

Task-level Control for Networked Telerobotics

by

Kevin M. O'Brien

S.B., Mechanical Engineering, University of California at Berkeley (1994)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

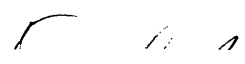
at the


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 23, 1996

© Massachusetts Institute of Technology, 1996. All Rights Reserved.


Author
Mechanical Engineering
May 23, 1996


Certified by
David Brock, Research Scientist
MIT Artificial Intelligence Laboratory
Thesis Supervisor


Accepted by
Ain A. Sonin
Mechanical Engineering
Chairperson, Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUN 27 1996 Eng.

LIBRARIES

Task-level Control for Networked Telerobotics

by

Kevin M. O'Brien

Submitted to the Department of Mechanical Engineering on May 23, 1996, in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

Telerobotic and networked robotic systems often encounter significant time delays in relation to their dominant dynamics, which causes well known stability problems. Good control performance can be achieved through task-level control and by allowing the human operator's intelligence to guide complex and unpredictable tasks. Supervisory control techniques were combined with task-level control theory to arrive at task-level supervisory control, a systematic approach to controlling robots under time delays in excess of 40 times their dominant time constants. Guidelines for designing a set of primitive tasks for various time delays and ranges of actions were presented. Sufficiency and completeness of the primitive sets along with the logic used to build complex tasks from the sets was discussed. An application of these guidelines to a real system was made and the results show effective robot control especially under network latencies, where the bulk of the time delay occurs in the forward path. Finally, some applications of such a system were presented.

<http://www.ai.mit.edu/projects/webot/robot>

Thesis Supervisor: David Brock

Title: Research Scientist, MIT Artificial Intelligence Laboratory

Acknowledgments

Throughout my time as a graduate student, Dave, my adviser, allowed me the freedom to pursue any ideas that came along, especially the ones such as this thesis, which had an uncertain direction. I didn't say it often enough then, so I'll say it now. Thanks, Dave. I'd also like to thank Juliet, who put up with my whining, goofiness, and "free food radar" as if they were the best personality traits a person could have. Your support kept me going in those rough times. Thanks, Tom Burbine, for saying things like, "*SCREW* the thesis," and for telling me how often you got published and how little you worked. Seriously, I always looked forward to coming back to Ashdown after a long day's work to hang out and hear all the new gossip. Benjie, Andy, and Eugene, thanks for allowing me into the "in crowd," even though I was just a "first-year." Thanks, also, to Tom "Big T" Stahovich, who tried, in vain I regret, to get me to be a pure researcher and stay out of industry. It was too late for me, my friend. Thanks to Ron Wicken, who never expected anything in return for his time helping me with all my hardware problems. Thanks to Darren "Double D" Dang, for asking me to help him with Matlab, which I actually consider *fun*. I thank my brother, Pat, who taught me early in my graduate career to have goals. I didn't have as many as he has, but I did achieve most of them, some a little later than others (e.g. this thesis). Finally, I'd like to thank my parents. They called on Friday nights, hoping that I would, for once, be home when they called, and then playfully chided me when I *was* home because I wasn't "out partying." But they understood my need to work and only gave me advice when I asked for it, allowing me to learn on my own, which is why, after all, I came to graduate school.

This research was supported under the Loral Systems Company contracts JS-380830-S, for "Control of Multiple Autonomous Vehicles" and JS-380888-Y, for "All Weather Semi-automated Forces." I thank Loral profusely and acknowledge their support for me and many other graduate students at the MIT AI Lab.

Table Of Contents

1	Introduction	15
1.1	Introduction.....	15
1.2	Overview.....	16
1.3	Outline.....	18
2	Control Under Time Delays	21
2.1	Introduction.....	21
2.2	Robust Stability.....	22
2.3	Feedback Control with Time delays	22
2.4	Supervisory Control.....	30
2.5	Time Delay Ranges and Types	31
3	Task-level Supervisory Control	35
3.1	Introduction.....	35
3.2	Overview of Task-Level Supervisory Control.....	35
3.3	The Goals and The Problems.....	36
3.4	Solution of Task-level Supervisory Control	37
3.5	How To Use the Task Primitive Set	42
4	Implementation of a Task-level Supervisory Controller	45
4.1	Introduction.....	45
4.2	System Description	45
4.3	The Complex Tasks	49
5	Analysis and Lessons Learned	59
5.1	Introduction.....	59
5.2	Experiments	59
5.3	Performance of the Interaction Set	60
5.4	Other Factors Affecting the Experiments	64
6	Conclusion	67
6.1	Introduction.....	67
6.2	Review	67
6.3	Contributions	70
6.4	Future Work.....	71
6.5	Conclusion	74
Appendix	Robot system hardware and control	77
A.1	Hardware.....	77
A.2	Network.....	79
A.3	Controller	79
A.4	Supervisory Stability.....	81
References	83

List Of Figures

Figure 2.1: Typical analog feedback loop without explicit time delays.	22
Figure 2.2: Analog feedback loop with explicit pure time delays.	23
Figure 2.3: Shower with time delay.	24
Figure 2.4: Block diagram of shower/bather system.	24
Figure 2.5: Simulated results of various model bather behaviors.....	26
Figure 2.6: Root locus diagrams for the shower bather system.	27
Figure 2.7: Nyquist plots of a typical 2nd order open loop stable system.....	29
Figure 2.8: Same 2nd order system as above but with a series pure time delay.	30
Figure 2.9: Hypothetical plot of effectiveness of control types.....	33
Figure 2.10: Schematic diagram of network latencies	34
Figure 3.1: Conceptual diagram of task-level control.	35
Figure 3.2: Task-level supervisory control system block diagram.	36
Figure 3.3: Schematic of the Internet.	36
Figure 3.4: Schematic of a task primitive.	37
Figure 3.5: Taxonomy of tasks for executing the interactive actions.....	38
Figure 3.6: Primitives within the interaction taxonomy.	39
Figure 3.7: Taxonomy of low-level control tasks.	40
Figure 3.8: Primitives within the low-level control taxonomy.....	41
Figure 3.9: General complex task diagram.	43
Figure 4.1: Illustration of system setup.....	45
Figure 4.2: Picture of robot's environment.....	46
Figure 4.3: Robotic system hardware and connections	46
Figure 4.4: Step responses of the robot from various initial conditions.....	47
Figure 4.5: Measurements of forward network latency	48
Figure 4.6: Measurements of feedback latency.	49
Figure 4.7: "Follow-surface" task-level diagram.....	50
Figure 4.8: "Find-stiffest-spot" task-level diagram.	50
Figure 4.9: "Map-surface" task-level diagram.	51
Figure 4.10: Sensor "query" task-level diagram.	51
Figure 4.11: "Moveto" task-level diagram.	52
Figure 4.12: Comparison of three velocity profiles.....	53
Figure 4.13: "Touch" task-level diagram.	54
Figure 4.14: 26 Cartesian direction specifiable in "touch" and "stiffness."	55
Figure 4.15: "Stiffness" task-level diagram.	56
Figure 4.16: "Follow" task-level diagram.	57
Figure 5.1: How the robot behaved for very soft (left) or irregular surfaces.....	62
Figure 6.1: The user interface for the task-level supervisory robot controller.	72
Figure 6.3: Predictive simulation and control system.	73
Figure A.1: Graphical diagram of the robot	78
Figure A.2: Low-level controller block diagram.	80
Figure A.3: Default behavior to achieve supervisory stability.	82

this page intentionally left blank

List of Tables

Table 5.1: “Find Hardest Spot” task completion times.....	60
Table 5.2: Primitive execution times and latency ratios	61
Table 5.3: Results of primitive set versatility test.....	63

Chapter 1

Introduction

1.1 Introduction

Robotic systems used in the presence of long time delays relative to their dominant dynamics can successfully be controlled using task-level supervisory control techniques. However, significant issues exist when designing these control schemes. Time delays can vary over a wide range for one robot, robots may possess varying levels of autonomy and “intelligence”, operating environments may change drastically, and performance requirements can change. User interface design and robot hardware architecture also are important when designing for supervisory controlled robots.

One of the keys to solving some of these problems is determining a general set of actions or tasks for a robot to execute under a supervisory system. This set of actions will vary depending on exactly how long the time delay is, relative to the speed of the robot itself. The set must be rich enough so that it can be used to develop more complex and useful tasks, with the addition of some logic. On the other hand, it can not be so complex that it requires extremely skilled operators to learn and perform with the robot.

Aside from doing repetitive, accurate work, one of the chief uses of robots is for remote action, or teleoperation. This use is becoming more and more prevalent because of more harsh environments to do scientific study in (Yoerger 1991), and because of the increasing cost of placing and supporting a human in these environments. Also, the use of robots to compliment a human's actions over distance is being more widely studied (Rosenberg 1993, Funda 1993). Allowing robots on a computer network to interact with each other, with the environment, and with their human operators offers a tantalizing research area and a significant increase in the use of robots in everyday life. The physical displacement of the robot from the operator in these situations, as well as latency periods in networks, often introduces significant time delays.

Important in most supervisory systems is “to achieve accuracy and reliability of the machine without sacrificing the cognitive capability and adaptability of the human.” (Sheridan 1992). We would like to retain the human's intelligence and project it to a robot. If that is not allowed, then intelligent computer algorithms must be relied upon. Even the most sophisticated programs, however, are still incapable of dealing with the variety of situations a human can (Steels 1989). The set of tasks given to a robot will be well designed if it allows the human enough freedom to sufficiently vary the actions of the robot.

The purpose of this thesis is to develop some ideas on control of dynamic systems with long time delays, to provide some results on task-level control for a specific range of time delays, to present the experimental work used to confirm these results, and to discuss some future applications of the ideas presented. The thesis will not dwell on other important aspects of supervisory control systems, such as user interfaces, human physiological and psychological factors, and robot design. Although all of these are important, there is considerable literature available on them (Sheridan 1992, see references therein).

1.2 Overview

1.2.1 Supervisory control

The subject of supervisory control was well outlined by Sheridan in 1992 (Sheridan 1992). He clarified and unified the ideas that had been presented up until then, and he detailed the most useful results. It is an excellent starting point for research into this topic. The main issues discussed in the book are briefly reviewed here.

The *form* of a supervisory system is influenced by four main aspects: the robot hardware architecture, control system, level of automation, and the scope of the work. All are heavily dependent on the application the robot is to be used for. Deep sea robots certainly have different forms from warehouse retrieval robots because of the different tasks they perform and the different environments in which they operate.

Time delay is another major issue discussed by Sheridan. Time delay's deleterious effects stem from its destabilization of feedback control loops. These effects have been studied and dealt with at the analog level to some extent (Oguztoreli 1966, Marshall 1979). Early supervisory designs required users to execute a "move-and-wait" strategy, where they would send a command to the robot, wait until they received some feedback to check if the robot had performed satisfactorily, and send another move-and-wait command based on that determination. The result was prohibitively slow, but usually successful, task completion.

Most solutions to this problem involve designing higher level tasks which can execute reliably and would take many of the intermediate steps away from the operator. This is the basis for supervisory control, and it was inspired by the need for better ways to deal with time delays. Sheridan states that the delays in control are acceptable as long as the task accomplishes a large enough portion of the overall goal, the disturbance bandwidth is low, and the supervised (subordinate) system is trustworthy.

Another effective solution is predictive displays (Noyes 1984, Park 1991, Hirzinger 1993), which attempt to simulate the response of the system and relay those results to the user. This would allow decision making based on these simulated experiments, and commands could be made with higher confidence in their success. This requires the robot's environment to be somewhat controlled, since the simulation knows only what it was programmed with and exogenous inputs may come from the environment. These systems have become more sophisticated with the introduction of faster and cheaper computer graphics, and have been shown in (Park 1991) to decrease reaction times, especially for gross robot motions.

The third main issue discussed by Sheridan is user interface design, of which predictive displays are a part. Many of the results discussed are empirical, and in general, show that displays should present only the *required* information, commands should be in a "natural language" form, and the operation should not require the operator's full and complete concentration (Sheridan 1983) if it stems from complexity of control.

One element of supervisory systems not extensively discussed by Sheridan is the design of the tasks a robot is to perform, especially in relation to time delays. Many systems which use tasks have very generalizable primitive actions (Hirzinger 1993, Watanabe 1993) whose completion times are well characterized but are not tailored for the specific time delay of the system. It is these *built-in* tasks that a robot can perform which are important, and the operators are able to use their intelligence to decide upon task execution strategies. This is part of the attractiveness of supervisory control.

The dominant dynamics of the systems *relative to the time delay* are also important. If the robot has a time constant on the order of a half a minute, a 2 second delay is insignificant; however, if the system's time constant is half a second, then a 2 second time delay is overwhelming. In this thesis, time delays will be considered in this relative fashion. A "long" time delay is long relative to the systems dynamics.

Sheridan arrived at a number of relevant and strong conclusions. First, despite the existence of many of the excellent solutions to specific supervisory control problems, the integration of these solutions into a general theory will require more time. The variety of the supervisory systems which have been designed and a lack of mathematical basis for designing them are the chief reasons for this.

Second, supervisory control is a good solution for systems with long time delays. Poor initial results using supervisory control with time delays made this conclusion not as obvious as it sounds. Considerable experimental work molded the idea into its successful current state.

Finally, Sheridan concluded that supervisory control is a good method of control for all types of automation systems, even those without significant time delay. Relieving the user of some burdensome, tedious, or repetitive work and transferring this to the quicker, more accurate robot is a goal sought since the inception of automatic systems.

1.2.2 Artificial Intelligence

Automation is at the heart of supervisory control. Plenty of definitions of "automatic" exist. The basic idea is that a machine which can operate independently of human interaction and make decisions to achieve tasks is automatic. There is a continuum of levels of automation, from a garage door opener to the automatic landing system for commercial aircraft.

Machine decision-making is a subset of artificial intelligence (AI). Part of the research deals with the completion of a task by a machine. Brock (Brock 1993) introduced *task-level control*. He showed that a task can be broken down into simpler tasks, which can all be further de-aggregated, until, at some level, the decisions to be made are intuitive and programmed simply. A programming language-like structure emerged, with the significant difference that logic was *built into* all the tasks, and little external logic was required to build the more complex behaviors. Failures could be traced to a single lower level task, which could be corrected. The tasks all had a standard architecture, which allowed proofs of convergence and stability, something which supervisory control has difficulty with. This idea yields stable, fully automatic systems for task completion, and is general enough to be applied to a wide range of systems.

Important here is the structure of the tasks, and the knowledge built into them. Even the lowest level tasks required some knowledge about their local environment. Available sensor data (context), available actuators (actions), and setpoints (goals) were built into the tasks. The gain here was the generalizability of the control system design using contexts, actions, and goals as design elements.

The definition of a *semi-structured* environment was introduced here, as well. It is a useful concept for telerobotics because guarantees some properties relating a robot's actions to their results on the environment and the effects of exogenous inputs, especially disturbances, on the environment. Tasks can be designed which make use of these properties. Stability proofs are possible because of this concept, so its power should not be overlooked. This thesis will assume semi-structured environments when presenting task-level control designs.

The work continued with Narasimhan (Narasimhan 1994), who built high performance tasks by accurately modeling the physics of the robot's environment. The models, and some thought about planning, led to tasks which were robust to uncertainties. The problem of pushing a block with static and kinetic friction to precise locations was solved using his techniques. This idea of having a good physical model available to the controller is known as model based control. It is a general extension to the Kalman filter so successful for linear feedback control systems.

Looking more deeply into planning, Maes (Maes 1989) arrived at some general results for fully autonomous systems, especially mobile robots. She found that action selection should be reactive, fast, goal oriented, and robust. Reactive and fast action selection is tantamount to having reflexes. Being goal oriented on top of this is a significant problem, solved primarily through heuristic means. Robustification, or insensitivity to uncertainties, can often be solved by more accurate models and by extensive debugging. Supervisory control, and this thesis in particular, seeks to eliminate some of this complexity in planning by allowing a human operator to perform goal-oriented planning, and by allowing reactive behavior by *both* the robot and the operator. This does not preclude the inclusion of this complexity if it is desired.

Another topic in AI is the autonomous sensing of the robot's environment for use by complex planning and control algorithms. Eberman (Eberman 1990, 95) used clever signal processing methods on sensor inputs to deduce environmental properties. It was found that with this more detailed information, old control algorithms became more reliable and new control algorithms which relied on this data could be built for better performance. Some of the tasks which a telero-bot will need are good sensing tasks.

1.3 Outline

This thesis describes one method of controlling systems with a pure time delay. This chapter presented an introduction to the topic and discussed some points relevant and essential to the thesis, as well a review of some literature and current research. Chapter two reviews feedback control with time delays and the problems with this. Some of the common methods of solution to these problems are presented. This leads to the introduction of the idea of task-level supervisory control, bringing it into the main thrust of the rest of the thesis. Chapter two closes with a presentation of various ranges of time delays and the choosing of a specific range which the thesis will be concerned with.

Chapter three goes into the details of task-level supervisory control, discussing what a task is, and how to build up a set of them which satisfies the requirements laid down in Chapter two. Issues of intelligence, knowledge bases, and automation are discussed. We concentrate mainly on the specific time delay range given in Chapter two, but will briefly go over other ranges. Finally, we present some general examples of how to use the task set for the intended purpose, and some extensions that can be made to a task set such as default behaviors and default task execution.

Chapter four is a detailed presentation of the application of the principles of Chapter three to a specific system. It will give a description of the system and why task-level supervisory control is a good method to control it. Then, the task set will be described, explaining how it was chosen, and giving specifications for each set. Descriptions of experiments will be given.

Chapter five presents the results of the experiments and a critique of the performance of the system under the control of the given task set. The performance of both the task set and the system as a whole will be reviewed. The results of the experiments are implicit in these discussions. Finally, an overview of other factors affecting the experiments will be given.

Chapter six concludes with a review of the thesis. Contributions of this thesis are presented and future work in the area are described, as well as related fields of study and applications. Finally, concluding remarks close the thesis.

Chapter 2

Control Under Time Delays

2.1 Introduction

The purpose of this chapter is to arrive at a good method to control dynamic systems with long time delays. A brief review of digital control theory and z-plane analysis will be presented, showing the deleterious effects of time delays in feedback loops, and suggesting that linear feedback type methods may not be appropriate for long time delays. That leads to the idea of supervisory control, where we introduce a novel approach, called task-level supervisory control, which relies on a set of primitive tasks, tailored for a specific time delay range, to be used in a supervisory manner. The delay ranges are quantified and classified, and the stability of the control system is discussed.

Time delays in control systems became important when processes under PID control in the 1940's (Ziegler, Nichols 1943) became more complex and latency in manufacturing processes became more dominant. Ziegler and Nichols' (Ziegler, Nichols 1942) famous PID tuning rules were often disregarded or significantly altered to compensate for delays. The reduction of gains seemed the only proper method to control these systems, at the expense of lower performance. See (Shearer 1990) for a description of PID control and gain modification to compensate for time delays.

Despite the achievement of an analytical understanding of the problems with time delays, gain reduction still seemed the only sure solution. With the advent of high-speed microprocessors, it was realized that the reason for the time delay might be eliminated if local control of a system could be built. Then, only setpoints would need to be sent over a time delay. This was when supervisory control became feasible.

Supervisory control systems became important primarily because of the space race which began in the early 1960's. The Apollo moon missions presented challenges to engineers because the time of a round trip transmission between the Earth and the Moon was about 3 seconds (Sheridan 1992). If moon-based vehicles and other systems were to be controlled properly from Earth, a hierarchy of control had to be installed, where an Earth-bound user could send commands and trust the system to obey satisfactorily.

The need for control under time delays will become more important with the increased interest in flights to Mars and beyond. Also, the meteoric rise in the use of the Internet indicates that systems which require control with time delay will become widespread. However, old challenges still exist. Time delays are getting longer, and systems are becoming more complex and more demanding in performance. A basic understanding of the issues involved will provide some necessary insights.

2.2 Robust Stability

2.2.1 Definitions of stability

Any discussion of control system design requires consideration of stability. We therefore present here some definitions of stability relevant to this thesis and discuss how they influence control system design. There will be two types of stability in this work.

Low-level stability will be asymptotic stability: all outputs exponentially progress in time to a constant value from any initial condition (see Figure 4.4 for an example). Oscillations within the decay are allowed. It will be applied to local feedback controllers within a supervisory system.

Supervisory stability will denote bounded outputs in time and no limit cycling. Limit cycling is a phenomenon in nonlinear systems in which a system is stable but not convergent. Low-level limit cycles such as gear backlash and discretization effects are removed from this definition. This type of stability is applied to the overall supervisory system.

To achieve supervisory stability a system must first possess low-level stability, which can be achieved via traditional linear control systems. Supervisory stability is achieved through the logic built into the tasks and actions to be performed by the robot. This logic, like the low-level control system, is not unique.

Robust stability, as applied to both types, means that the system is stable even under uncertain conditions, such as varying mass or blockages to robot paths. It is known that low-level robust stability can be achieved through analysis of the probable uncertainties (Doyle 1992), and we use this approach for supervisory stability.

2.3 Feedback Control with Time delays

2.3.1 Block diagram paradigm

A typical analog feedback system consists of a plant, or system to be controlled, and a controller as shown in Figure 2.1. The plant can be separated from its sensors and actuators or all can be lumped together, depending on which parts are deemed most important by the engineers.

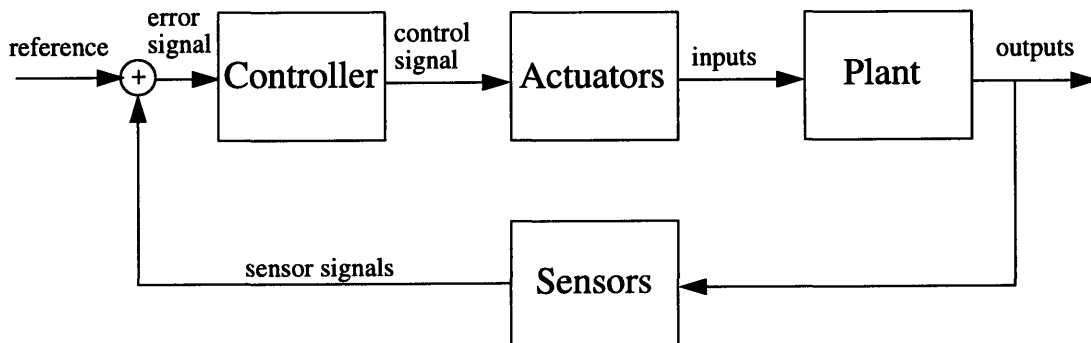


Figure 2.1: Typical analog feedback loop without explicit time delays. This is a common way of diagramming systems for the study of linear control systems.

Time delays are often present in dynamic systems but are lumped together with the system's dynamics because they are short compared to those dynamics. For this discussion the time delay

will be explicitly shown. Figure 2.2 shows common places for time delays to enter a system, such as between the controller and the actuator (actuator delays) and between the sensor and the controller (sensor delays).

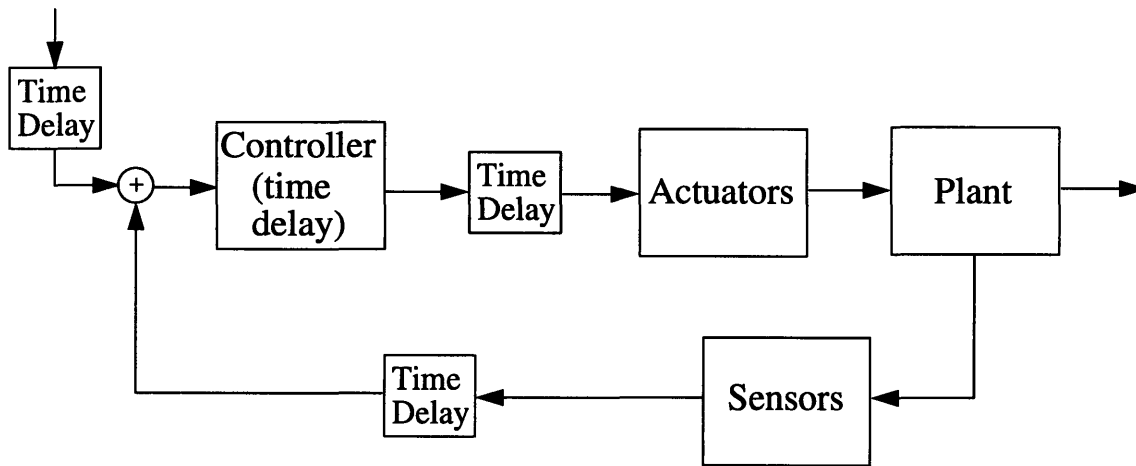


Figure 2.2: Analog feedback loop with explicit pure time delays. Common places for the time delays to enter the system are shown. Systems with time delays may or may not have multiple sources of delays.

2.3.2 Intuitive example of time delay’s effects

Suppose a bather, B, is showering where the pipes are much longer than normal (Figure 2.3), yielding a time delay of d seconds between control action and sensing of T , the water temperature at B (Figure 2.4). B’s desired water temperature, T_r , can be achieved by the proper setting of the water control handle between “hot” (130 degF) and “cold” (40 degF). The bather can be modelled as a simple proportional controller by effecting a temperature change, ΔT , after an *expected* delay of e seconds, where we assume $e \leq d$. So ΔT is given by

$$\Delta T(n) = K [T_r - T(n)] \quad (2.1)$$

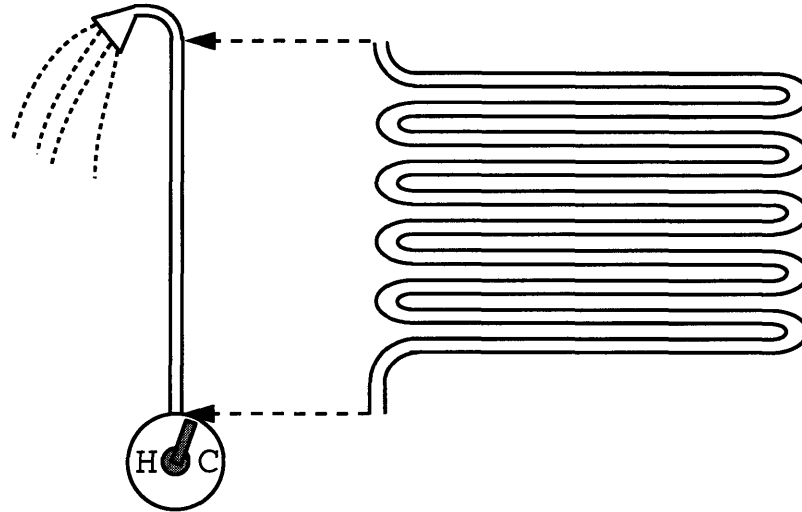


Figure 2.3: Shower with time delay. When the longer pipes are inserted in place of the short one, a bather who is used to the short pipe will have a difficult time keeping the shower's temperature stable.

and n is shorthand for time $t = (e)(n)$ seconds, so each n is an e second interval. The dynamics can be written as a difference equation

$$T\left(n + \frac{d}{e}\right) = T\left(n - 1 + \frac{d}{e}\right) + K [T_d - T(n)] \quad (2.2)$$

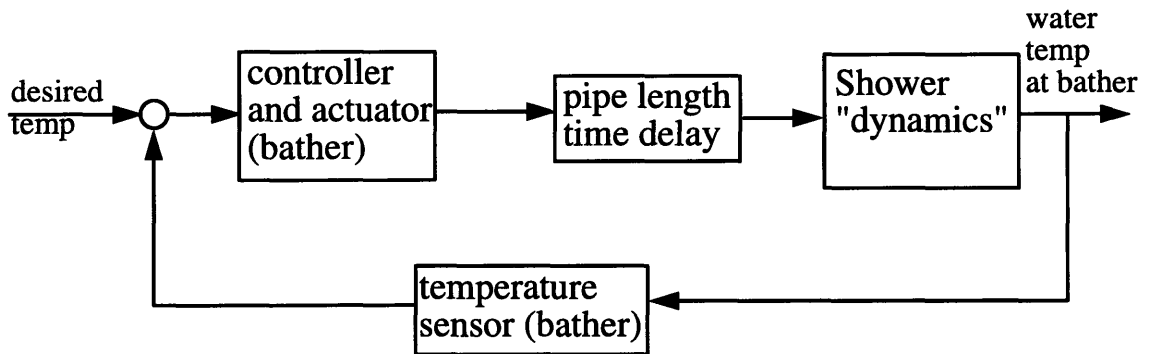


Figure 2.4: Block diagram of shower/bather system. The time delay appears here between the actuator and the plant, which is common in fluid flow systems.

To check this equation for stability it is necessary to only solve the homogenous part of (2.2), since we are assuming the external input, T_d , is constant, and stable linear systems possess bounded-input bounded-output (BIBO) stability.¹ This gives

1. A BIBO system yields bounded outputs when bounded inputs are applied. All asymptotically stable linear systems exhibit this behavior.

$$T\left(n + \frac{d}{e}\right) - T\left(n - 1 + \frac{d}{e}\right) + KT(n) = 0 \quad (2.3)$$

which is solved is solved with

$$T(n) = Az^n \quad (2.4)$$

Applying this to eq (2.3) and performing some algebra yields

$$z^{d/e} - z^{\frac{d}{e}-1} + K = 0 \quad (2.5)$$

For stability, the roots of (2.5) must lie within the unit circle on the complex plane. Simulations of eq (2.2) are shown in Figure 2.5 for various values of K and e, with a constant pipe delay, d, of 10 seconds. T_d will be 100°F, and the initial temperature will be 50°F.

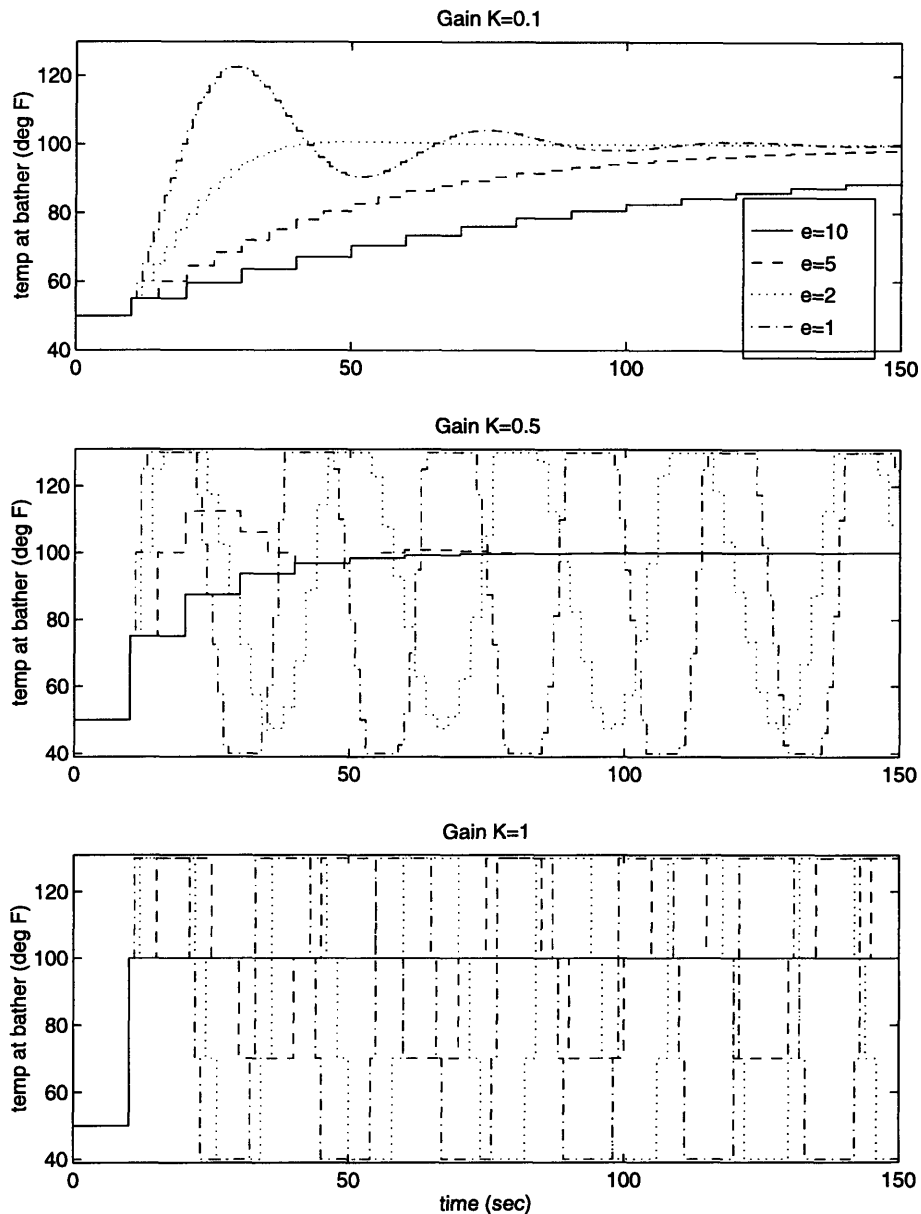


Figure 2.5: Simulated results of various model bather behaviors. Instability results very easily. Although a bather would probably dampen his or her actions, automatic systems are frequently not as intelligent.

Root locus diagrams for this system are plotted in Figure 2.6 as the gain, K , varies, for the values of e in Figure 2.5, along with the actual roots of eq (2.5) for these parameters. All points inside the unit circle yield a stable system. It would help the reader understand this example by applying it to oneself, and guess one's reaction if the shower pipes were suddenly lengthened by a factor of ten or more.

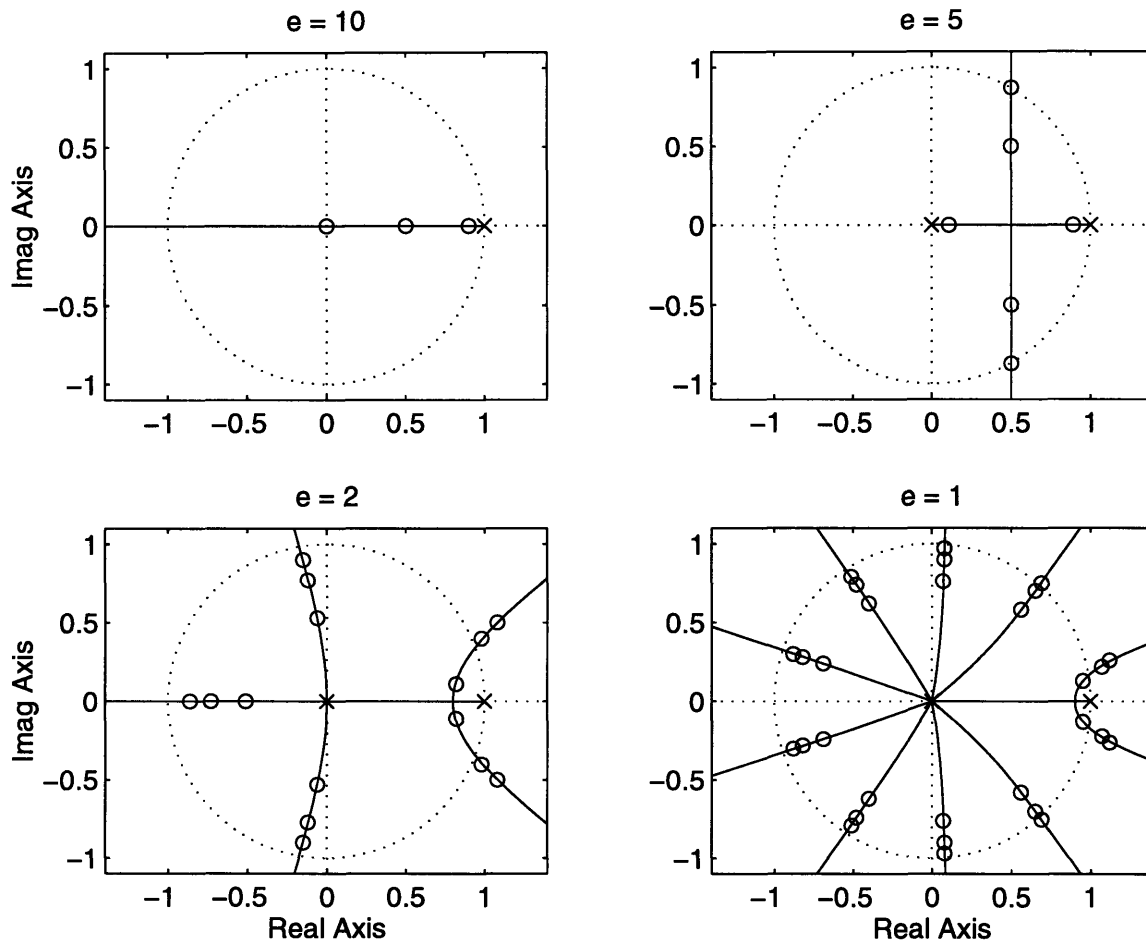


Figure 2.6: Root locus diagrams for the shower bather system. The circles denote the roots of eq (2.5) for the gains used in Figure 2.5. As the gains increase, the roots move away from the x's along the root locus paths. When the paths cross the unit circle the system is unstable. Note how early the circles leave the unit circle for $e=1$.

2.3.3 Principles of time delays in discrete sampled control systems

For discrete systems like the bather example, events such as actuation and sensing happen at periodic intervals, and no new control events occur in between these intervals. This is a common situation in supervisory control systems, where new information or actuation change only after some latency period, often due to transmission delays. Thus, supervisory systems with time delays can frequently be thought of as discrete sampled systems.

As seen in the example, *slower systems require slower control* to achieve adequate responses. With discrete systems this idea becomes more important, as the latency time is embedded within the dynamics. Z-plane analysis (Franklin, et al 1990), shows how a difference equation can be analyzed by examining its characteristic equation, like the one of eq (2.6), and how to achieve stable systems by moving the roots inside the complex unit circle. The root locus technique used in Figure 2.6 is a useful tool for this purpose. Note any parameter, not just gain, can be varied to get a root locus. Figure 2.6 shows how increasing gain always results in instability.

In discrete systems, better stability is achieved when the expected time delay is nearer to the actual time delay. This is analogous to the “move and wait” strategy often employed by supervisory systems. Although it is slower, it is much more stable than the more aggressive, “move five times and wait” strategy (i.e. $e = 2$, $d = 10$ in the shower example, see Figure 2.5). It will be shown that, although the move and wait strategy will still exist in the supervisory control system, it will be moved to a higher level and can be automated so that certain types of time delays can be minimized, especially network latencies.

2.3.4 Principles of time delays in continuous systems

Control engineers have long known the destabilizing effects of pure time delays on continuous-time feedback control systems (Franklin, et al 1994). Although not as relevant as the discrete systems to this thesis, continuous systems still provide some insights into the behavior of systems with time delays.

Characteristic equations of continuous systems also determine stability, and another way of analyzing characteristic equations is with a Nyquist plot, which provides, on the same plot, both gain and phase (time lag) information. A general result from Nyquist (Doyle, 1992) was that the stability of a closed loop system under negative feedback could be determined from the dynamics of the open loop system (including zeros). Simply put, if the Nyquist plot of the open loop system encircled a critical point a number of times depending on the number and placement of poles (roots of the characteristic equation) and zeros, the system would be unstable under unity feedback. Figure 2.7 shows a typical Nyquist plot for an open-loop stable second order system of the form

$$T(s) = \frac{1}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.6)$$

Second order systems are of great importance because many supervisory systems control a device whose dynamics have been modified, via local control, to behave as a second order system.

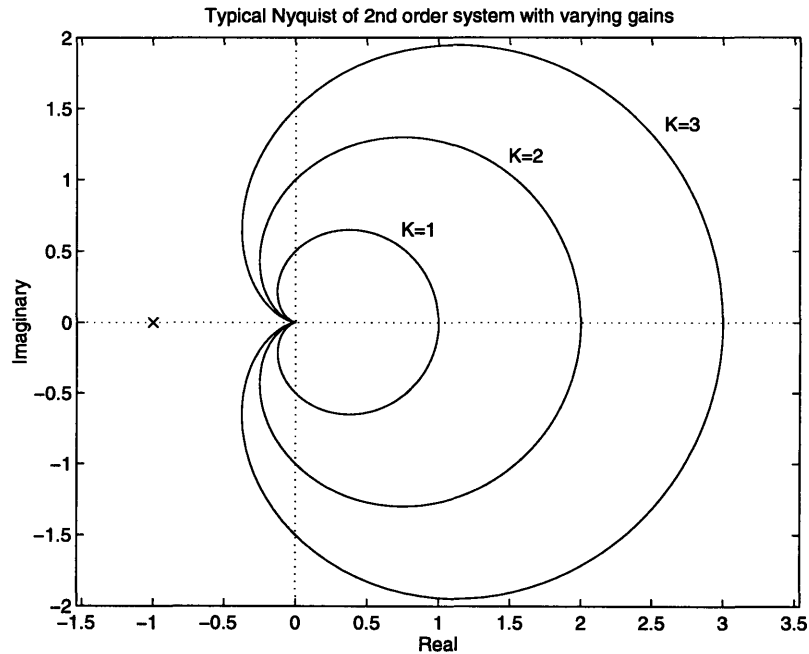


Figure 2.7: Nyquist plots of a typical 2nd order open loop stable system. $\zeta = \omega_n = 1$. Increasing the gain, K , does not affect the stability, since the critical point (marked by the x) will never be encircled.

Time delays can be represented in the Laplace domain in terms of gain and phase as

$$d(s) = e^{-T_d s} \quad (2.7)$$

Figure 2.8 shows the behavior of the same system as Figure 2.7 but with a pure series (multiplicative) time delay according to eq (2.7). The spiraling effect is disastrous for stability since increasing the open-loop gain always increases the radius of the spiral which, when high enough, will cause encirclements of the critical point. Again, we see the problems with high gain in time delayed systems.

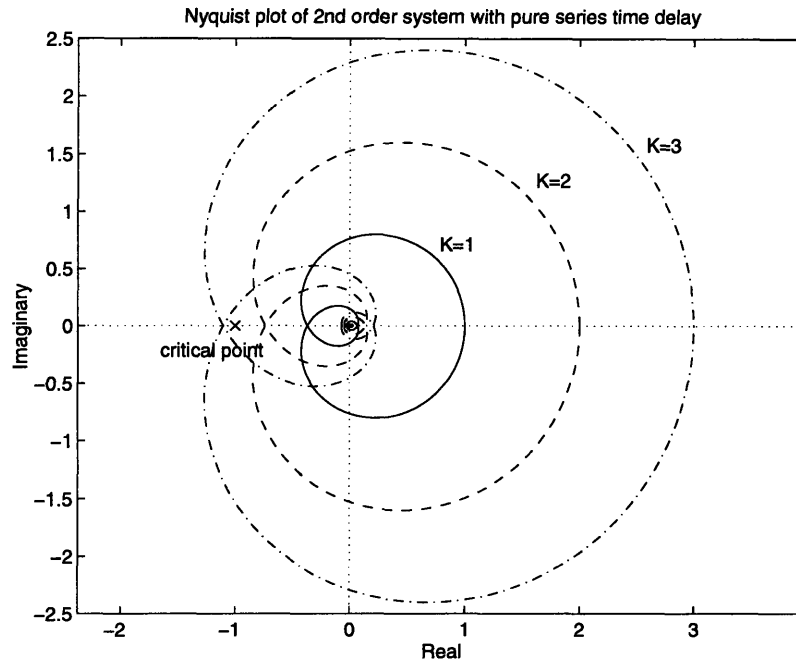


Figure 2.8: Same 2nd order system as above but with a series pure time delay. The spiraling effect is introduced, which, when the gain is increased enough, will cause the curve to encircle the critical point as shown by the (dotted) plot for $K=3$. This implies closed loop unity feedback instability.

Various methods have been used to compensate for time delays in an analog or linear systems fashion. (Oguztoreli 1966, Marshall 1979) are some good starting points for those interested. The solutions are valid under some strict assumptions.

The relevant value of most continuous systems is its time constant, τ . A general rule of thumb is that for second order systems, stability can be maintained by a typical feedback system if the time delay is approximately five times shorter than the dominant time constant of the system.

$$5T_d < \tau \quad (2.8)$$

In second order systems the dominant (slowest) time constant is most important. This determines the speed of response of the system. If analysis of a system shows this rule will frequently be violated, then a supervisory or automated control system should be used. High performance, which typically comes from high gains, is difficult to achieve from linear systems with long pure time delays, as shown in the last two sections. Supervisory control is needed.

2.4 Supervisory Control

Supervisory control is a heuristic subject, so much of the discussion in this thesis will be to convince the reader of the effectiveness of the algorithms presented. There are two criteria to consider when designing a supervisory control system. First, the system must be robustly stable. Second, it must meet performance requirements.

For supervisory systems, stability is not a well-defined concept. Here we give a condition for supervisory stability.

Sufficient condition for supervisory stability: Once a supervised system has returned a value, and no other requests are given, nor are there any disturbances, a query of that value at any instant will produce the same value as was returned initially.

This can be justified by noting that if the condition is true, then the system remains at the state which was present after the value was returned; the state is bounded and not limit-cycling. This means that after each task a supervised system performs, the system should come to rest or to an unchanging steady state, and should be able to return a statement of its completeness to the requesting agent. This statement often will be a description of the final state of the system after completing the task. A return value from a task is necessary but not sufficient for stability.

(Yoerger, 1991) describes an unmanned underwater vehicle which maintains a desired course in the face of changing currents. A return value might be the course error, which would go to a steady state and then, under disturbance-free action, would remain there until a new course was specified. This system has supervisory stability.

To meet performance requirements, the supervisory control system is designed so that it can make use of the system's hardware to complete the required tasks, assuming this hardware is capable of meeting the performance requirements. It is the operator's responsibility to use the control system properly to meet the requirements, which could include success rates, task completion speed, and disturbance rejection. During the design procedure, comparisons of measured and required performance determine if either the requirements are too strict or the control system can be improved. *Robust* performance implies that the requirements are met even under uncertain conditions such as variable time delays or changing environmental conditions or robot parameters. In many cases, increased operator experience implies increased performance robustness.

2.5 Time Delay Ranges and Types

The range of time delays acceptable for a given system is determined from its dynamics; slower systems can tolerate longer time delays. The ranges can be roughly broken down into three main categories: short, midrange and long time delays.

2.5.1 Short time delays

Short time delays satisfy the requirement of eq (2.8). This delay can be added to the time constant of the system, and the resulting dynamics analyzed accordingly. Low level robust stability and performance can be achieved under these conditions through typical feedback control, and setpoints can be sent to this controller from any source at any desired rate. The shorter the time delay the faster the system's response will be.

2.5.2 Midrange time delays

Midrange time delays satisfy

$$\frac{\tau}{5} < T_d < 10\tau \quad (2.9)$$

When the delay is approximately 10 times the time constant, digital control theory starts to tell us that traditional feedback controllers will often be unstable and can not meet even the weakest performance requirements. It is a risky endeavor to attempt stabilization of these systems, since even a small variation in time delay or gain can cause instability, as seen in Figure 2.8.

These types of systems are often studied on a case by case basis. More often than not, a supervisory scheme is employed, mostly to guarantee stability, since disturbances and parametric uncertainties are too large to design a robust feedback control system.

2.5.3 Long time delays

Long time delays are those given by

$$T_d > 10\tau \quad (2.10)$$

These delays require some sort of supervisory or task-level control to achieve satisfactory performance from the system. As the time delay increases, the control system needs more automation because the robot must perform actions longer without user input; it must make more of its own decisions while waiting for this input. If the robot finishes a task with a significant portion of the time delay remaining, unacceptable task completion rates result. Figure 2.9 presents a conceptual, hypothetical plot of what types of control may be best suited for a particular range of time delays. This plot is not the result of data, but merely a conceptual tool for illustrating the idea of this section. The term “effectiveness” is an equal measure of successful task completion, waiting period between each task, versatility of the control system, and the controller’s level of complexity.

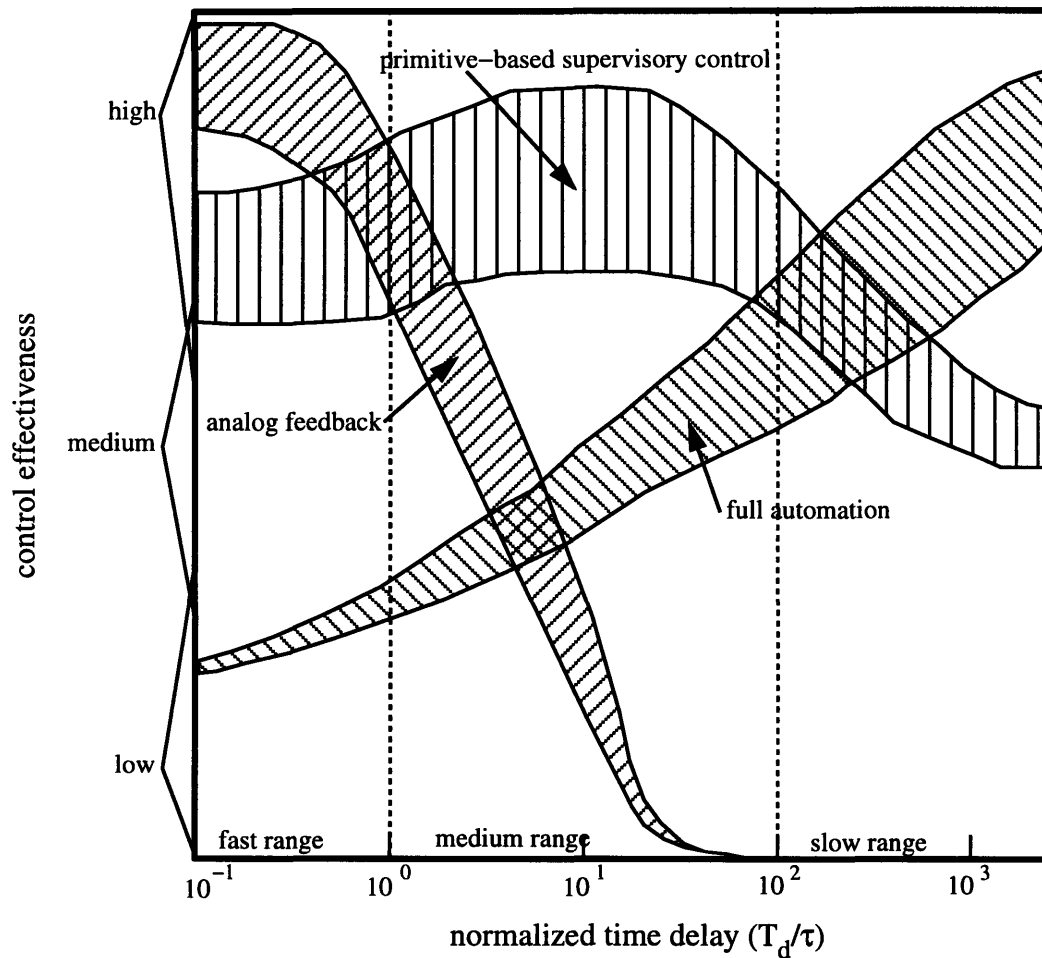


Figure 2.9: Hypothetical plot of effectiveness of control types over varying time delays. Note that pure feedback control is the only type of control that is completely ineffective over some range.

2.5.4 Types of time delays

Communication, computational, or mass transport delays are the most common types. Communications delays are the most prevalent in telerobotics and networks. *Latency delays*, frequently found on the Internet, can also appear in telerobotic systems, where a serving computer can not respond quickly. It is a waiting period before the request for robot action is processed. *Most of the time delay in network latencies occurs in the forward path.* Feedback of results is faster and decisions on the next required task can be made more quickly. With a pure transmission delay, feedback usually requires the same amount of time as forward requests. See Figure 2.10 for a diagram. This becomes important when using primitive task sets because required tasks differ depending on when they are executed in relation to the previous task. The experimental work in this thesis deals heavily with latencies.

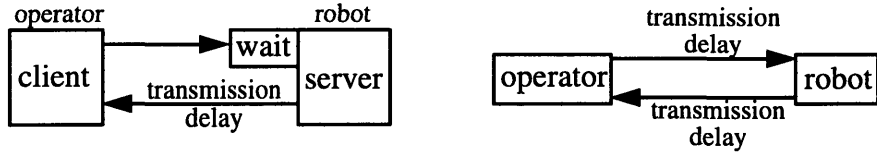


Figure 2.10: Schematic diagram of network latencies versus normal communications delays. The bulk of the time delay for network latencies (left) occurs in the forward, or requesting, path, whereas delays for typical communications (right) is spread evenly between forward and feedback paths.

Network delays can also vary significantly, especially those on the Internet, where network loads depend on user traffic and available computational resources.

Chapter 3

Task-level Supervisory Control

3.1 Introduction

The purpose of this chapter is to explain task-level supervisory control and to apply it to a real system. Section 3.2 gives a brief overview of task-level supervisory control as applied to our system. Section 3.3 states the control problems in the system and outlines the our desired goals. Section 3.4 presents the solution we used, task-level supervisory control, introducing the idea of task primitives, and designing a set of these for application to the system. Here we also discuss the sufficiency of the solution and how primitives are used. Finally, section 3.5 presents a brief general example of how the complex tasks are built and structured.

The work described here is an attempt to combine the long standing application history of supervisory control with the advanced theory of task-level control to achieve a well-designed control system.

3.2 Overview of Task-Level Supervisory Control

Task-level control theory decomposes complex objectives into simpler tasks until the they are easily implemented. Specific and limited knowledge of the environment is given to each task, which acts upon sensor input with actuator output based on this knowledge and the rules given to the task. They can be made fault tolerant and disturbance rejecting. This has been shown to be an effective control method for complex task automation (Brock 1993). See Figure 3.1 for a conceptual diagram of task-level control.

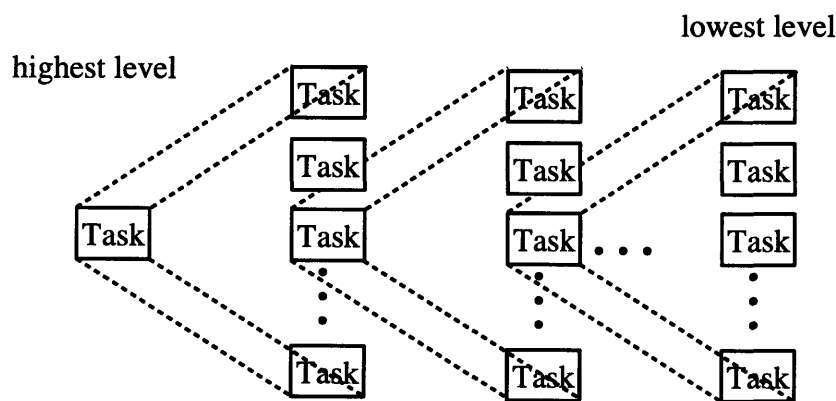


Figure 3.1: Conceptual diagram of task-level control. A task is decomposed into simpler tasks, which are further decomposed until each task is easily implemented.

This type of control system works well in structured and so-called semi-structured environments, where parts of the environment have guaranteed properties (Narasimhan 1994). However, in completely unstructured environments, this knowledge is ambiguous, so the reliability of the tasks built around it is suspect.

Task-level control was first used to make a fully automatic system, and although successful, the environment was highly structured. Supervision and intervention by a human would provide the advantages of on-line fault correction and debugging, and would relax the amount of structure needed in the environment, since a human supervisor could anticipate and account for many unexpected situations. For the case we are considering in this thesis, these advantages outweigh the disadvantages of possibly slow supervisors, complex controller-supervisor interaction and the actual physical need of a supervisor.

Figure 3.2 shows a diagram of a typical task-level supervisory control system where a robot is being controlled. This is nearly identical to the feedback control block diagram of Figure 2.1, with the inclusion of the controlling agent, which could be a human or a computer program, or some combination of both. This agent makes the system supervisory. The controller of Figure 2.1 is replaced by a task-level controller, which makes this system a task-level control system.

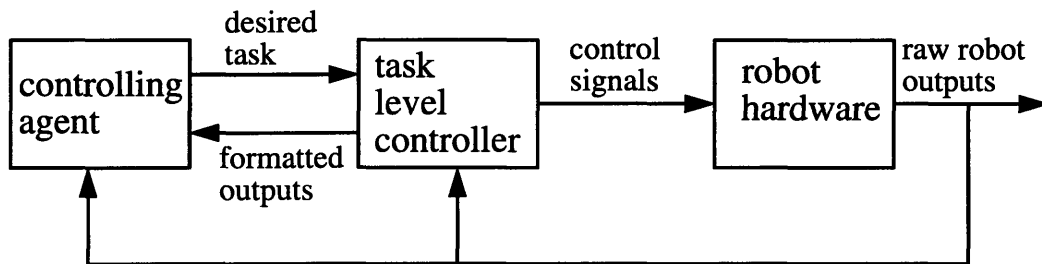


Figure 3.2: Task-level supervisory control system block diagram. The difference between this and a typical feedback control block diagram is the inclusion of a controlling agent and the replacement of the controller with a *task-level* controller. Note that the feedback is given to the agent in both processed and raw form.

3.3 The Goals and The Problems

The main goal of this research was to control an articulated robot over the standard Internet computer network. See Figure 3.3 for a schematic drawing of the Internet. The clients request an action from a server, which executes the action and returns a resulting response of data. Our robot would be on one of the servers, as shown, and the operator on a client.

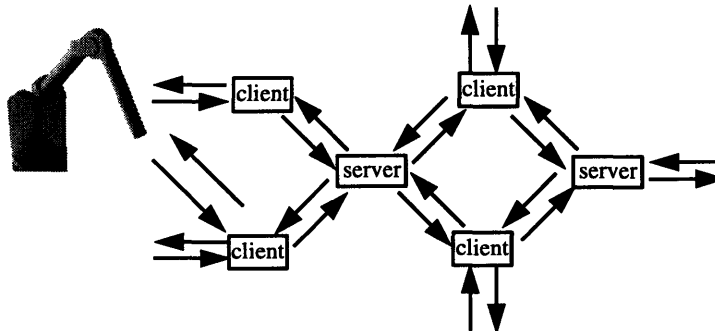


Figure 3.3: Schematic of the Internet. Server computers execute actions requested from client computers and return resulting data. Our robot would be on a server, and an operator on a client.

Most of the tools available for accomplishing tasks over the Internet deal with data, and little with physical systems. The primary problem is the long time delays due to server *latencies* and transmission times. As shown in Chapter 2, time delays are disastrous for the kind of real-time control needed for a robot. We must find a solution which can control a robot under long time delays and allow a rich set of tasks to be performed.

We want the robot to interact with the environment. Therefore, the controller should allow (1) arbitrary motions within the robot's workspace and, and it should allow (2) execution and recording of contact events.

3.4 Solution of Task-level Supervisory Control

To achieve the set of actions described above over the a network with long time delays, we used a task-level supervisory control system. We built a set of task primitives (or simply primitives) which, when combined with the appropriate logic, allow the motion and contact actions over these long time delays.

3.4.1 Basics of a primitive

A task primitive, shown schematically in Figure 3.4, takes an input, executes an action based on the input, and returns an output indicative of the action's results.

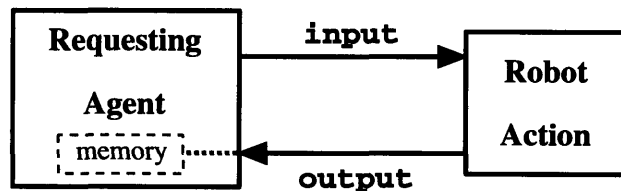


Figure 3.4: Schematic of a task primitive. There is a strict sequential architecture to the primitive. Also, outputs can easily be stored for later use. The robot action is an entity unto itself which may request other primitives.

We chose as input a list of floating point numbers. They were interpreted symbolically (i.e. symbolic values were represented as numbers), or literally. The size of the input list varied according to the bandwidth available to pass data to the primitive and the number of inputs needed. The output is also a list symbolic or literal of floating point numbers, which represented the results of the action completed. The size of the list was just large enough to return the necessary data, regardless of bandwidth. *The action respected the time delay* present in the system, which was assumed known to within ten percent. Completion times much longer than a time delay period, T_d , indicate the action was probably too complex and could have been decomposed into simpler, faster actions. Completion times much less than T_d indicated the action could have been combined with other primitives.

3.4.2 Success criterion for a primitive

During our design of primitives we determined their success using the criterion of deterministic completion times. Sever executions of a primitive given the same input and starting from the same initial conditions should return the same output in the same amount of time. The environment is also part of the initial conditions. There may exits environmental disturbances which

change the completion time and output of the primitive. We called a primitive successful if there was a deterministic between a given disturbance and the primitive's output and completion time. A primitive was not successful if its completion times or output changed under the same conditions. This can be called repeatability.

There are other criteria, such as positioning repeatability and accuracy (Wolovich 1987), which deal with the *performance* of the primitives. However, poorly repeatable motions do not imply the primitive which executes the motions is unsuccessful.

3.4.3 Common errors in primitive execution

Regardless of the work debugging a primitive, it is generally accepted that no real-time control program can be perfect, since it always faces unpredictable situations during application (Auslander 1990). Some common errors are given below.

1. **Nonsensical inputs.** Each primitive should check the inputs and return an error if the inputs are not within the allowable set. For example, inputs may require the robot to move beyond its working volume.

2. **Limit cycling.** Oscillations about the goal state are often caused by tolerances which are too strict for the robot to meet with the available sensors and actuators. The phenomenon of "hunting" due to discretization effects is an example.

3. **Network loss.** A failed network may be indicated by a much longer time delay compared to recent ones. A primitive should not depend on the network for this reason. It should complete the primitive regardless of network loss.

3.4.4 Generalized interaction task taxonomy

The set of primitives needed to achieve the two general actions given in section 3.3 can be generalized into three categories: perception of the robot and its environment, maintenance of spatial relationships between the robot and the environment, and maintenance of force relationships between the robot and objects in its environment. Figure 3.5 shows this taxonomy.

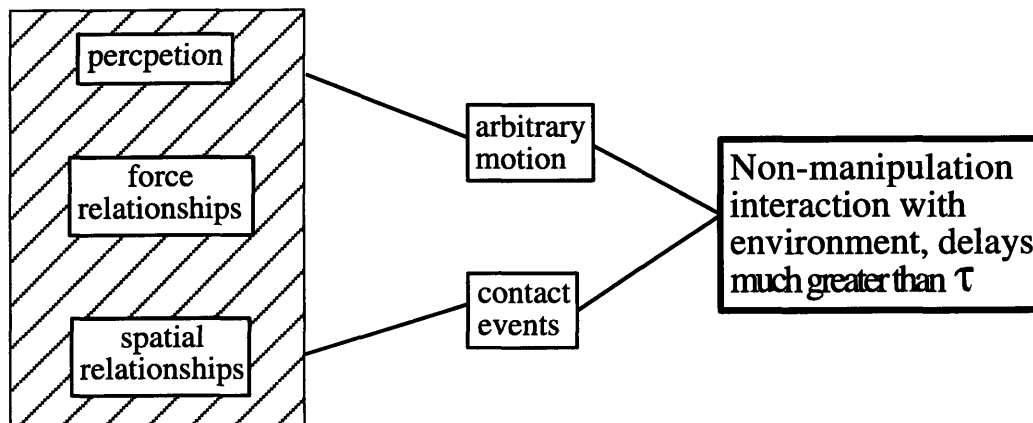


Figure 3.5: Taxonomy of tasks for executing the interactive actions of arbitrary motion and execution of contact events. These can be built into the general set of environmentally interactive actions, under time delays much greater than the time constant (τ) of the mechanism.

Perception allows the acquisition of environmental information and places the robot in a context. This **context** is basically a set of variables that defines the robot's state *within the environment*, and helps the robot decide its next action. The maintenance of spatial relationships allows the robot to place itself in a specific context, and to prepare itself for subsequent actions. Maintenance of force relationships allows direct interaction (contact) with the environment, and may also affect the robot's context. Thus, this taxonomy is complete *for contact based interaction tasks* with time delays much greater than the time constant of the robot, because it allows the robot to sense, react to, and affect its context via contact interactions, without requiring any rates of action.

3.4.5 Specific primitive set for the interaction taxonomy

We arrived at five specific primitives which can be used to build up the interaction taxonomy described in the previous section. Figure 3.6 shows how the primitives fit within the taxonomy.

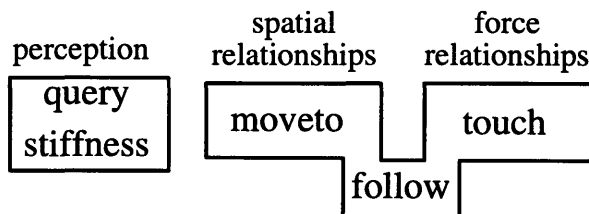


Figure 3.6: Primitives within the interaction taxonomy.

Perception contains the sensor **query** and **stiffness** primitives. For query, inputs are symbolic, indicating which internal sensor is to be checked and how the data is to be processed. The action queries the sensors and processes the data, and the output is the result.

The **stiffness** primitive takes desired Cartesian direction as input. The action moves the robot endpoint in that direction, and measures the stiffness of the first contacted surface. The output is this stiffness and the contact point.

Spatial relationships contain the **moveto** primitive. Inputs are a desired position and a symbol representing the type of move to be made. The action is the robot moving at some rate (which could also be in the input) to the specified point. The output is the final position of the robot in terms of the input. The type of motion could be along a straight spatial line, or along a straight line in joint space, or in an arc, etc. The desired position can be specified in angles or Cartesian position, and the output will return either angles or Cartesian position, respectively.

Force relationships consist of the primitive **touch**. The input is a desired direction in Cartesian space, the action is the movement of the robot endpoint in that direction until a surface is touched. The output is the position of the contact point.

The primitive **follow** could be classified as either a spatial relationship or a force relationship, depending on how it is applied. Its inputs are a desired direction and a desired force, perpendicular to that direction. The action moves the robot in that direction while attempting to apply a constant force in the desired direction of motion. The output is currently undefined, but could be extreme positions along the direction. For simplicity we restricted motions to the x-y plane in the robot's coordinate frame and the force in the negative z (down) direction, and the output was

made to be the greatest and least z values during the motion. The motion stopped after a time-out period or when the robot reached its kinematic limits.

3.4.6 Generalized low-level control task taxonomy

The system also requires low-level control and sensor actions for robot position servoing. The control computer's interface to the robot hardware is specialized electronic hardware which accepts requests and returns data. The delay for a request is comparable to the time required for the hardware to complete the requests. Thus, we also use a task-level control system for this purpose. The set of actions we wish to achieve provides all low-level PID feedback control functions. The tasks can again be grouped into categories, which are analogs of signals found on a feedback control block diagram (Figure 2.1). They are sensing, reference, and control. See Figure 3.7 for a taxonomy diagram.

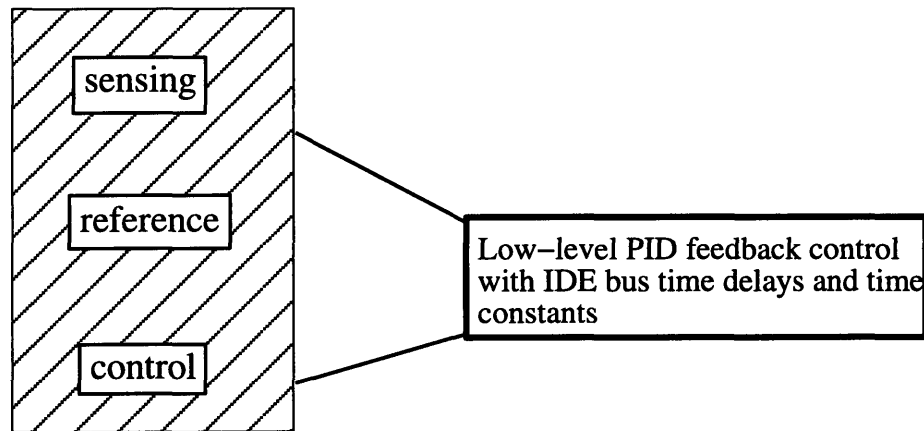


Figure 3.7: Taxonomy of low-level control tasks. These tasks make up the set of needed for low-level control of a 3 DOF robot with IDE bus time delays.

Sensing allows measurements of the robot's state, which is required for determination of the control action. In our case, position measurements are the goal. Reference allows programmable inputs (setpoint) for direct control of the robot. The setpoint is compared to the measured state, and the error between them dictates the aggressiveness of the control action. Control allows the robot's state to be influenced so that it can force this error toward zero. This taxonomy is complete for low-level PID control action because it allows input, measurement and control signals to be generated, which is all that is required of a PID controller.

3.4.7 Specific primitive set for the low-level control taxonomy

We built seven low-level control primitives useful in a variety of situations for a robot under PID control with only encoders for position sensing and IDE bus time delays. Figure 3.8 shows which categories of the taxonomy each primitive fits into.

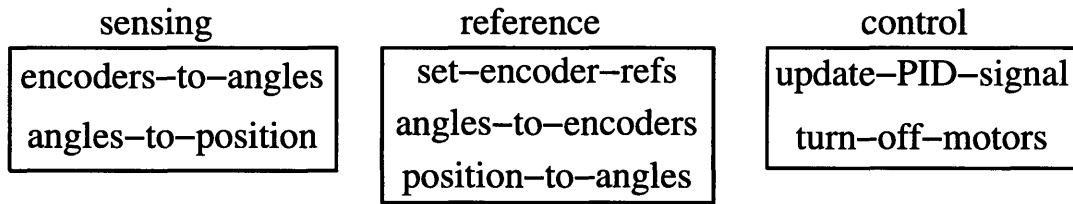


Figure 3.8: Primitives within the low-level control taxonomy

Sensing is made up of the following primitives:

1. **Encoder-to-angles** (encoder query) - Input was irrelevant, action measured the robot encoders and converted the value to robot joint angles, and output was these joint angles (any units).
2. **Angles-to-position** (forward kinematics) - Input was a full set of angles, the action calculated the robot's endpoint when the robot's angles were configured this way, and the output was this position in Cartesian coordinates in the robot's global reference frame (in any units desired).

Reference includes the following primitives:

1. **Set-encoder-refs** - input was a list of encoder values, the action set these values to the current controller setpoint, and the output was irrelevant
2. **Angles-to-encoders** - Input was a full set of joint angles, the action converted these angles into corresponding encoder values, and the output was a list of these encoder values.
3. **Position-to-angles** (inverse kinematics) - Input was a Cartesian (x, y, z) position, the action computed a set of robot angles which will yield this position¹, and the output was this set of angles.

Control consists of the following primitives

1. **Update-PID-signal** - Input was irrelevant, the action computed, based on current position and setpoint, the control signal using a PID algorithm (see appendix) and sent this signal to the appropriate hardware, and the output was also irrelevant. The action also updated certain control states, such as the integrator sum.
2. **Turn-off-motors** - Input and output were irrelevant, and the action forced all motor currents to zero, thereby effectively turning off all the robot's motors.

3.4.8 Logic with which to build complex tasks

The existence of a set of primitives, like those defined above, is not sufficient to build automated complex tasks. Some form of logic or "glue" to build these task is required. *Finite state machine* logic is used. This has *memory* and *state (or variable) assignment* functions to store data calculate with it, and it allows *if-then-else logic* and *looping* constructs. Memory is necessary because primitive execution depends on previous results. If-then-else logic is necessary to test remembered states (or variables) to determine the next primitive. Looping constructs are necessary when continuous operation is required. Loops allow the task to return to a previous state.

1. This set is often not unique in robotics. Care must be taken to ensure the desired set is calculated.

3.4.9 Sufficiency of the “language”

How do we know that the primitive set and the logic described are sufficient to achieve the desired goals? Meeting performance requirements is a good indicator of sufficiency, but requires possibly costly testing. We can also check if each category in a taxonomy contains the primitives required to fulfill the category’s purpose. Once this is established, the logic should allow the execution of any primitive, in any order, for any length of time. Note this is not a strict rule but a general guideline for determining if the language available to a task designer is sufficient to design a rich set of tasks.

The interaction primitive set is sufficient for, point-to-point, contact based tasks over a network with large time delays. It allows *perception* of stiffness at discrete contact points, *spatial motion* to discrete points, and both discrete and continuous *force interaction* events. The logic is sufficient because it allows a continuous arbitrary execution order of these primitives. Together, these give a sufficient language with which to build a set of complex tasks to perform the environmental interaction we desired.

The low-level control primitive set is sufficient because it allows the generation of all the signals seen for PID feedback control: reference, control, and measurement. Many control block diagrams also include noise, which our set does not account for, and hence the primitive set is *not* sufficient to handle noise in low-level control. A “filter” category would include control and noise attenuation primitives.

3.5 How To Use the Task Primitive Set

3.5.1 User level versus automatic interaction

The primitive task set can be thought of as a high-level robot programming language specific to the robot being used. Users can execute the primitives one at a time or use their own intelligence rather than the available logic to decide upon an execution strategy and inputs. This is full supervisory control, and is robust to disturbances and unexpected situations. However, humans have their own time constants and can not execute primitives which require millisecond or less reaction times, (such as low-level feedback control), or actions which take several hours of concentration, so automation must be employed, which makes use of the finite state logic. Significant speedups result because the computer can often calculate faster and the entire logical sequence or program can be given to the computer over the time-delayed path, thus eliminating the delay after one time across the network.

3.5.2 Construction of automated complex tasks

An abstract schematic of a complex task built from primitives is shown in Figure 3.7. The boxes are the execution of primitives, the ovals are assignment statements, branches in a path are logical decisions (labelled with its logical condition), and paths which return to a previously executed function (primitive or assignment) are loops, shown as dash-dot lines. The => symbol (pointing into or out of the object as appropriate) indicates an input to our output from a primitive or assignment, and can come from or go to anywhere since memory is allowed.

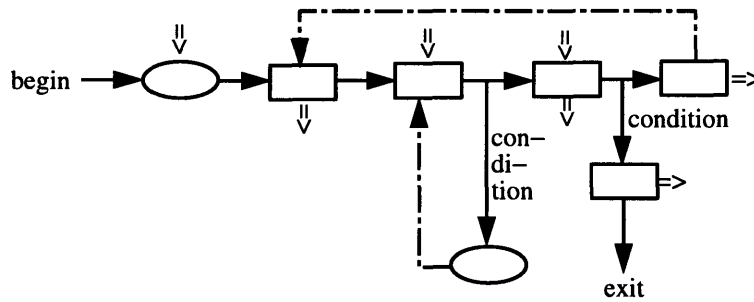


Figure 3.9: General complex task diagram. Each primitive is constructed from a lower-level set of primitives. Inputs to the primitives (designated by =>) can come from any source since the logic allows memory and assignment, and outputs(=> pointing out) can go to any variable.

Several complex tasks can be built and executed as primitives from a higher level control system. This is the basis and attractiveness of task-level control. At many levels the tasks may be suitable for human operators to execute, providing supervisory control, yielding a task-level supervisory control system which can be built for an individual system. In our case, this range is the set of 3 degree-of-freedom articulated robots with *only encoders* for sensors and controlled over Internet length time delays. This is the design we set out to achieve. The low-level control primitive set, however, covers the range of all encoder based PID feedback controlled robots with IDE bus level time delays.

Chapter 4

Implementation of a Task-level Supervisory Controller

4.1 Introduction

The purpose of this chapter is to present the implementation of a task-level supervisory controller based on the design principles discussed in Chapter 3. We start by briefly describing the physical system to be controlled and characterizing it in terms of its dynamics and time delays. We then describe the details of the primitive sets outlined in Chapter 3, showing the task-level construction of each. This allows the generality of task-level control to be shown. We also describe some of the practical issues involved with building the tasks, such as how to make tasks which allow a human user to efficiently execute complex robot actions.

4.2 System Description

The system we are controlling is shown in Figures 4.1, 4.2, and 4.3 as an illustration, photograph, and diagram, respectively. The robot is connected to a computer which performs control and is in turn connected to another computer which accepts requests over the network. See the appendix for a detailed description of the robot's hardware and control system, the computers used, and how the network was hooked into the system. (Note that two computers are not necessary, but because one of the computers was running under a non-multitasking operating system, the use of two computers allowed greater programming freedom and removed some concern about network interfacing.)

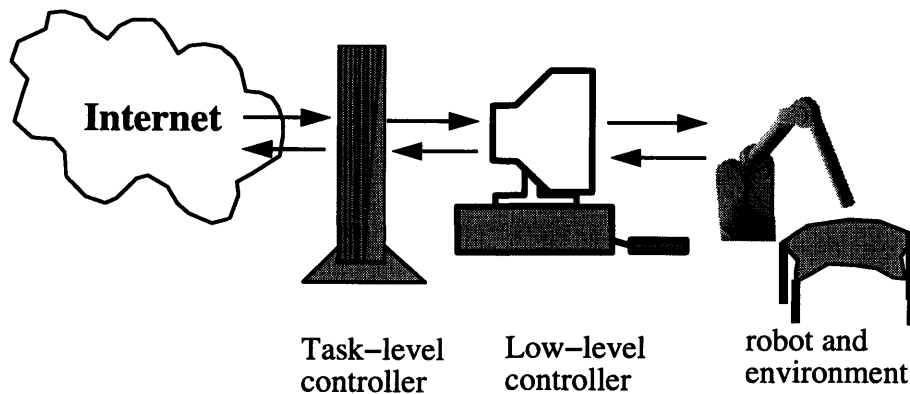


Figure 4.1: Illustration of system setup. The robot is controlled directly by a control computer, and the control computer is controlled directly by a task-level control computer which also accepts network requests.

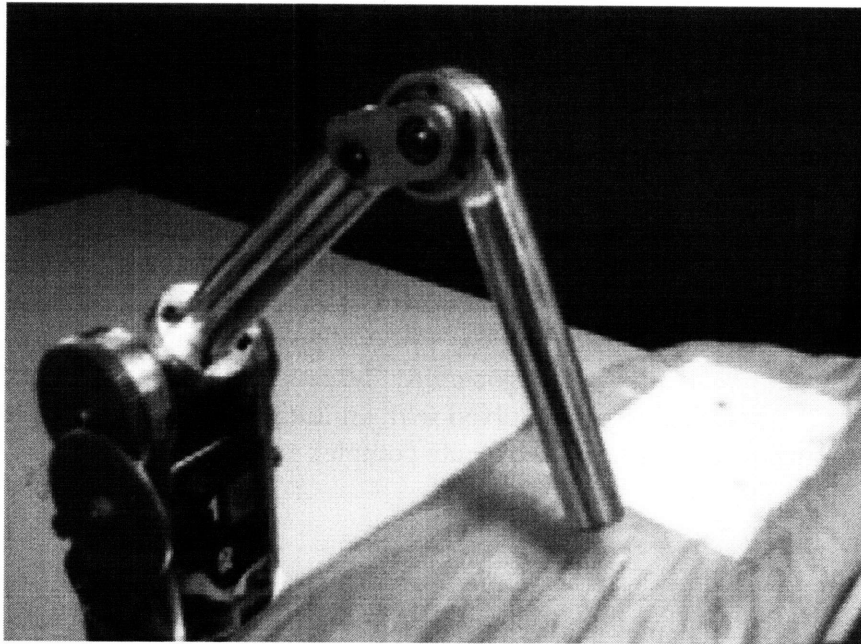


Figure 4.2: Picture of robot's environment with the robot positioned in a typical configuration. The surface is compliant and has local stiffness anomalies, which is meant to simulated a human chest.

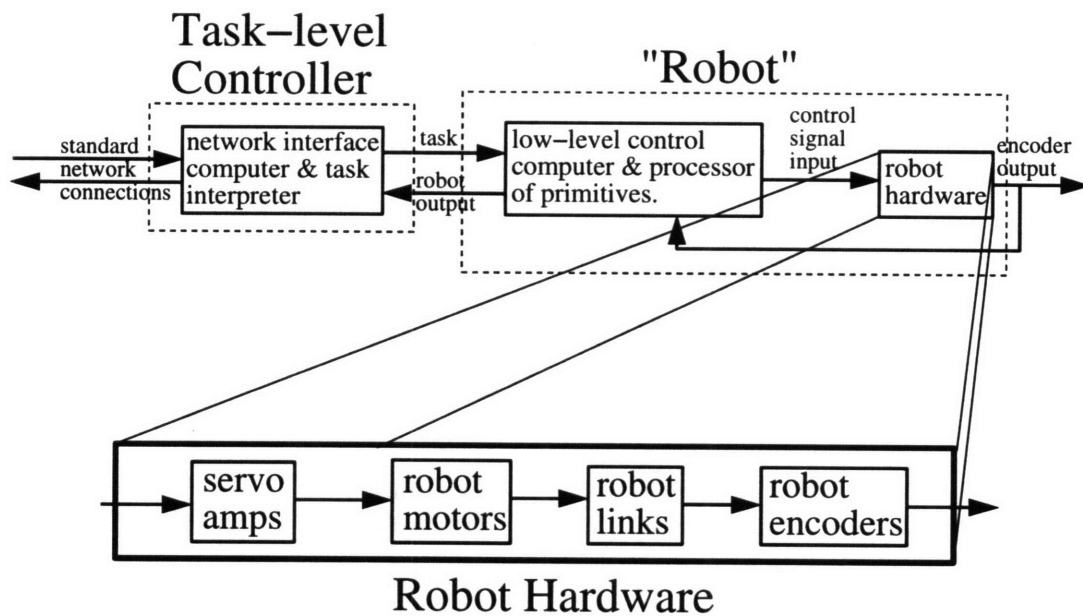


Figure 4.3: Robotic system hardware and connections including the network connection points. The network is a standard Ethernet Local Area Network with the common protocols and Unix based file sharing.

4.2.1 The environment

The robot's environment is meant to simulate a telemedical application. The surface (see Figure 4.2) is compliant and has local stiffness anomalies as might a portion of a human chest. The robot interacts with this environment through contact with the surface at arbitrary locations. The controller will allow this interaction under long time delays. Disturbances come in the form of blockages to a path or from possibly moving surfaces, but the disturbance bandwidth is very low, much lower than the time delay bandwidth. This environment can be considered semi-structured because the surface remains grossly fixed relative to the robot and arbitrary motions of objects occur infrequently and can be anticipated in advance.

4.2.2 System characterization

The closed loop low-level controller was tuned until a good response was achieved. Figure 4.4 shows some step responses on each of the three controlled joint axes. From these, the longest closed loop time constant was measured to be approximately 0.13 seconds.

$$\tau \approx 0.13\text{s} \quad (4.1)$$

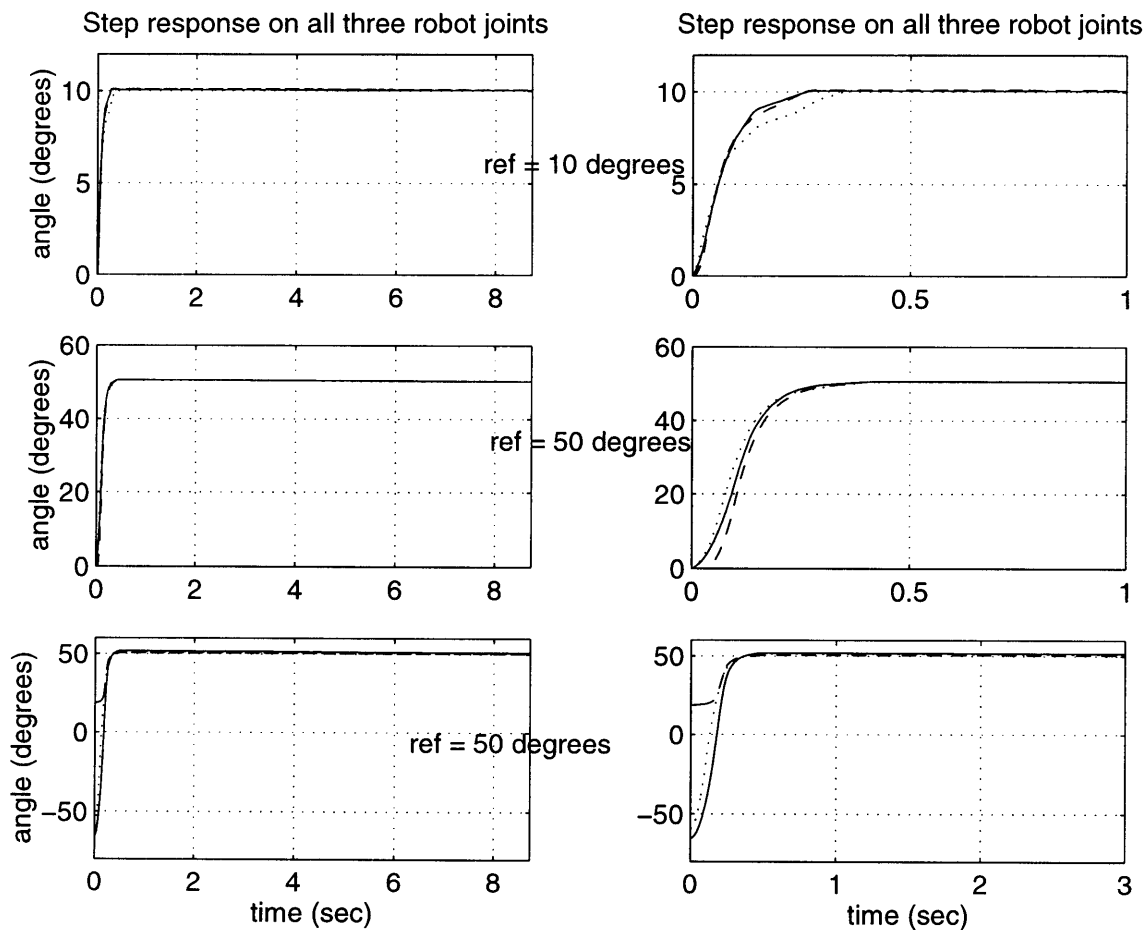


Figure 4.4: Step responses of the robot from various initial conditions on all joint angles. The time constants were measured from these and the longest one taken

to be the system time constant. Notice the zero steady-state errors and lack of overshoot desirable in a robotic system.

Several measurements of the network latency were performed. Figures 4.5 and 4.6 show the results of these measurements. *Forward latency*, L_f , is the total time from when a request is made until the robot receives the request, and *feedback latency*, L_r , is the time from when a primitive is completed until feedback is displayed to the user. *Total network latency*, L_n , is given by

$$L_n = L_f + L_r \quad (4.2)$$

which is the total round trip time for a primitive minus the task completion time. The figures show that forward latency is longer and less deterministic than feedback latency. Connection delays, a large part of forward latency, depend on network and server loads which can be viewed as stochastic processes. Refer to Figure 2.10 as a review of connection delays.

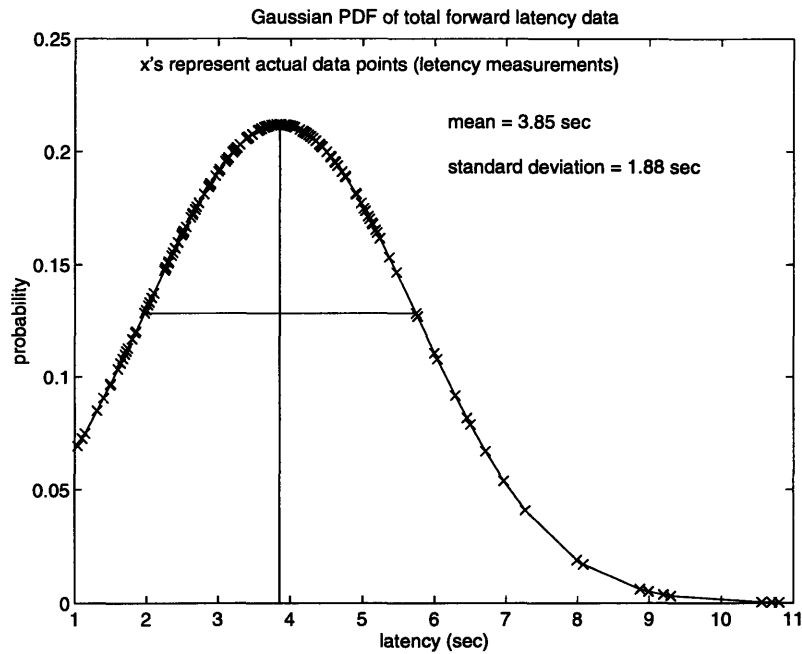


Figure 4.5: Measurements of forward network latency over a period of several days at various times in the day. The solid curve is the Gaussian distribution consistent with the data; the x's represent actual measurements of total forward latency. Periods of network breakdown were removed from the data. The large standard deviation comes from the variable network traffic and server loads occurring on the system.

The average values of the latency measurements are

$$L_f = 3.85 \pm 1.88s \quad (4.3)$$

$$L_r = 2.60 \pm 0.66s \quad (4.4)$$

which yields a total network latency of

$$L_n = 6.45 \pm 2.54s \quad (4.5)$$

Together with eq. (4.1) this implies

$$L_n = T_d \approx 50\tau \quad (4.6)$$

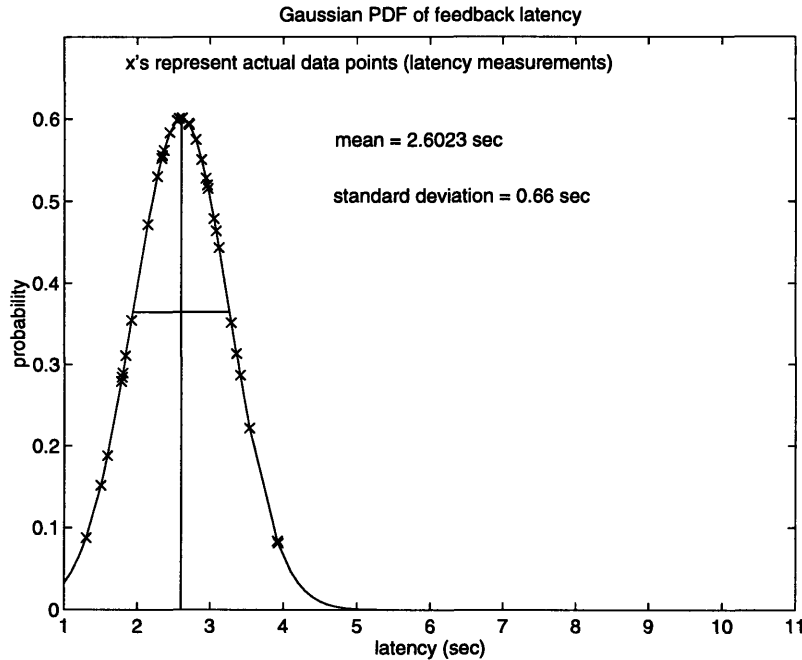


Figure 4.6: Measurements of feedback latency. Notice the smaller standard deviation than the forward latency. Feedback latency was primarily a computational delay, with only a small dependency on network traffic. Much of the variation is due to measurement errors.

This is well into the long range discussed in section 2.5.3 so a task-level control system is needed for this robot over this network.

Although we could not explicitly measure the time constant or delays present in the low-level computer-robot interface present in the IDE bus and control hardware, we concluded that task-level control was useful for this phase because the specialized hardware lent itself well to specific tasks, and the delays and time constants were both derived from the same source, transistor logic gate switching times, so their values were assumed to be approximately equal.

4.3 The Complex Tasks

4.3.1 The interaction tasks

We built three complex interaction tasks for use by Internet clients: “follow-surface”, “find-stiffest-spot”, and “map-surface.” “Follow-surface” has the robot touch the surface directly below the endpoint and follow that surface along a line projected onto the X-Y plane, applying a nearly constant force in the negative Z direction. The direction of the line (the input) is any of the seven

direction specified by $x=-1, 0, \text{ or } 1$ and $y = -1, 0, \text{ or } 1$. That is, it can move directly along an axis or along a 45° line in any quadrant. $x = 0, y = 0$ is no motion and is invalid input. Motion stops when the robot's position is out of its allowable range. The output is the lowest and highest spots recorded during the move. The diagram is given in Figure 4.7.

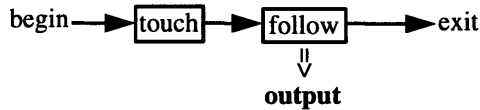


Figure 4.7: “Follow-surface” task-level diagram.

“Find-stiffest-spot” systematically probes the surface at prespecified increments and moves the robot over the stiffest spot found. The range of the search is the same as that used for “follow-surface.” The input is the search start position, and the output is the X-Y location of the spot and its measured stiffness. The task-level diagram is shown in Figure 4.8

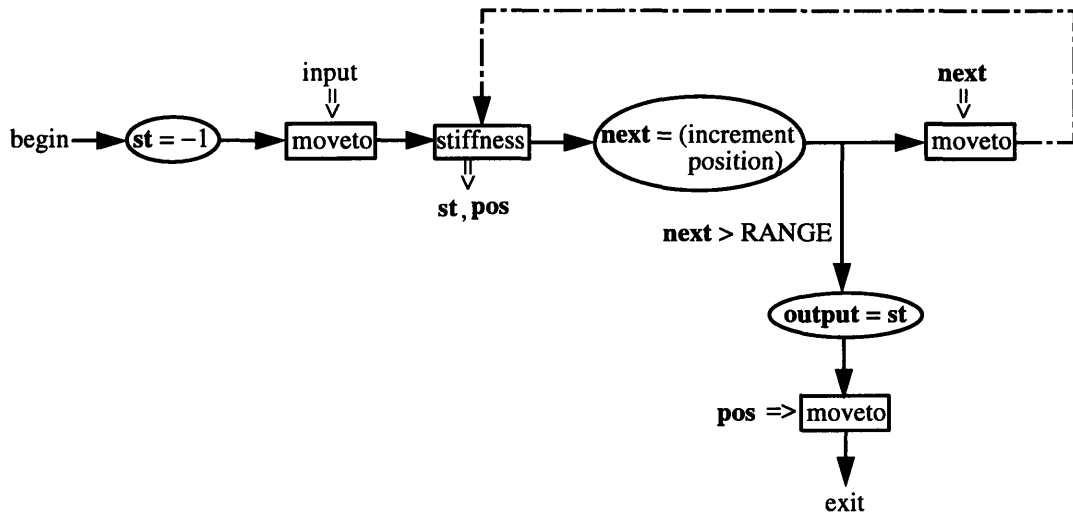


Figure 4.8: “Find-stiffest-spot” task-level diagram.

“Map-surface” is identical to “find-stiffest-spot” except it replaces the stiffness measurement with a “touch” and records the position of each touch. The output here is large, a collection of three dimensional data points, and is most efficiently stored in a data file which can be passed back to the calling agent as requested. Figure 4.9 shows the task-level diagram

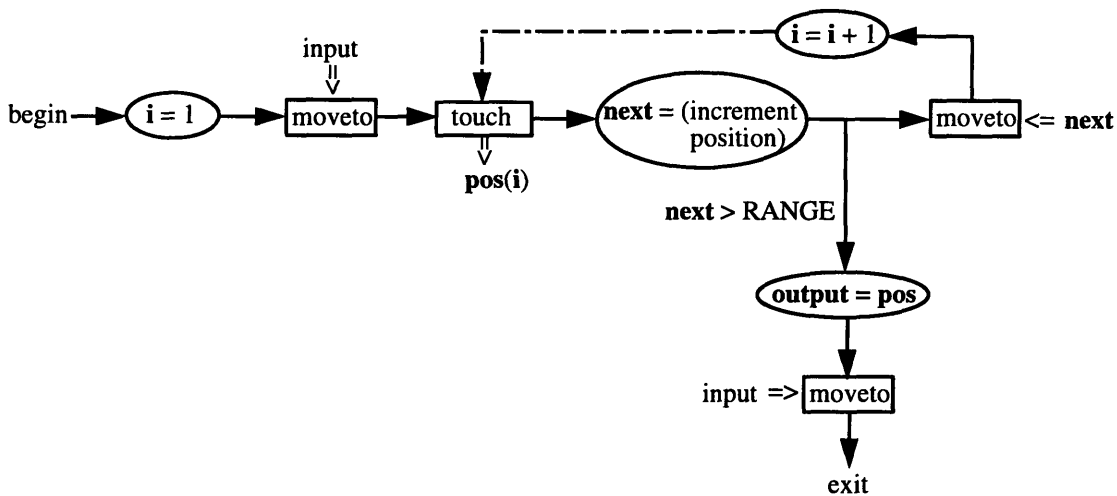


Figure 4.9: “Map-surface” task-level diagram. Note the similarity between this task and the “find-stiffest-spot” task.

4.3.2 Low-level control tasks (task-level implementation of the interaction primitive set)

The low-level control (complex) tasks, which make up the interaction primitive set, were specified in Chapter 3. Here we give the details of their task-level implementation and some practical issues. Note in the following diagrams that all the primitives are taken from the set given in section 3.4.5., but the “language” is identical in all other respects. All error output arguments are symbolic.

The sensor “query” task (or primitive), shown in Figure 4.10, is a short sequence of primitives.

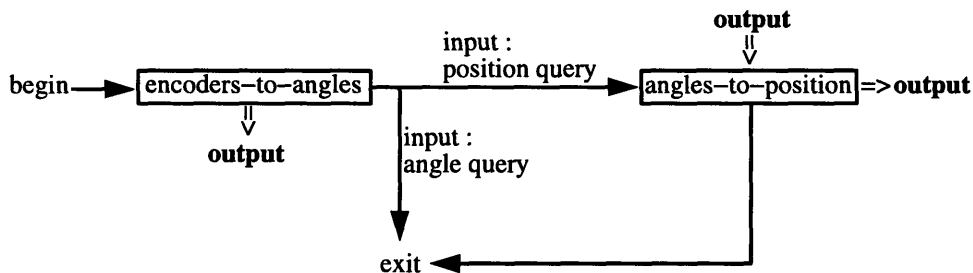


Figure 4.10: Sensor “query” task-level diagram. It is just a simple sequence of primitives.

The “moveto” primitive, shown in Figure 4.11, can have several variations. The most common is robot trajectory. This enters into the computation of reference values in the figure. Shown is the simple “moveto,” which moves each angle at some speed, thus incrementing the reference angle by a given amount on each loop. To move the robot endpoint along a straight line, one would first compute the speeds needed for each Cartesian direction based on their respective distances to the goal point, then increment a reference position by these amounts, and use the inverse kinematics to get the reference angles. Straight line motion implies that the robot may pass through a kinematic singularity, so caution must be exercised when performing this task. Also,

various velocity profiles can be used to achieve better completion times at the expense of increased task complexity and robot fatigue. Some example velocity profiles are shown in Figure 4.12. Generally, we might simply like to specify the speed of the move.

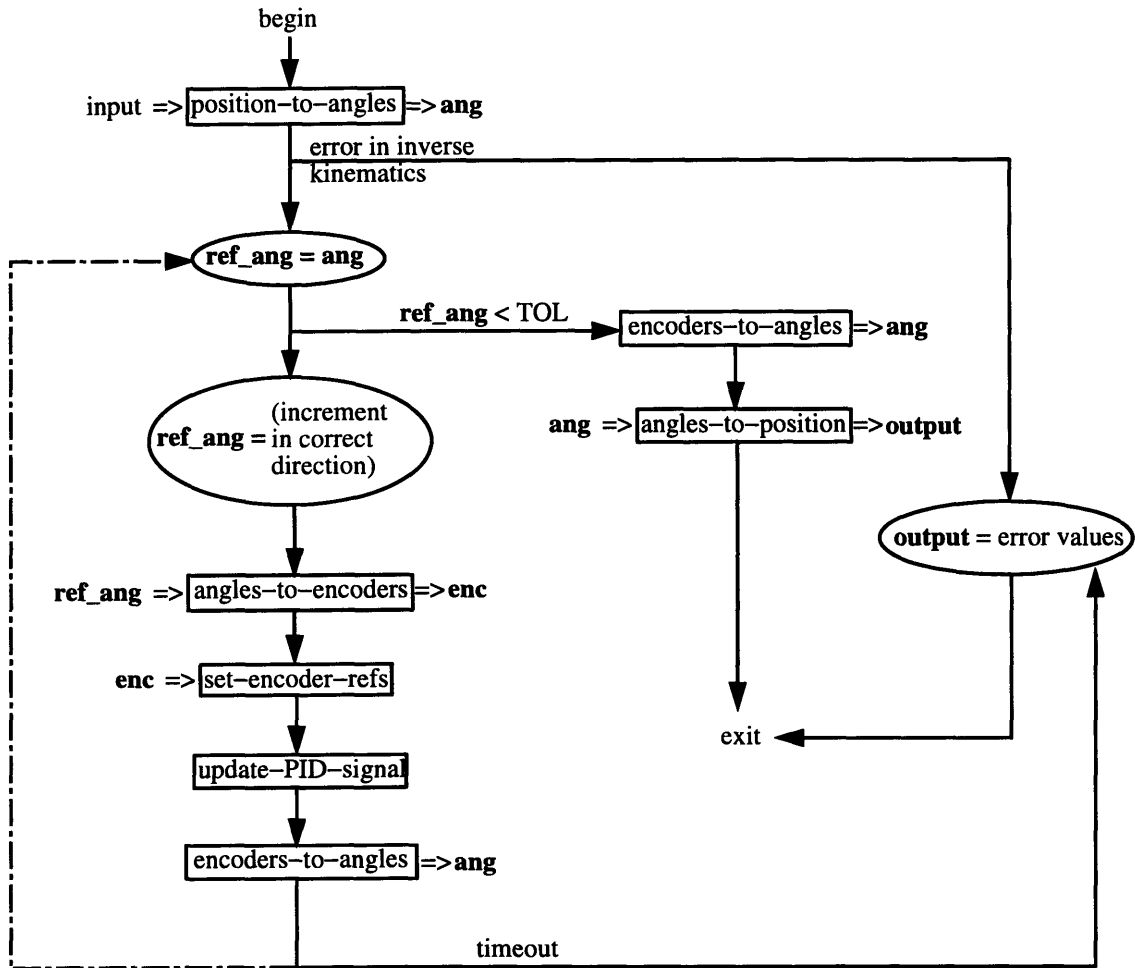


Figure 4.11: “Moveto” task-level diagram. There are variations on this task, such as forcing straight line motion, or changing the velocity profile.

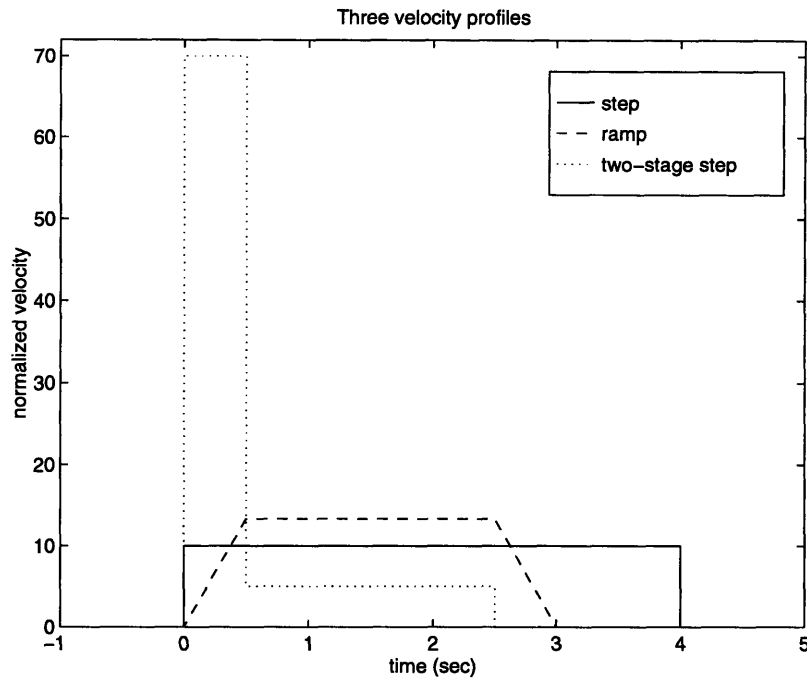


Figure 4.12: Comparison of three velocity profiles. The solid line shows the current profile used, which is a step. Large velocities can't be used because intolerable overshoot would result. The dashed line uses ramps which allow greater speed at the expense of increased programming complexity, and still do not totally prevent overshoot. The dotted line uses velocity stages, or multiple steps to achieve maximum velocity, with even more complexity and less reliability when the desired motion is small. Robot hardware fatigue also results. The integral under all curves is equal implying the same final endpoint position.

The “touch” primitive is shown in Figure 4.13. A practical issue is specifying the direction of motion while “feeling” for a touch. We chose to use symbolic input arguments which specify one of 26 directions in three dimensions: x, y, and z can be -1, 0, or 1, with 0, 0, 0 being no motion and an error. Figure 4.14 shows these directions. The speeds of these motions is a prespecified, built-in, value, which could also be an input argument.

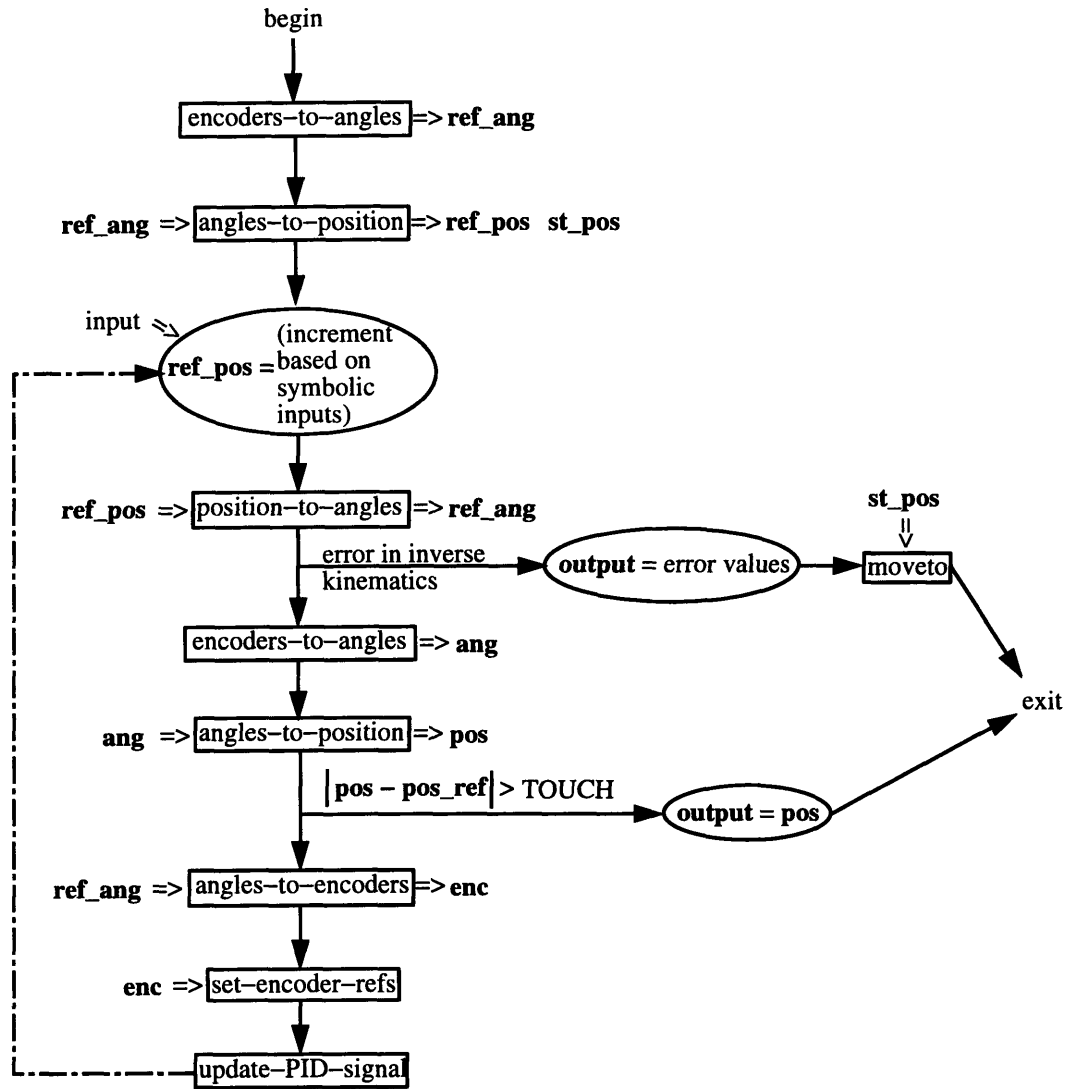


Figure 4.13: “Touch” task-level diagram. The input arguments are a symbolic representation of a direction shown in Figure 4.14.

The “stiffness” task/primitive is really just two “touch” tasks combined in a sequence, and the input is the same one as used for “touch.” The computation of stiffness is taken indirectly by measuring robot motor torques. This is merely a rough measure but sufficient to determine large differences in stiffness. Forward dynamics of the robot can be used, but were not, to measure the stiffness more accurately. The task diagram is shown in Figure 4.15.

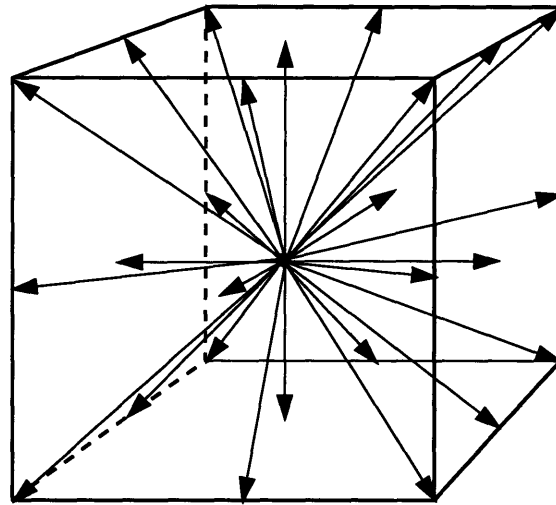


Figure 4.14: 26 Cartesian direction specifiable in “touch” and “stiffness.” 0, 0, 0 is no movement and returns an error value when called with these.

Finally, the “follow” primitive, shown in Figure 4.16, can trace the contours of a surface, taking symbolic inputs, to specify an X-Y direction: x and y can be -1, 0, or 1, to yield seven directions (again 0, 0 is no motion). By setting the Z reference position to a small number less than the current Z position, the robot effectively applies a small downward force during the motion.

Comparing the task-level diagrams from this section with those of the previous section shows that their construction is identical. The only distinguishing factor between them, aside from the obvious superficial differences, is the primitives which are used. The *primitives from one level can be thought of as complex tasks at a lower level*. This is the basis of task-level control, and we can compare this with Figure 3.4, which shows how tasks are built from other tasks. This power-

ful idea of “zooming” in “closer” to a task to see how it is composed allows complex behaviors to be built which rely on reliable sub-tasks, or primitives.

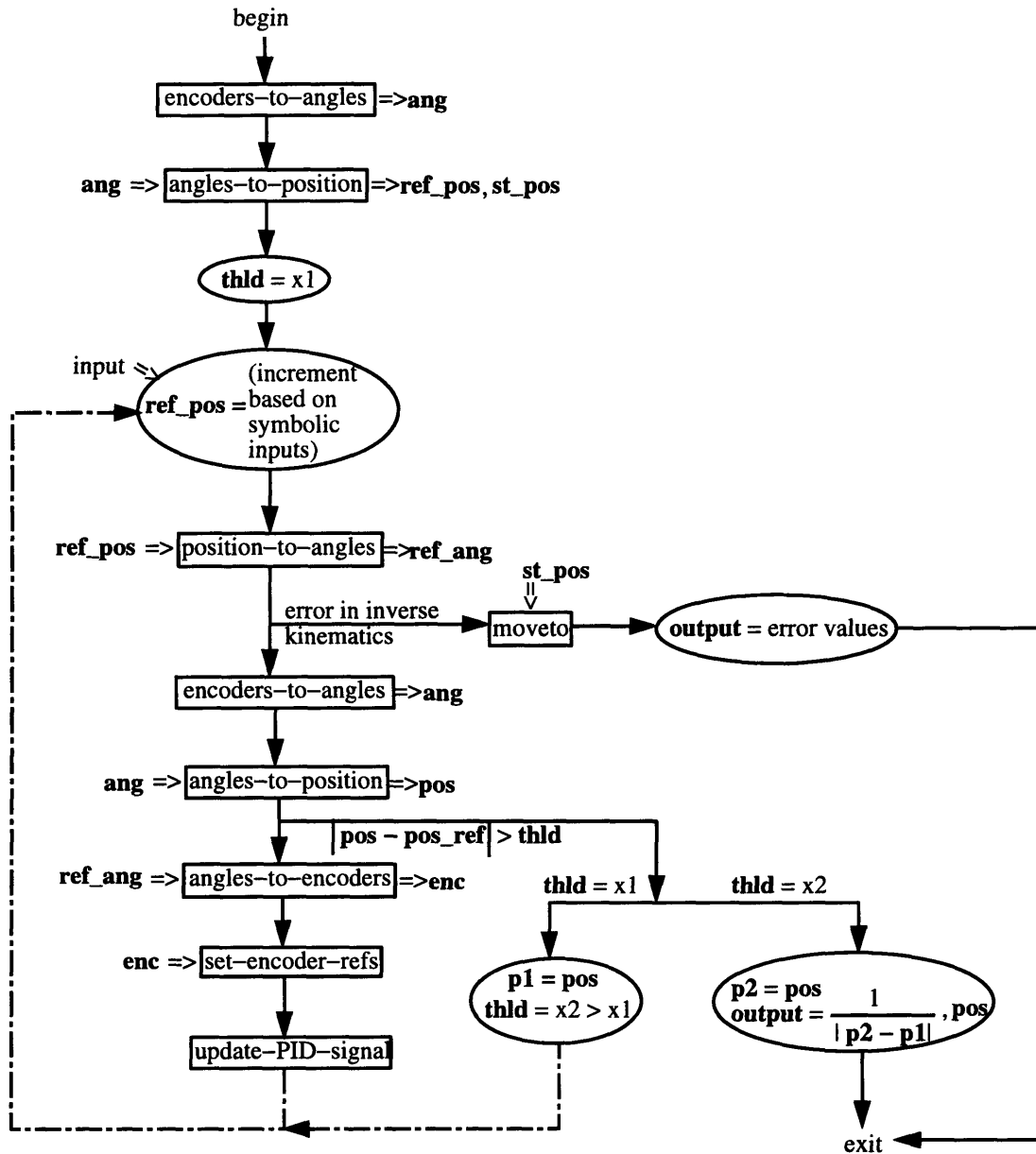


Figure 4.15: “Stiffness” task-level diagram. This is a combination of two “touch” primitives, but with enough low-level differences to make it into a separate primitive, rather than a complex task built of the other primitives.

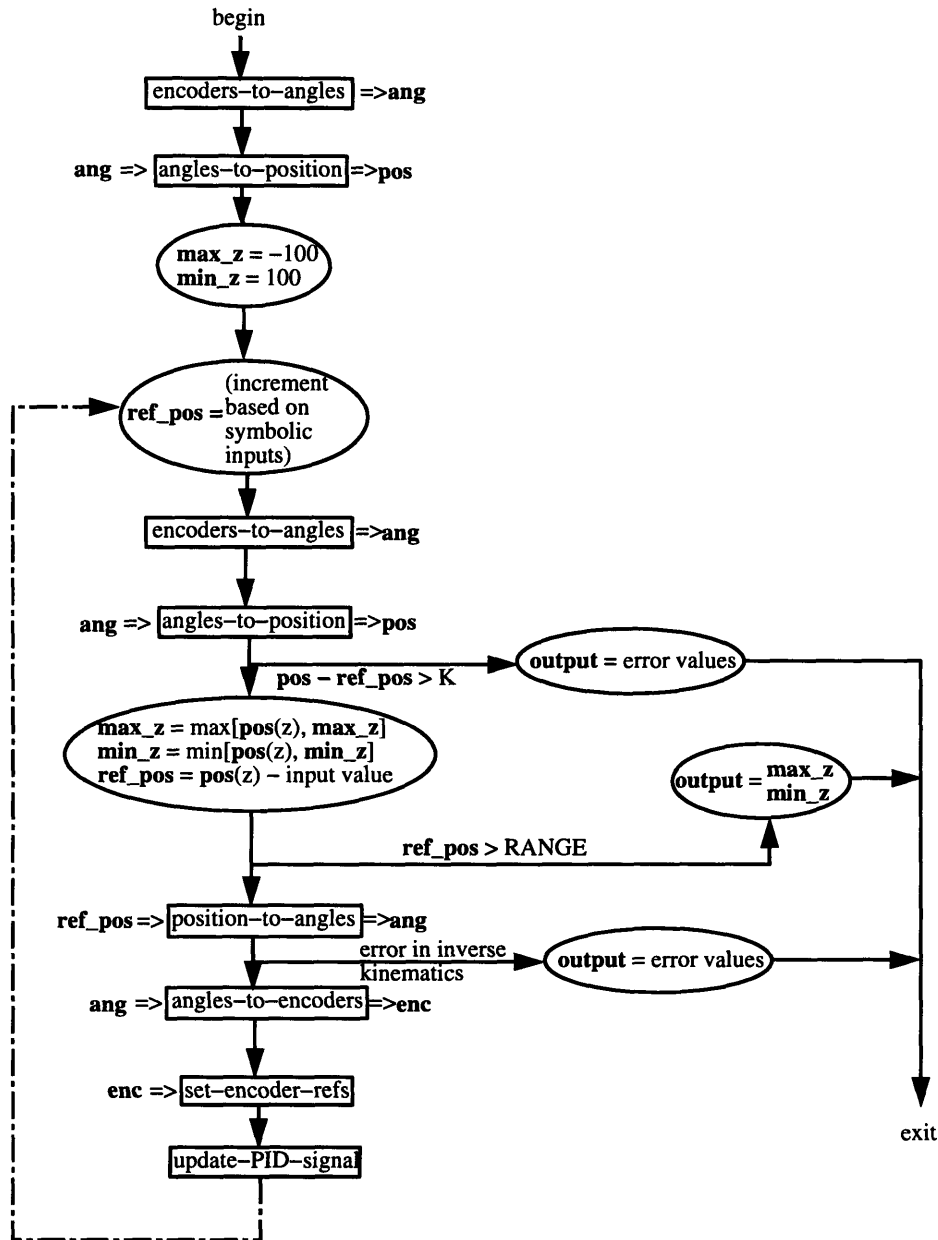


Figure 4.16: "Follow" task-level diagram.

4.3.3 Default execution

We execute a position "query" after each primitive in the low-level control set is executed. The values are used to give the final position to the user whenever it is desired rather than requiring the user to explicitly ask for the position when it is needed. Although not done so, the values can also be used to check the outputs of some primitives which return position. This is a self diagnostic action, which is not used. However, self diagnostic primitives are another good example of default behaviors that can be executed in the form of primitives after each explicitly requested primitive is executed.

4.3.4 Human user friendly tasks.

We have created explicit instances of the “moveto” task by prespecifying its inputs. This is known as **homing**. Rather than having a user spend the time to input a desired robot position or configuration, he or she can simply execute one of these to move the robot to a known and useful position.

The “sleep” primitive is run by default after a given time-out period. Sleep moves the robot to a known stable position and executes “turn-off-motors” to shut the robot off. “Sleep” thus lets the robot move under its own weight, and if one were to query the sensors directly after “sleep” with the robot in some arbitrary configuration, the next sensor query might reveal a different position for the robot. This seems to be in violation of supervisory stability. However, once the motors are turned off it is assumed that the robot is no longer being supervised, or it is being handled locally for repair or maintenance, so stability does not apply. Also, “sleep” sends the robot to a standard location before turning off the motors. Under most circumstances this will leave the robot resting on some surface or in a stable configuration (such as pointing straight up). This is a recommended procedure to include in any type of primitive which turns off power to the robot.

Chapter 5

Analysis and Lessons Learned

5.1 Introduction

The purpose of this chapter is to present the results of experiments performed with the system. Section 5.2 describes these experiments and section 5.3 presents the numerical results. We present results of measurements directly on the primitive set, such as primitive completion times and latency ratios, and we describe the versatility of the set, by reviewing how well the user was able to perform a task. Suggested improvements, deletions, and additions to the set are suggested. Section 5.4 briefly outlines some unexpected or uncontrollable factors which affected the experiments and how to eliminate them in the subsequent experiments.

5.2 Experiments

One of the advantages of task-level control is the ability to automate tasks through proper planning. The primitive algorithms discussed in the previous chapter are all amenable to computer implementation, which will both speed up the process and require little or no human supervision. Some experiments were performed to determine the effectiveness of automation versus human or manual task completion.

The task to find the stiffness of the hardest point on the surface was implemented on a computer, and the completion time was compared against that of a human operator performing the same task manually. Success rates were also recorded, where success was defined by the correct location of the hardest spot being returned.

To determine the versatility of the interaction primitive set, we performed an experiment to determine dimensions of a stationary block. The user was given an unlimited amount of time to complete this task. With the availability of visual feedback, the user was able to quickly determine the general location of the object, but the exact location (center) and edges were more difficult and required some ingenuity and trial and error on the user's part.

As an initial measure of the primitives' effectiveness, we compared completion time, T_t , from the request to the average total network latency of the system, given by eq (4.2) and whose value for this system was given in eq (4.5), both repeated here.

$$L_n = L_f + L_r$$

$$L_n = 6.45 \pm 2.54s$$

The primitive can be characterized by a quantity we call the *latency ratio*, Λ , given by

$$\Lambda = \frac{T_t}{L_n} \tag{5.1}$$

A value of Λ greater than $\Lambda = 1$ is acceptable, though normally not desirable, but anything larger than $\Lambda = 3$ is ineffective. Very low values of Λ are acceptable for a few primitives; however, if all

primitives in a set have low latency ratios, it may be beneficial to increase the complexity of some of the primitives, since it implies that the primitives are not accomplishing as much as they could during each latency period. “Query” is naturally short because it executes electronic instructions as fast as possible. The latency ratios of all the primitives in the set were measured for a wide range of inputs using the average total network latency given in eq (4.5).

5.3 Performance of the Interaction Set

5.3.1 Complex interaction tasks

The experiment to find the hardest spot on the surface was completed by both the user in a manual, step-by-step (or move-and-wait) process by three subjects, and by the computerized, automatic version. The completion times are given in table 5.1. As can be seen, the user times are significantly longer. All experiments resulted in the correct identification of the hardest spot.

User Type	Completion Time
automatic	90 ± 1 seconds
automatic	93 ± 1 seconds
experienced human	590 seconds ± 10 seconds
experienced human	780 seconds ± 10 seconds
inexperienced human	3000 seconds ± 20 seconds

Table 5.1: “Find Hardest Spot” task completion times

Even practiced and experienced user could not approach the completion times of the automatic task. This is mostly due to network latency and the fact that the user was required to record certain results for later use, which took some time. This is a user interface problem, and if the user was able to more quickly record the data, an increase in speed would result.

Also, the time taken to input arguments caused the greatest delay. The user interface could be improved by allowing direct control of the robot’s endpoint through graphical means. The user could point to a particular spot on the graphics window with a mouse and click the mouse to indicated that was the desired position. The display of information was also somewhat tedious. Text was fed back to the user with the necessary data, but visual feedback, though not required, was often requested by the user for confirmation of some result. The visual feedback replaced the text feedback, so both could not be viewed simultaneously. This could be changed in future implementations. The user interface is clearly and important aspect in task-level supervisory control.

The robustness of the mechanical system was an important aspect to determine. Robustness aspects include repeatability, accuracy, and disturbance rejection. The system was operating continuously (not including power outages) for four consecutive weeks without failure or false feedback or reports. The repeatability and accuracy of the positioning was about 1 mm. There were several disturbances the robot was able to handle well. When the robot was mechanically blocked, it returned most likely cause of the error. When the robot was requested to complete a move that

was not physically possible, it also returned an error, and when requested to perform uninteresting or useless actions, this was also reported. Actions such as the “touch”ing or finding the “stiffness” of itself were examples. In these cases it simply did not perform the action and suggested that the requesting agent try another action or correct the arguments of the previous one. The robot also reliably turned itself off when it was not in use. This significantly cut down on power consumption and hardware fatigue, and eliminated the noisy chatter of backlashing gear teeth.

The robot was unable to handle power shutdowns and other hardware failures well. One problem is the lack of an auto-executing start-up program for the robot. This problem could be solved by installing an auto-executing robot server, which first measured its workspace, set its zero position to a preprogrammed configuration, then started the server program.

5.3.2 Speed of the primitives in the interaction set

All of the primitives’ latency ratios were measured for a wide range of inputs. Table 5.2 shows the results of the measurements. The consistently slowest primitive was “stiffness” because it required a slow robot speed to be sensitive enough to measure two touches on a single pass. Much of the execution times depended on the inputs and on the robot’s starting point. As a normalizing factor, we started all appropriate primitives from a standard position. The “moveto” and “line” primitives have rates instead of average times given because their execution is so dependent on starting and ending position. “Line” is an implementation of moveto which moves the robot endpoint along a straight line. The speeds given are constants. For “moveto” the speed is that of the slowest link.

Primitive	Average Λ	Maximum Λ	Minimum Λ
“moveto”	27.5°/sec	N/A	N/A
“line”	0.13 m/sec	N/A	N/A
“touch”	2.03	2.07	2.00
“stiffness”	2.12	2.15	2.08
“follow”	.68	.75	.57
“query”	.15	.17	.13

Table 5.2: Primitive execution times and latency ratios

The primitive “query” is purely computational and so has a deterministic completion time, which is much shorter than any of the primitives which execute robot movements.

From this data we see that there is a significant spread of the ratios. Some are much less than one (“query”) as expected, some are near one, and some are greater than 2. “Touch” and “stiffness” can often be made to have $\Lambda < 2$ by a thoughtful choice of starting positions. However, sometimes $\Lambda > 2$ can not be avoided especially when there is uncertainty in the environment.

Some improvements to the primitives were suggested by the data. Increasing the speed of “touch” and “stiffness” by increasing the rate at which they move the robot would reduce latency ratios but would decrease the already limited resolution of the primitives. The robot would need a larger force to sense a touch and the repeatability of the stiffness measurement at a particular loca-

tion would be worse. We deemed it was more valuable to have better resolution and sensitivity over speed, especially with automatic task execution in mind.

“Moveto” could be sped up by using some of the velocity profiles shown in Figure 4.12. Care must be taken so as not to induce significant overshoot.

The “follow” primitive was difficult to improve. It could have been sped up by increasing the speed at which it moves the robot across the surface. Again, this would have led to less resolution and possibly inaccurate readings of the surface heights.

The robustness of the primitives was important, as well. “Moveto” had a time-out installed on it, so that if a move took too long, the primitive would cease execution and return an error code for this error. The system would then tell the user that this was due to mechanical blockage. It also returned an appropriate error if the desired position was kinematically not achievable.

“Touch” and “stiffness” were not allowed to be executed when the action would cause the robot to enter into a singularity. It was further restricted to performing movement only in the negative Z direction, since for our semi-structured environment, there were no interesting environmental aspects in any other direction. If a primitive was executed in violation of these constraints appropriate error codes and messages were returned.

“Follow” checked to make sure that its position was close to the desired reference position during a move. If it was too far away, it assumed blockage and returned an error. The robot was unable to handle irregular or very soft surfaces during a “follow” primitive. It would not be able to lift itself over high blocking points or be able to come out of soft areas that it had pushed down too far. See Figure 5.1 for graphical clarification. A possible correction would be to reduce the amount of downward pressure applied during “follow” or to install a part to the algorithm that would lift the robot when it encountered what seemed like a blockage.

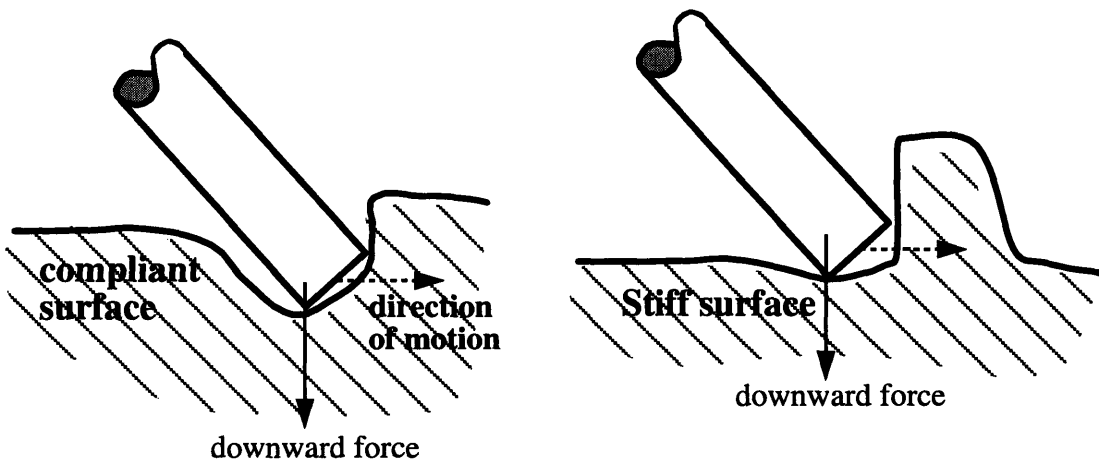


Figure 5.1: How the robot behaved for very soft (left) or irregular surfaces. Since a nearly constant downward force was applied, the robot rarely would be able to “lift” itself out of holes it created by this pressure, and blocking objects created the same problem.

In all cases tested so far, the only time when a false error was reported was when, during “moveto” the robot was trying to position at the end of the move. The integrator was sometimes too slow to reduce the steady state error enough to get the desired acceptable accuracy, and “moveto” timed out because it would not be within this accuracy for more than the time-out

period. Three fixes suggested themselves. Decrease the desired accuracy, increase the time-out length, or increase the integral gain. Decreasing accuracy and increasing time-out (currently set at 7 sec) would decrease performance, upping integral gain would lower the controller bandwidth, requiring slower speeds for the robot. It was found that the speeds at which the robot normally operates, however, were much less than the bandwidth, anyway, so the integral gain was increased to solve the problem. However, if high speed positioning becomes necessary, the integral gain would need to be reduced to avoid overshoot.

5.3.3 Versatility of the primitive set

The versatility is important in designing a primitive set. The test for our primitive set was to have the user find the dimensions of a fixed block on the surface using as much time and any method he or she chose. Typical results are in table 5.3, which show the accuracy of the measurements is low. As with the stiffness measurements, this could only be used for gross estimation. The user relied heavily on visual feedback, which was of low resolution, fixed angle and one dimensional, which did not give a full view of the robot's position relative to the block. This was seen as a problem in (Mar 1985). Another problem was that the endpoint of the robot was very blunt and the angles at which it touched surface changed depending on the configuration. This did not give consistent readings of the height of the robot at a touched point. A sharper or pointed endpoint would have been useful.

The surface mapping algorithm given in the previous chapter with a reasonably high resolution turned out to be the best method for finding the dimensions. Although this was slow it gave the best accuracy.

Measured Dimension	Actual Dimension
0.10 meters (length)	0.06 meters
0.14 meters (width)	0.10 meters
0.03 meters (height)	0.025 meters
$x = -.17 \ y = -.08 \ z = .27$	$x = -.17 \ y = -.09 \ z = .26$

Table 5.3: Results of primitive set versatility test

Another unexpected use of the “stiffness” was to allow the robot to push reasonably hard on a surface. Repeated executions of “stiffness” at the same point forced the robot to push harder each time. Although the stiffness readings increased with each subsequent execution, if the purpose was only to push hard on a surface, this would work well. This could be incorporated as a new interaction primitive, called “press,” which takes a desired pressing force as input.

With regard to set versatility, much of the capability of this robot has been achieved in this primitive set. The robot is primarily capable of accurate and high bandwidth movement and of low resolution force sensing (from current measurements). The primitives in this set utilize all of these capabilities to varying degrees depending on the application. The limiting factors on performance are the user interface and network latency.

5.4 Other Factors Affecting the Experiments

Besides arbitrary inputs and starting positions, there were a number of factors which affected the performance of the system. The most noticeable was the highly variable and unpredictable network latencies. Our system relied heavily on a public computer which was the Internet server for the laboratory in which the system was located. The popularity of the lab's research dictates a large volume of accesses from other clients to this server. Often the server was overloaded, and would not give a connection for the robot program. This would severely limit task completion times for tasks being performed by hand.

The experience of the user and his or her familiarity with the system dictated how much thinking each user needed to do before deciding on the next primitive to execute. When the user was first introduced to the system, the first reaction was to make sure exactly what was happening after each move. Once familiarity and confidence in the primitives, and knowledge of the primitives limits, were gained, the user began executing primitives one step ahead of time, in an open-loop fashion. The system only allowed one request to be cued while the current primitive was being executed. The primitive executed after the current one finished was the most recently requested one. In almost all cases, no more than one move in advance was needed.

Finally, there were a number of parameters set within the control program that altered performance of the system. Initially these were used for debugging purposes, but many remained because they often needed to be changed after some other part of the system was altered. These include, but aren't limited to, joint speed for "moveto", speed for "line", positioning accuracy, controller gains, time-out lengths, sensitivity for "touch", force required to measure "stiffness", speed of "touch" and "stiffness" robot movements, motor current output limits, configuration of home positions, speed of "follow" movements, and positions where the robot would return error if certain primitives were requested. Changing any of these could significantly affect the performance of the system, but are set according to specific preferences. That is, there is no optimal combination of these parameters.

Chapter 6

Conclusion

6.1 Introduction

This chapter concludes the thesis, starting in section 6.2 with a brief review of the work done. Section 6.3 describes some of the contributions to the area of task-level and supervisory control research and engineering that this work has made. Section 6.4 then presents some possibilities for future work and further research in and related to these areas. Finally, section 6.5 concludes with final remarks and a summary of the important points of the work.

6.2 Review

The significant latencies often encountered in networked and teleoperated robotic systems were characterized and described in terms of their deleterious effects on the control of such systems. A review of these effects for analog and discrete-time control systems was presented to give the reader an idea of the difficulties of these effects, and to suggest the method by which these time delays can be overcome. This method is task-level supervisory control.

Previous research in task-level control and supervisory control was presented separately and then the major concepts were combined to yield a systematic approach to designing control systems with latencies as pure time delays. These latencies were characterized in relation to the speed of the robot being controlled, allowing a normalized quantity to dictate certain aspects of the control system design. The ranges of these normalized latencies were broken down into three overlapping ranges. Particular properties of these ranges were discussed in terms of methods of control. Very short delays could be analyzed and designed around in the traditional manner, since the delay could be thought of as merely part of the robot's dynamics. Midrange and longer time delays began giving the traditional methods trouble with stability and performance, so the new method was described. Task-level control does not require long time delays to be successful. It is a general method of automation.

A particular range of time delays, the long range, was selected for close study. This is mostly because of the real system we used, to which the application of the principles were applied. The real system was then characterized and identified for this design, and a control system was built for it. Tests and experiments were performed to measure the performance and the results wrought some practical conclusions.

Although the system described in Chapter 4 was designed to compensate for time delays, and network latency in particular, it was also the primary limiting factor in the performance of the system. The user interface was also a limiting factor, but as far as system performance is concerned, this was not an issue because the automated tasks did not use the interface. A picture of our user interface is shown in Figure 6.1

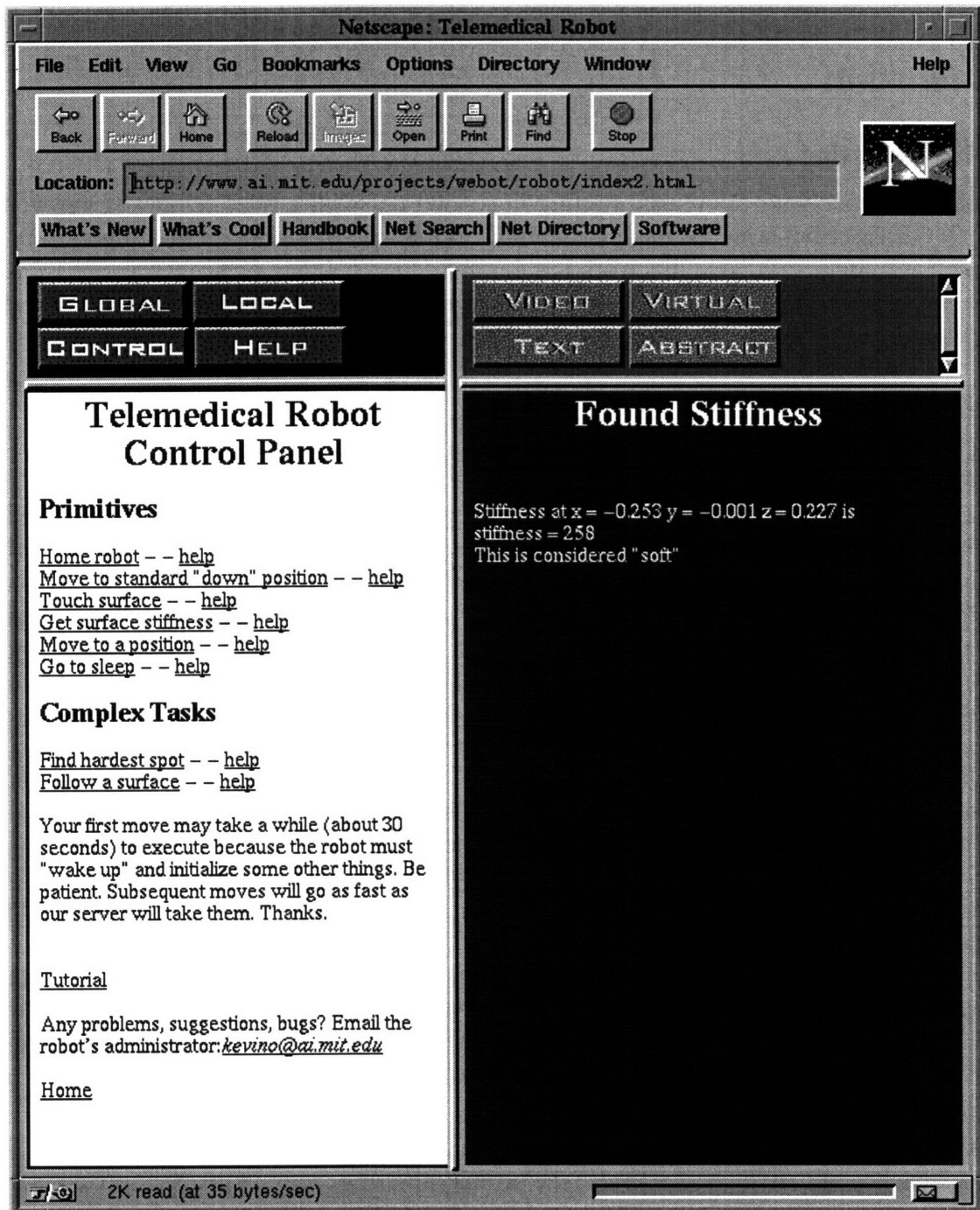


Figure 6.1: The user interface for the task-level supervisory robot controller. This was a World Wide Web site, and the interface was designed around Netscape Navigator.

Primitives are stand-alone behaviors that were written into a logic sequence to allow automation of their execution. Two sets of primitives were described and implemented, each set completing a taxonomy of actions, one for higher level interactions and one for low-level control actions.

Finite state machine logic was used as the glue to build automated tasks. Automated task execution was used to compensate for network latency.

When used with automated tasks, the system performed well. The task completion times were significantly faster than those of a human user. The human was able to complete the same tasks, but at often very slow rates. The user interface and network latency were the primary causes of these slow times.

The task-level supervisory controller was general and useful enough to place the user interface on the Internet's World Wide Web (<http://www.ai.mit.edu/projects/webot/robot> ; see Figure 6.1). It was robust and reliable under heavy load conditions.

The primitive sets allowed many interaction tasks to be completed over distances and with large time delays. It also allowed both interaction and measurement of the environment with minimal sensory equipment using recently worked out principles of data interpretation. The planning and testing of tasks to be automated later was also an easier task. The user interface allowed visual views of the robot's environment as well as text feedback with actual position and stiffness data measured by the robot.

The position measurements, however, were not accurate in relation to the positioning accuracy of the robot (about 1mm). Causes include robot design, trade-offs in primitives, lack of controller sophistication, and user interface design. The design goal for the robot, built before this study began, was high bandwidth and low inertia, with no end effector design. We used it primarily for low bandwidth purposes with environmental interaction better suited for robots with end effectors. Some of the trade-offs in the primitives were, in fact, their speed of execution. Others were their sensing resolution and accuracy, as well as their complexity. The user interface, though robust and useful, was not optimized for any one group of users since it was to be presented to a very large and general audience, most of which would not be familiar with robot control.

Also, the controller did not allow very fast execution of some tasks. Causes include time delays and slow primitives. Optimization of some tasks could be done by identifying the specific purpose and actions of the tasks and designing automatic versions of them. Other optimizations on the primitives could be done by varying the many design parameters programmed into each primitive.

We designed the system as outlined in (Brock 1993) by having a set of tasks, which were, in turn, composed of simpler tasks, and so on. In this thesis, we had three levels, the complex interaction tasks, the interaction primitives, and the low-level primitives. The supervisory element of the system was brought about by allowing some of these tasks to be executed under human control via the Internet.

Some aspects of task-level and supervisory control were not used in the design of this system. Here we present those that were most important and how their inclusion would change and improve the system.

Predictive displays, a supervisory method mentioned earlier, could have been used to allow the user to foresee the robot's action before it occurred. This would allow better planning and increased visual cues for the user. However, because of the unstructured part of the environment in which the robot was acting, this simulation would have been necessarily limited. Most likely, it would have included constructors to allow for the environment to be built as the robot explored it. Thus, it would have been a self-building simulation. Programming complexity and a desire to concentrate on the task-level portion of the system prevented us from installing this on the sys-

tem. This type of display is now becoming more easily implementable with standardized graphical display programs, such as VRML.

Related to this was the use of high-fidelity dynamic simulations assuming a more structured environment, as discussed by Narasimhan (Narasimhan, 1994). This would have allowed predictive task planners better information to plan behaviors and actions. Again, this would lead to better automation. However, it would have significantly slowed the system because of the high complexity of the program and the computation it would require.

Direct, semi-real-time robot control is now possible with the Internet using Java because once a connection is established, a small bandwidth data line could be held open. Continuous reference positions could be sent from some type of user interface to the robot controller which could servo to those positions. At the same time, joint angle positions could be passed back over the same line to update a graphical simulation of the robot. This is all low bandwidth communication that allows semi-real-time control of the robot with visual feedback. The reason this wasn't implemented was the newness of the technology available to do this. It's use is already being studied.

Finally, the use of high performance sensory techniques, as described by Eberman (Eberman, 1995) could have been used to achieve better environmental measurements which would have allowed faster performance with primitives such as "touch" and "stiffness". These methods require a good knowledge of the robot's parameters (such as inertia and motor torque constants) which were not available and difficult to measure accurately, and the program complexity would again have slowed the system significantly, so there may have been very little net gain. Also, much of the data taken for these methods would be analyzed off line.

6.3 Contributions

We were able to combine aspects of task-level control and supervisory control in this thesis. By studying semi-structured environments, common in many robotic applications, and by holding to the desire for the human user's intelligence to be a part of the system, we made a system that was neither fully automatic, nor fully manual, but could be customized to achieve either level if that was what was needed for a particular part of the robot's operation.

We developed a taxonomy of task primitives useful for specific interactions over long time delays and for general low-level feedback control of a 3 DOF robot, and we developed sufficiency criteria for supervisory stability and for primitive set sufficiency.

We were also able to successfully control a robot under highly variable time delays by allowing users to request tasks while others were being performed. This allows the user to use the latencies to advantage in planning for the next step. Extensive long range planning by the user before task execution was an option, but in environments which are not fully structured, this planning is often wasted because new or unexpected situations arise which require unplanned actions.

We built a networked telerobotic system which was reliable and robust. It's interface, though not fully optimized, was easily configurable to be used by any number of robots, all of which could communicate to each other through very standard network protocols. The time delays involved were compensated for using the principles discussed in this thesis.

Finally, with this networked principle, we were able to demonstrate the concept of having a robot perform numerous "useful" tasks over long distances using the Internet. Internet browsers, available to the general public, could be used with the user interface we built to send requests and to get both visual and data feedback. The system performed well and was successful primarily

because of the task-level controller. The set of primitives developed were general enough that a type of finite state machine logic could be used to develop automated tasks over the Internet. Using the system we assembled, global users controlled the local robot within its environment with no training using standard and readily available tools. Users located in Japan, Korea, France, The United Kingdom, Canada, Australia, and The United States of American were among the many who participated in the unofficial experiment.

6.4 Future Work

There are several research avenues that can be taken from this thesis. The area of telerobotics is large and developing rapidly. The design of control systems, robot hardware and architecture, computer networks and programs, planning, task building and simulations are all areas in this thesis.

6.4.1 Control system design

In the design of control systems there is further work to be done to allow faster task execution. Most of this work would rely on previously known areas, such as adaptive robot control and system identification. Further work into the design of primitives should also be done, as well as a general theory about task-level supervisory control, which formalizes the ideas of stability and convergence of a particular control scheme.

Also, there is significant work to be done on control over latent networks where a low bandwidth channel can be opened to pass control data across. Simulations which require only small and intermittent updates and robots which require joint positions as control inputs are surely to be integrated soon over the Internet with the advancement of the computer independent Java programming language.

6.4.2 Behavior building

The building of behaviors for simulated entities has been studied recently, and much of this research can be applied to the building of robot behaviors and tasks. Using finite state machine logic and a given set of task primitives, a whole host of behaviors can be defined and implemented for a robot. These behaviors can be simulated and tested, as well, before they are actually programmed into the robot. In fact, they can be tested in real time while the behavior is being modified. This is a very exciting concept in behavior building and automation. Figure 6.1 shows a primitive behavior building tool which uses finite state machine logic. This graphical interface could be tied to a running simulation and edited while it is running. This work is already in progress.

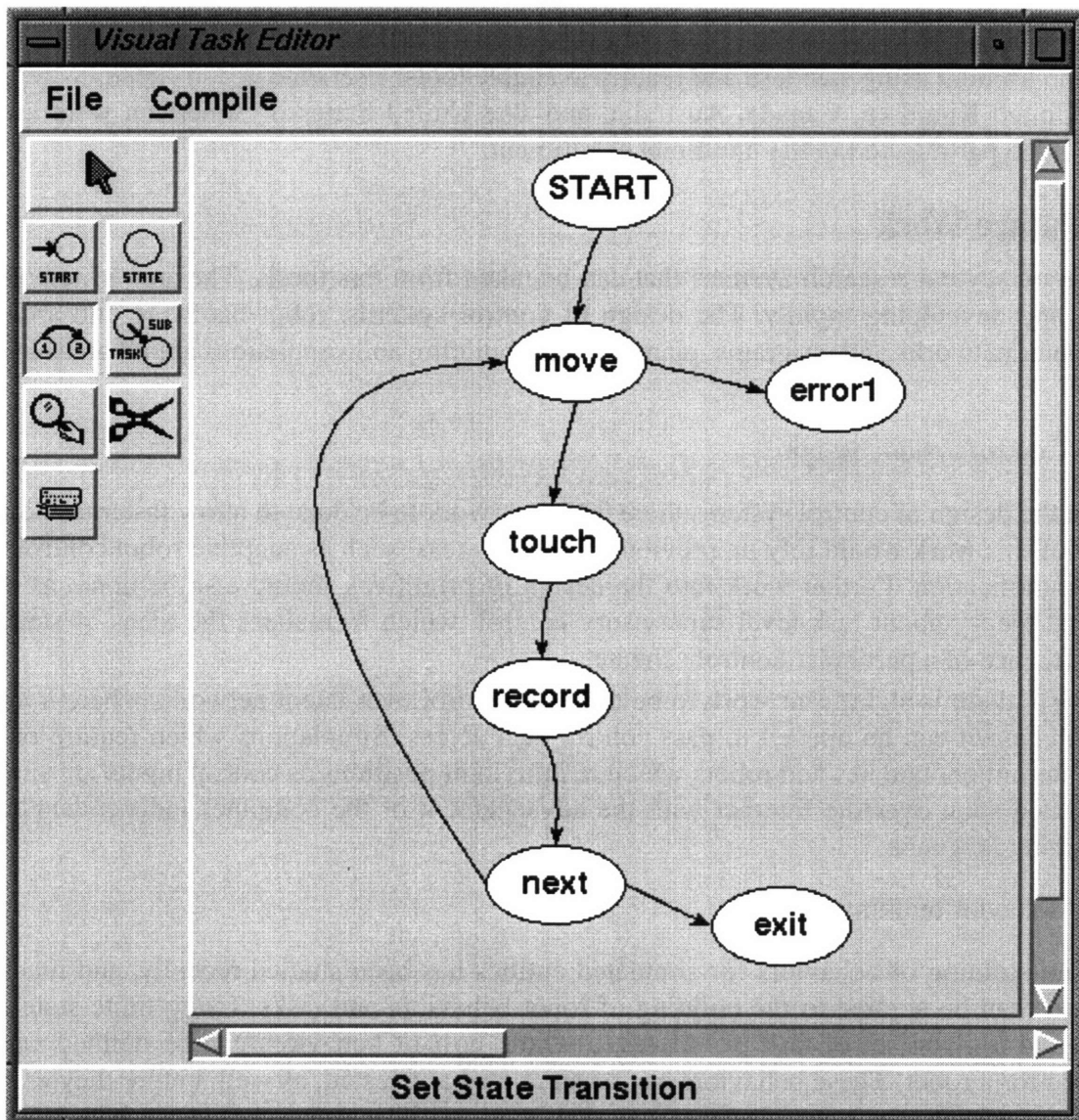


Figure 6.2: An early behavior building tool. This can be used to edit behaviors running on simulations in real time.

6.4.3 Networked interaction

The robot used in this thesis work was connected directly to a network and was capable of receiving data from any network source. There was nothing to prevent another robot from sending and receiving data to or from the robot. In this way, two or more robots could interact over a network, and networked groups of robots could all be connected and used through automated agents which request action from the robot, receive results, and based on those, request action from another robot. Also, two close proximity robots could interact more easily if they were able to interface over a standard network with common protocols. The building of these robot networks is a large and possibly fruitful and entertaining area of research.

6.4.4 Simulation

Tasks built using a behavior builder discussed in section 6.4.3 can be checked with simulations. This would allow extensive off-line debugging and more rapid task prototyping.

The technique of *predictive* simulation has grown and become popular in recent years in the supervisory control field (Mar 1985, Park 1991). There has also been current research (Narasimhan 1994, Brock 1993) to use this type of technique in conjunction with accurate models (or simulations) of the environment to develop artificially intelligent robots which use the tasks available to them to negotiate complex situations by simulating what the tasks do in the simulated environment. Figure 6.2 shows how a predictive simulation and control system might work. Task-level control is conducive to using predictive simulation because it restricts the actions which the robot can perform, and hence the environment (usually a semi-structured one) does not require a complete model, but rather one which models only the aspects the robot's task can affect. Also, each task can have its own environment model, each model including only what is necessary to allow the task to be simulated fully. We can also have adaptive simulations, which build on the environment as data is taken in. The predictive simulation is a specific case of knowledge based systems (Sheridan 1983, Hewitt 1989). High fidelity simulations have allowed hyper-accurate prediction and estimation to be combined with supervisory control systems with time delays to allow better control.

Another feature of graphical simulated display is the use of *mappings*. A graphical display need not present the actual system architecture as it appears in reality. In fact, it needs only to present what data is important. It can *map* the real data into some other form which is more convenient, more usable, more intuitive, or simply different for some other reason. For example, suppose a robot was tracking the motion of a beating heart with the precision that made the motion of the heart relative to the robot almost nil. Now, on a graphical display, this very small relative motion would be displayed, *not* the full heart motion. A surgeon could use this data to manipulate the robot's relative position to the heart, while seeing only very small motions. This would allow surgery on a beating heart and from a distance, since only graphical information is needed, and the surgeon's presence is not. A system to demonstrate this was developed by the authors.

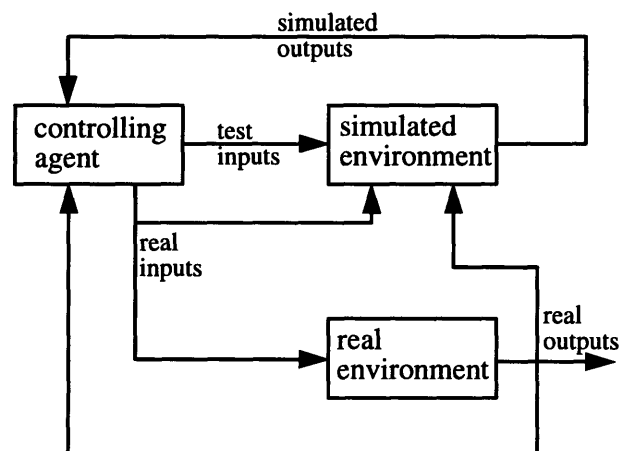


Figure 6.3: Predictive simulation and control system. Note that the environment simulation can be influenced by data returned from the real robot environment.

The controlling agent is most likely a human user but can be an automated system.

6.4.5 Applications

Telerobotic surgery is an exciting new application for this type of work. It has already been studied (Funda, et al 1993) somewhat with good results. This was for a high bandwidth system with minimal time delays.

Applications to shared resources over computer networks is another application. An example is the shared chemistry laboratory. If a person wished to conduct an experiment, he or she need not have an entire supply of equipment in house, but rather can “go out over the net” using a networked telerobotic system and execute the appropriate tasks to conduct the experiment. The robots performing the procedure would be controlled in a manner very similar to that discussed in this thesis.

Control of autonomous mobile agents can also benefit from task-level supervisory control. Networked groups of mobile robots could interact with each other based the perceived input from each other and use tasks to cooperate.

6.5 Conclusion

We have seen that a robot can successfully be controlled in the presence of long time delays and over a large distance. It can be made to perform useful tasks under these conditions, as well. Because of the destabilizing effects of long time delays on traditional feedback controllers, a task-level controller should be used with supervisory concepts included to allow the human user to use his or her intelligence in planning and task execution.

We took a real robot and system with a given time delay derived mostly from network latency, and built a task-level supervisory controller for it. The robot was given the ability to execute highly reliable primitive tasks fully independent from each other. The primitives were chosen based on the required task set the robot was to have. The primitives were part of a set which, when designed properly, allowed all of these primitives to be executed reliably. The set was also versatile enough to execute tasks for which the robot was not initially intended, and to design other tasks.

The controller, though successful in its goal, is sometimes prohibitively slow, but can be made automated to reduce the primitive completion times. Modular primitives can be easily built from lower level primitives available to a task builder. The primary time delay was from forward network latencies, so once this latency has passed, robot control is fast and simple, because the local control has very small time delays. This allowed a logic-based sequence of primitives to be sent over and executed locally, giving the robot automatic task execution capabilities.

The control system had a simple user interface which was made available to the general public. It performed reliably and robustly under varying user loads. This demonstrated the concept of remote, time-delayed, telerobotic primitive execution, and the concept of general remote manipulation was demonstrated.

We have showed that although this concept is viable and usable, there is still extensive study and more practical work that needs to be done. With the constant introduction of new and faster technologies, the development of systems which take advantage of these is required. Also, the heuristic nature some of this work demonstrates a general lack of formal theory available in the

supervisory control area, which allows more freedom in the design of controllers, but also fails to help analyze these controllers and build them with guaranteed properties.

Appendix A

Robot system hardware and control

A.1 Hardware

The robot is a four degree of freedom research robot shown graphically in Figure A.1. Only three of the available degrees of freedom are actively controlled, while the fourth is held fixed for simplicity of kinematic calculations. The robot is a low inertia, high torque design, which incorporates both gear and cable transmissions for speed reduction and torque amplification. There is some backlash in the gear trains which will limit positioning accuracy. The forward and inverse kinematics of the robot were determined using the Denavit-Hartenberg procedure (Craig 1985).

Power is transmitted to the robot via three pulse width modulating (PWM) servo amplifiers (made by Copley) which take a computer generated, zero-order-hold (ZOH) voltage signal as input and generate current at an appropriate duty cycle as output. This PWM current is fed directly to the brushed DC motors on board the robot. The amplifiers require minimum 2 amp DC at 25 volts. The DC used here was rectified from a transformer bringing down supply current from 110 volts AC. The servo amplifiers were able to regulate the supply current to pure DC.

The only sensory equipment on the robot are four shaft encoders on each motor (the encoder corresponding to the inactive motor is not measured). Quadrature counting (Auslander 1990) is used to measure 2000 counts per revolution (cpr) from a computer-based controller card. These counts are measured directly, and the measurements are taken on an interrupt basis by to ensure minimal loss of data.

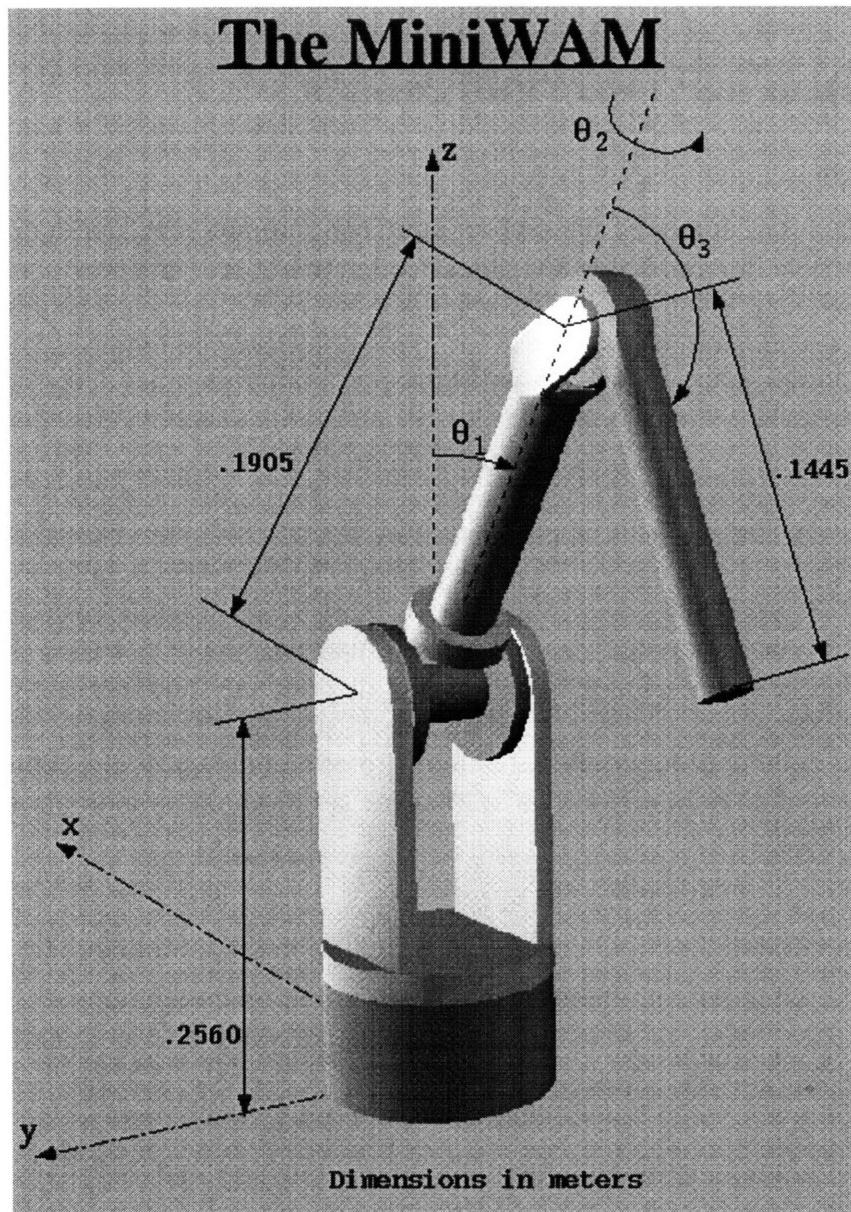


Figure A.1: Graphical diagram of the robot with joint angles and dimensions shown.

The “low-level control computer” in figures 4.1 and 4.3 is an IBM compatible PC, with an Intel Pentium processor running at 90 MHz clock speed. The computer has a controller card (MC Series) located at a directly accessible memory location. Software, supplied with the controller card (written in C), allows the appropriate memory addresses to be read and written. For motor control, the card interprets the values of these memory addresses and sends an appropriate analog voltage signal to the amplifiers. The card also writes to specific memory addresses with current encoder readings. The software is able to read these as desired. The computer also has a connection to an outside computer network with an Internet address. It is capable of using any of the current standard Internet communications protocols.

A.2 Network

Figures 4.1 and 4.3 also show the network connection to which the robot hardware and control software is attached. Communication over the network is done *concurrently* with the robot's controller via Berkeley sockets. This fast network is periodically polled for input from the network at approximately the servo rate. The input is interpreted and the software executes the appropriate program code. Any outputs are then immediately sent back over the network. Feedback via text and still images is provided to the user. There is also a graphically rendered feedback which can be used to insert virtual objects and motions into the environment using VRML (Bell 1995). This is one of the advantages of remote robot control: the real situation does not need to be depicted, only what is necessary and desired.

For this system, for those interested, UDP protocol was used because of its connection speed and ease of implementation. It is an unreliable protocol however, so care was taken to ensure that critical network data was retransmitted if it was lost (Stevens 1990)

Because the network was a standard ethernet type with direct access to the Internet, this experiment was broadened by allowing the robot to be controlled from *anywhere* on the Internet. The robot and it's low-level controller (the computer and hardware) were made available by the server so that any client program with the appropriate protocol, could connect and send commands to the robot. Internet browser programs, such as Netscape Navigator, use these protocols and can be used to send commands.

The user can execute local common gateway interface (CGI) programs (written in *perl*, see Wall 1990) which write commands to files (Savola 1995). These files are polled about once a second by the network interface computer, which has a direct socket connection to the control computer (see Figure 4.3). When a new command is detected in this file it is executed from this computer by sending it to the control computer.

Anyone familiar with this type of setup knows that network latencies are significant. This was the primary source of time delay in this system. Computational delays were secondary, and transmission times almost negligible. The latency forced us to use a task-level control system. However, we could not expect a typical user on the Internet to know any robotics or control engineering principles, so the task-level control system served the dual purpose of efficient network control and user-friendly interfacing, since the primitives were made available to the user as well.

A.3 Controller

Because of the low link inertia and accurate, low-noise position data available, and anticipation of minimal loading on the robot, it was decided to use a traditional proportional-integral-derivative (PID) controller on each link separately. Figure A.2 shows this feedback loop. The time domain control law is given by

$$u(t) = K_p e(t) + K_i \int_{\text{time}} e(t) dt + K_d \frac{d}{dt} e(t) + c \quad (\text{A.1})$$

$u(t)$ is the output signal (voltage), and $e(t)$ is the tracking error

$$e(t) = r(t) - x(t) \quad (\text{A.2})$$

where $r(t)$ is the desired joint angle and $x(t)$ is the actual angle. The gains K_p , K_i , and K_d , are the proportional, integral, and derivative gains, respectively. The constant, c , in eq (A.1) is to eliminate any offsets present in the servo amplifiers.

Equation (A.1) can be translated into the Laplace domain for ease of notation and familiarity

$$U(s) = K(s)E(s) + C \quad (A.3)$$

where $U(s)$ and $E(s)$ are the Laplace transforms of the signals $u(t)$ and $e(t)$, C is the Laplace transform of the constant c , and $K(s)$ is

$$K(s) = \frac{K_i + K_p s + K_d s^2}{s} \quad (A.4)$$

The robustness of PID controllers is well known and documented. See (Franklin, et al 1994) for a good treatment. Figure A.2 shows this loop in accordance with the block diagram paradigm with the controller block shown in its time-domain representation and split into its constituent parts (P, I, D).

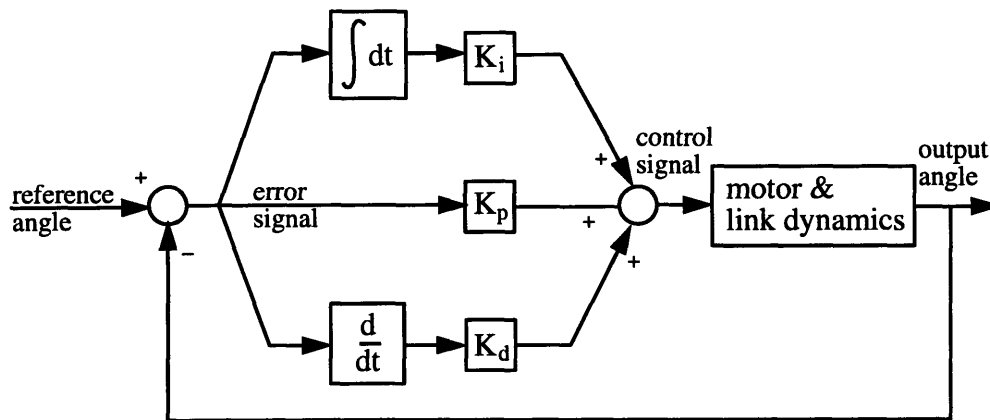


Figure A.2: Low-level controller block diagram. Each part of the PID controller is shown individually. The entire controller is most often wrapped into a single block, and its transfer function is displayed or represented.

Because this controller is to be implemented on a digital computer, eq (A.1) must be transformed into a discrete-time one. Although a proper treatment would require some z-plane analysis to ensure stability within a discrete sampled system, our sampling rate is greater than 1000 Hz, which gives a sampling period of

$$T < \frac{1}{(1000\text{Hz})} = 0.001 \text{ second} \quad (A.5)$$

With minimal knowledge of the system it is still clear that this is at least twenty times faster than the robot's open loop time constant, and hence the discrete time controller will be very close in performance to the continuous one. Only the gains must be tuned to account for the sampling rate.

Equation (A.1) can be rewritten to return a value of the control signal at each sampling instant, k , whose length is T (see Franklin, et al 1990):

$$u(kT) = K_p e(kT) + K_i \sum_{i=0}^k T e(kT) + K_d \left[\frac{e(kT) - e((k-1)T)}{T} \right] \quad (\text{A.6})$$

The approximation to the integral is the Forward Euler technique and the approximation to differentiation is the simple first order difference technique. With the sampling period implied, this equation can be written to give a programmer the value of the control signal during the k th sampling loop of the control system

$$u(k) = K_p e(kT) + K_i T e_i(k) + \frac{K_d}{T} [e(k) - e(k-1)] \quad (\text{A.7})$$

where

$$e_i(k) = e_i(k-1) + e(k) \quad (\text{A.8})$$

So the gains K_i and K_d must be modified by the sampling period, T , according to eq (A.7). This is a standard result in discrete-time control systems. As T approaches zero, implying faster and faster sampling, eq (A.6) approaches eq (A.1) and is identical in the limit. The implementation of this algorithm is straightforward (Auslander 1990).

No other low-level control was implemented. Some types of adaptive PID control could be used to account for varying loads (Narendra 1989), and others to account for the gear backlash, but these are considerably more complex and require significant computation and stability analysis. Although this is a good idea in any system, the purpose of this experiment was not low-level feedback controller design. Also, there was no explicit velocity control installed. The robot is only capable of position servoing. This is important when developing the algorithms for the primitives.

A.4 Supervisory Stability

To achieve stability, we tried to give the robot the ability to maintain the sufficient condition given in section 2.4. Therefore, there must be a default behavior which keeps the robot at its current position while waiting for the next request. We can't simply turn off the motors because gravity will pull the robot away from its current position. Instead, we had the robot actively servo about the last position it was at after a task was completed, unless it was in a sleep state, which is set by the "sleep" primitive. If we know that the servo control loop is asymptotically stable, then this implies the condition given in section 2.4. From Figure 4.4, we see that the loop is, in fact, asymptotically stable.

The default behavior of forcing the robot to servo about its current position while waiting for the next request is achieved by setting the reference angles to the current angles after a primitive has been completed. Then, while the robot is waiting, a PID servo loop is continually running in conjunction with the network checking loop. The two run nearly simultaneously. A diagram of this behavior is shown in Figure A.3.

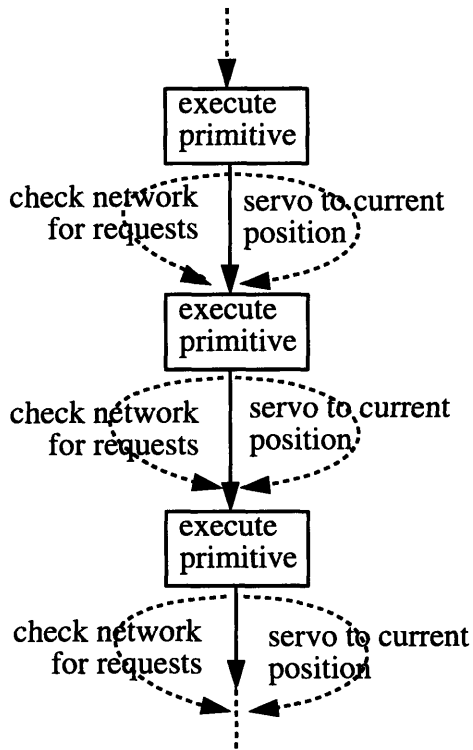


Figure A.3: Default behavior to achieve supervisory stability. The curved dotted lines show how the system goes through default behaviors *after* each primitive execution.

References

- [Auslander 1990] Auslander, D.M., Tham, C.H., *Real-Time Software for Control*, Prentice-Hall, New Jersey, 1990
- [Bajcsy 1984] Bajcsy, R. and Goldberg, K.Y., "Active Touch and Robot Perception," *Cognition and Brain Theory*, v2, Summer 1984
- [Bajcsy 1987] Bajcsy, R., and Constantinos, T., "Perception via Manipulation," *Proceedings of the International Society of Robotics Research*, 1987
- [Bicchi 1990] Bicchi, A., Salisbury, J.K., Brock, D.L., "Contact Sensing from Force Measurements," MIT Artificial Intelligence Laboratory, AI Memo no. 1262, Oct. 1990
- [Bell 1995] Bell, G., Parisi, A., Pesce, M., "The Virtual Reality Modeling Language: Version 1.0," (internet publication), May 1995
- [Brock 1993] Brock, D.L., "A Sensor Based Strategy for Automatic Robot Grasping," Ph.D. Thesis, ME, MIT, 1993
- [Doyle 1992] Doyle, J.C., Francis, B.A., Tannenbaum, A.R., *Feedback Control Theory*, Macmillan Publishing Co., New York, NY, 1992
- [Eberman 1990] Eberman, B.S., Salisbury, J.K., "Determination of Manipulator Contact Inferred from Joint Torque Measurements," in Hayward, V., and Khatib, O., (ed) *Lecture Notes in Control and Information Sciences*, Springer-Verlag, New York, NY, 1990
- [Eberman 1995] Eberman, B.S. "Contact Sensing: A Sequential Decision Approach to Sensing Manipulation Contact Features", Ph.D. Thesis, ME, MIT, 1995
- [Ferrell 1965] Ferrell, W.R., "Remote Manipulation with Transmission Delay," *IEEE Transactions on Human Factors in Electronics*, 1965
- [Franklin, et al 1990] Franklin, G.F., Powell, J.D., Workman, M.L., *Digital Control of Dynamic Systems*, 2e, Addison-Wesley, 1990
- [Franklin, et al 1994] Franklin, G.F., Powell, J, Emami-Naeini, A., *Feedback Control of Dynamic Systems*, 3e, Addison-Wesley, 1994
- [Funda 1993] Funda, J., Taylor, R., Gruben, K., LaRose, D., "Optimal Motion Control for Teleoperated Surgical Robots," in *Telem manipulator Technology and Space Telerobotics*, Won S. Kim, ed., Proc. SPIE 2057, p. 10 (1993)
- [Hewitt 1989] Hewitt, C., "Open Systems and Computer Science," in *Concepts and Characteristics of Knowledge-Based Systems*, Tokoro, M., Anzai, Y., Tonezawa, A. (ed.), Elsevier Science Publishers, B.V. (North Holland), 1989
- [Hirzinger 1993] Hirzinger, G., Landzettel, K., Heindl, J., "ROTEX - Space Telerobotic Flight Experiment," in *Telem manipulator Technology and Space Telerobotics*, Won S. Kim, ed., Proc SPIE 2057, p. 10 (1993)

- [Kernighan and Ritchie 1988] Kernighan, B.W., Ritchie, D.M., *The C Programming Language*, 2e, Prentice-Hall, 1988
- [Maes 1989] Maes, P., "How to Do The Right Thing," MIT Artificial Intelligence Laboratory, AI Memo no. 1180, Dec. 1989
- [Mar 1985] Mar, L.E., "Human Control Performance in Operation of a Time-delayed, Master-Slave Manipulator," B.S. thesis, ME, MIT, 1985
- [Marshall 1979] Marshall, J.E., *Control of Time-Delay Systems*, Institution of Electrical Engineers, Peter Pererniush Ltd. (pub.), New York, NY, 1979
- [Massimino 1993] Massimino, M.J., Campbell, P.D., Kearney, M.E., Meschler, M.F., "Manipulator Position Display (MPD) for Human Machine Interaction," in *Telem manipulator Technology and Space Telerobotics*, Won S. Kim, ed., Proc SPIE 2057, p. 10 (1993)
- [Narasimhan 1994] Narasimhan, S., "Task Level Strategies for Robots," Ph.D. thesis, EE, MIT 1994
- [Narendra 1989] Narendra, K.S., Annaswamy, A.M., *Stable Adaptive Systems*, Prentice-Hall, 1989
- [Noyes 1984] Noyes, M.V., Sheridan, T.B., "A Novel Predictor for Telem manipulation Through Time Delay," in *Proc. of the Annual Conference on Manual Control*, Moffett Field, CA, NASA Ames Research Center, 1984
- [Oguztöreli 1966] Oguztöreli, M.N., *Time Lag Control Systems*, Academic Press Inc., New York, NY, 1966
- [Park 1991] Park, J.H., "Supervisory Control of Robot Manipulator for Gross Motions," Ph.D. thesis, ME, MIT, 1991
- [Rosenberg 1993] Rosenberg, L.B., "Virtual Fixtures as Tools to Enhance Operator Performance in Telepresence Environments," in *Telem manipulator Technology and Space Telerobotics*, Won S. Kim, ed., Proc SPIE 2057, p. 10 (1993)
- [Salisbury 1988] Salisbury, J.K., Townsend, W.T., Eberman, B.S., DiPietro, D., "Preliminary Design of a Whole Arm Manipulation System (WAMS)," *Proc. 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, April 1988
- [Savola 1995] Savola, T., *Special Edition: Using HTML*, Que Corp. 1995
- [Shearer 1990] Shearer, J.L., Kulakowski, B.T., *Dynamic Modeling and Control of Engineering Systems*, Macmillan Publishing Co. New York, NY, 1990
- [Sheridan 1983] Sheridan, T.B., National Research Council, Committee on Human Factors, "Research Needs for Human Factors," National Academy Press, Washington D.C., 1983
- [Sheridan 1992] Sheridan, T.B., *Telerobotics, Automation, and Human Supervisory Control*, MIT Press, Cambridge, MA, 1992

- [Steels 1989] Steels, L., "Artificial Intelligence and Complex Dynamics," in *Concepts and Characteristics of Knowledge-Based Systems*, Tokoro, M., Anzai, Y., Tonezawa, A. (ed.), Elsevier Science Publishers, B.V. (North Holland), 1989
- [Stevens 1990] Stevens, W.R., *Unix Network Programming*, Prentice-Hall, 1990
- [Wall 1990] Wall, L., Schwartz, R.L., *Programming perl*, O'Reilly & Associates, O'Reilly, T. (ed.), 1990
- [Watanabe 1993] Watanabe, I., Aoki, T., Maruyama, T., Uchiyama, T., "Interactive Task Planning System for Space Robots," in *Telem manipulator Technology and Space Telerobotics*, Won S. Kim, ed., Proc SPIE 2057, p. 10 (1993)
- [Wolovich 1987] Wolovich, W.A., *Robotics: Basic Analysis and Design*, CBS College Publishing, New York, NY, 1987
- [Yoerger, D. 1991] "The Application of Supervisory Control to Underwater Telerobotics," in *Robotics, Control and Society*, Moray, N., Ferrell, W.R., Rouse, W.D., ed., Ch. 7.
- [Ziegler Nichols 1942] Ziegler, J.G., Nichols, N.B., "Optimum Settings for Automatic Controllers," *Transactions of ASME*, v. 64, pp 759-768, 1942
- [Ziegler Nichols 1943] Ziegler, J.G., Nichols, N.B., "Process Lags in Automatic Control Circuits," *Trans., ASME*, vol. 65, no. 5, pp.433-444, July 1943