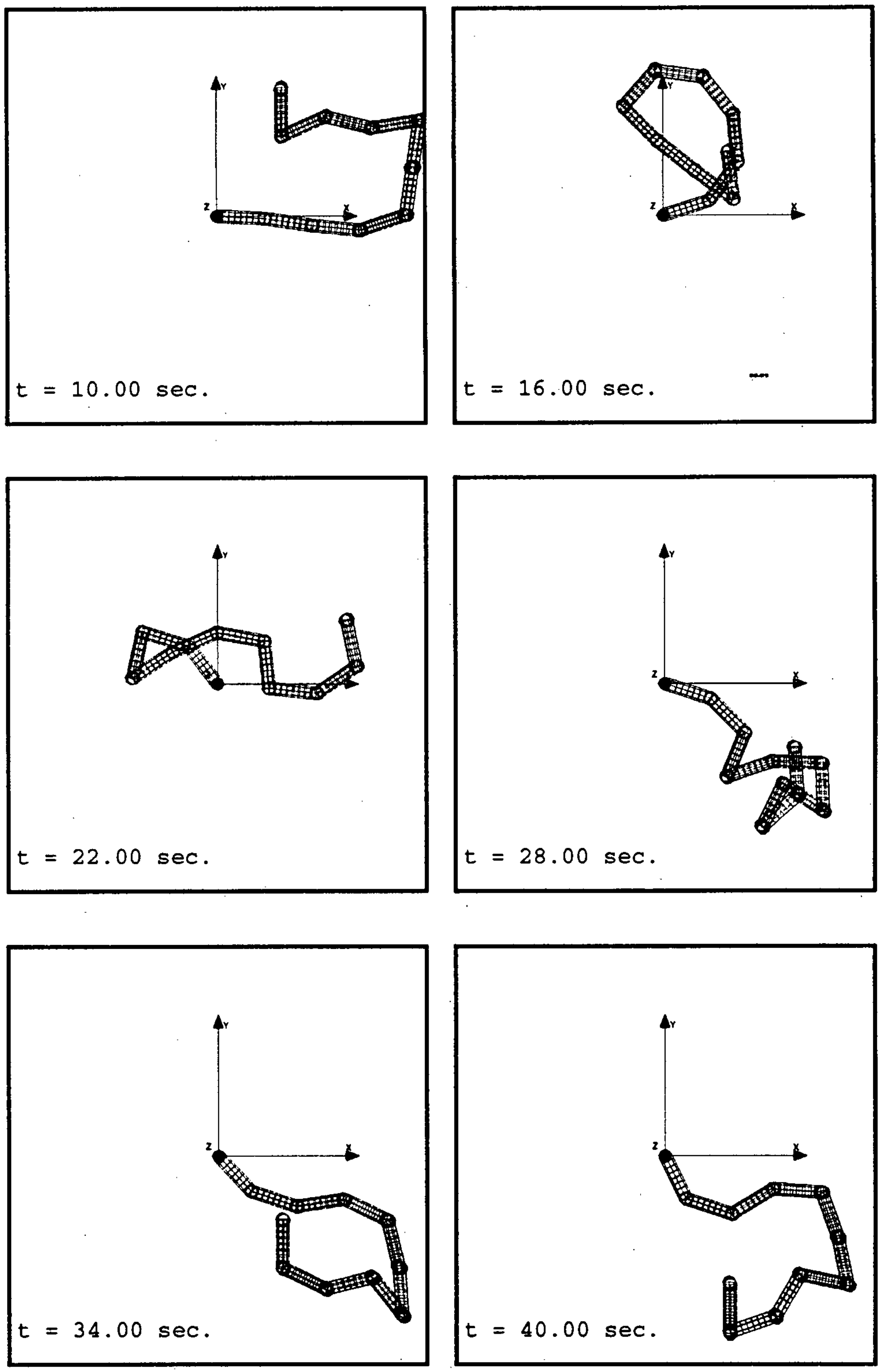


Figure E.88: Trajectory tracking history for the reference trajectory under Force Convergent Control (RMFC nominal rate of 30hz, actual control rate 120Hz).

real world performance. Of course, force sensing is not unusual for contact tasks where desired force magnitudes are often of limited dynamic range (e.g. fine manipulation or micromanipulation). In these situations, highly sensitive, fast response semiconductor strain gages are the preferred force sensing method. Though random noise is considered negligible in semiconductor strain gage sensors [de Silva, 1989], vibration due to unmodelled dynamics in the robot or payload is not insignificant. An, Atkeson, and Hollerbach's study on model based control [An, 1989] identify sensor vibration and drift as the two major sources of semiconductor sensor error. Though it seems likely the combination of real world sensor noise, drift, and miscalibration could significantly erode tracking performance, additional signal processing or mechanical damping could compensate for these effects.

The most serious drawback, however, is the adherance to a payload model. Though the system is reasonably robust to payload errors, a priori payload models are not likely in unstructured environments. It is true that some form of learning phase could be adopted to determine the payload mass, but this defeats the purpose of both a known payload and force convergent control – if the payload inertia can be learned, then why not the manipulator inertia? RMDAC does exactly this, through the use of a model reference adaptive controller.