Minimum Energy Trajectory Planning for Robotic Manipulators

by

Glen Arthur Field
B.A.Sc., University of Waterloo, 1986
M.A.Sc., University of Waterloo, 1988

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the
Department of Mechanical Engineering

We accept this dissertation as conforming
to the required standard.

Dr. Y. A. Stepanenko, Supervisor (Dept. of Mechanical Engineering)

Dr. B. Tabarrok, Departmental Member (Dept. of Mechanical Engineering)

Dr. R. Podhorodeski, Departmental Member (Dept. of Mechanical Engineering)

Dr. E. Manning, Outside Member (Dept. of Electrical Engineering)

Dr. Y. Altintas, External Examiner (Dept. of Mechanical Engineering, University of British Columbia)

Supervisor: Dr. Yury A. Stepanenko

Abstract

This dissertation develops a practical approach to the planning of minimum energy consumption trajectories for robotic manipulators. That is, a trajectory is sought between given initial and final states such that the energy consumption is minimized.

This functional optimization problem may be described using classical variational theory. However, the two-point-boundary-value problem which arises from this formulation is practically insoluble. Therefore, the problem is converted to a function optimization by searching for an optimum among a class of functions which can be described by a fixed number of parameters; namely spline functions. Conversion to a function optimization problem allows a large number of numerical function optimization methods to be attempted. Methods attempted in this work are Powell's method, interval methods, genetic algorithms, simulated annealing and dynamic programming. Each of these methods are found to be wanting in some respects. Powell's method is a local optimization procedure. As such, it tends to get trapped in local minima. Interval methods, while interesting in that they provide the promise of a global optimum, fail, since a suitable interval bounding function for the energy consumption is not available. Genetic algorithms, simulated annealing and dynamic programming all require excessive computational effort.

To overcome these difficulties, the basic dynamic programming method is modified to perform a series of dynamic programming passes over a small reconfigurable grid covering only a portion of the solution space at any one pass, rather than attempt to perform a single dynamic programming operation over a large tightly spaced grid. Although this modification changes the dynamic programming approach from a global to a local

optimization process, it greatly reduces the number of function evaluations required. It is important to note that the developed modified dynamic programming approach is not local in the sense of Powell's method. Since it still searches over regions of the solution space, it retains the ability to avoid some poor local minima. This ability is demonstrated.

It is discovered that the form of spline representation used to represent the manipulator trajectories has a major effect on the performance of the modified dynamic programming method. Both time domain and phase space representations are proposed. While more difficult to start and handle programmatically, the phase space representation proved a more natural trajectory representation for planning minimum energy consumption trajectories. Better results were achieved with as few as two percent of the function evaluations used by the time domain representation.

The modified dynamic programming approach is verified experimentally by planning and executing a minimum energy consumption path for a Reis V15 industrial manipulator. To plan the path a dynamic model for the Reis V15 manipulator is developed. This model includes actuator and controller dynamics.

The modified dynamic programming approach using a splined phase space trajectory representation is shown to be an effective tool for the computation of minimum energy trajectories for real manipulators. The algorithm has an inherent parallel structure, allowing for reduced computation time on parallel architecture computers. No limiting assumptions are made about the performance index or function to be optimized. As such, extremely complex functions and constraints are easily handled. Joint actuator and time constraints are considered in this work. Other possible constraints for future work include reaction forces and obstacles.

Examiners:

Dr. Y. A. Stepanenko, Supervisor (Dept. of Mechanical Engineering)

Dr. B. Tabarrok, Departmental Member (Dept. of Mechanical Engineering)

Dr. R. Podhorodeski, Departmental Member (Dept. of Mechanical Engineering)

Dr. E. Manning, Outside Member (Dept. of Electrical Engineering)

Dr. Y. Altintas, External Examiner (Department of Mechanical Engineering, University of British Columbia)

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my supervisor, Dr. Yury Stepanenko, for his guidance in the development of this work. Thanks to the National Science and Engineering Research Council of Canada, the Institute for Robotics and Intelligent Systems (IRIS) and Precarn Associates Inc., the Canadian Space Agency and International Submarine Engineering for their financial support. Thanks also to: Al Vermeulen for his assistance with B splines; Dave Gawley for his programming tips, computer system knowledge and work on the TEST user interface; Ken Pittens for his work on the Reis V15 actuator identification; Qing Zhang for his design of the A/D conversion hardware for the force sensor; Janko Pesic for his work on current and tachometer measurement connections; Graham Wheeler for his assistance with Cadkey; and Assurance Technologies Inc. for providing the force sensor strain gauge calibration matrix. Finally thanks to my parents Wesley and Marion for their support and encouragement.

Chapter 1
Introduction

In this work, the problem of determining minimum energy trajectories for robotic manipulators is studied. Minimum energy is one of several performance criteria which may be studied under the scope of the optimal path planning problem. Other common performance criteria include motion time, values of reaction forces, and control effort. The minimum energy performance criterion has some distinct advantages over other criteria for optimum path planning of robotic manipulators.

Minimizing motion time can lead to numerical problems as the solution is often degenerate. Consider, for example, a multiple link arm. The minimum time to complete a specified move depends only on the slowest link. The number of trajectories possible for the remaining links is infinite; an undesirable characteristic for numerical computation. This does not mean that the minimum time problem should be neglected since the throughput capabilities of a robot may be enhanced by the solution of this problem. Fortunately the minimum time problem may be treated as a limiting case of the minimum energy problem by appending a weighted time penalty parameter to a minimum energy cost function. As the weight on the time penalty parameter is increased the optimum tra-

jectory approaches one of minimum time. Also, in some cases, the motion time may be determined by external factors. In such a situation, energy consumption can be minimized subject to a motion time constraint.

The idea behind minimizing reaction forces is to keep the forces/torques within the mechanism and/or between the manipulator and environment small. In many cases however, reaction forces and torques may be expressed as limiting values (not necessarily constant) and may be included as constraints in a minimum energy problem.

Minimizing control effort is usually expressed as the sum of squares of the control variables. For robotic manipulators, and many other control systems, the control variables are of very low power in comparison to the system being controlled. As a result there is little physical benefit to minimizing the control effort.

Minimum energy cost criteria do not suffer from the degeneracy of the minimum time criterion because of the coupling between joint variables such as angular velocity and torque, and energy consumption. In addition, as mentioned above, other performance criteria may be added to minimum energy criteria either through weights or as constraints on the system. The question now arises: Why study the problem of determining minimum energy trajectories for robotic manipulators?

It may be effectively argued that on the factory floor minimizing the energy consumption of robotic manipulators is unimportant for any one of several reasons:

- The power consumption of manipulators is small in comparison to other machinery in the plant.
- A virtually unlimited power supply is conveniently available through the existing electrical grid.
- High throughput is of greater importance in an industrial setting.

Although the factory floor may not be a suitable place for the use of minimum energy trajectory planning, it is not the only place where robots are making an impact. Currently a more dynamic area of robotic development is that of autonomous systems for operation in remote or hazardous environments. Consider that robots used in manufacturing environments and the tasks they perform have not changed significantly in the last twenty years. Traditional tasks such as spot welding, machine tending, and spray painting have long been in the realm of technological possibility.

In contrast to robotics on the factory floor, a rapidly growing area of robotics research is that of autonomous or semi-autonomous machines. These machines are required to operate in remote environments with limited supervision. Forces driving this research include space exploration and the desire to exploit the yet untapped resources of the world's oceans. Remote manipulators may be supplied with power through some kind of tether arrangement, carry a supply of energy with them (batteries), or generate their own power (solar). In many remote environments the use of a tether is too cumbersome. For example, the mass of a long cable for an autonomous deep sea vehicle would severely restrict freedom of movement. Therefore if the manipulator is to carry its own power supply or generate its own power, motions should be planned to make the most effective use of these limited energy resources.

This work extends the knowledge in the field of optimal path planning for robotic manipulators by providing practical methods for computing minimum energy consumption trajectories between two given states taking into full account manipulator dynamics, dissipative characteristics of the actuators and realistic actuator and world constraints. Further, the developed methods are experimentally verified by demonstrating their ability to compute minimum energy trajectories for a real robot arm.

Chapter 2
Theory

2.1 Statement of the General Problem

The objective of this chapter is to state the general form of the class of problems under which optimal trajectory planning falls and to present a historical development of the theory relating to this problem. Toward that end, it is necessary to consider a mapping from a space S to the real line R^1. This is shown in Figure 2.1. The operator J, which performs this mapping is called a functional. J assigns a real value $J(f)$ to every $f \in S$. The operator J allows the elements of the space S to be ordered relative to one another. The functional J is therefore just a device for ranking the elements of the space S. With this definition, the basic *local optimization* problem may be stated as follows.

> Find $f^* \in S$ such that $J(f^*) \leq J(f)$ for all $f \in S$ in the neighborhood of f^*.

Before continuing with the development of the theory necessary to solve problems of the above type, the term neighborhood must first be defined. The neighborhood of f^* may be defined as all points with distance $\delta \leq \varepsilon$ from f^*, where $\varepsilon > 0$. In Euclidean space

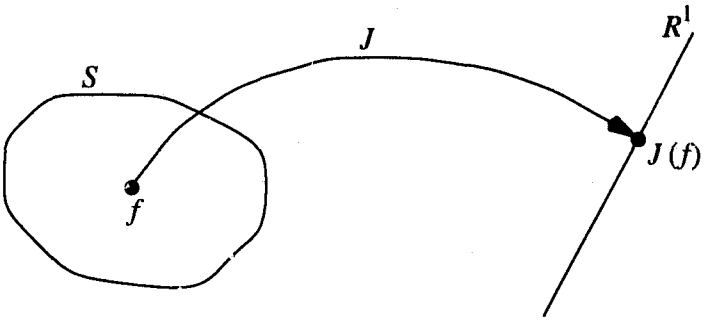


Figure 2.1 A Functional Mapping From Space S to the Real Line R^1.

the meaning of distance may be clearly defined with a Euclidean norm. If the space of concern is a function space however the meaning of distance is not as clear. In fact, several types of "distances" are defined and these definitions set the types of extremum which may be defined. The greatest of,

$$max_x|f(x)\,(-f^*(x)|\,)$$

$$max_x|f'(x)-f^{*\prime}(x)|$$

$$\cdots$$

$$max_x|f^n(x)-f^{*n}(x)| \qquad (2.1)$$

is called the n^{th} order distance of the two functions $f(x)$ and $f^*(x)$. Therefore, a small zeroth order distance implies that the two functions are near each other. A small first order distance means that both the functions and their first derivatives are near each other. With the above definitions of "distance" the extrema may be classified. If the value of the functional on the given curve is less than on all other curves to which the zeroth order distance is small, the extremum is called strong. If the value of the functional on the given curve is

less than on all curves to which the first order distance is small, the extremum is called weak. Note that any strong extremum is a weak extremum but the converse is not true

A more general and difficult class of problem is the *global optimization* problem, which may be stated as follows.

Find $f^* \in S$ such that $J(f^*) \leq J(f)$ for all $f \in S$.

Approaches to solving this problem will also be discussed.

2.2 Historical Survey

The development of the methods of differential calculus by Newton and Leibniz near the end of the seventeenth century provided the theoretical basis from which methods of solution to the local optimization problem could be developed. These newly developed methods could be employed immediately to solve many practical extremization problems. Problems where the extremum depended on the selection of a function as a whole rather than the value of a few independent variables required further development.

Jean Bernoulli was the first to pose one of these new types of extremum problems when he published the brachistochrone problem in 1696. The problem statement reads as follows.

> Among all lines connecting two given points, find the curve traversed in the shortest time by a material body under the effect of gravity.

The solution to this problem was first provided by Leibniz and a year later, after a second publication of the problem, solutions were given by Jacob Bernoulli, L'Hospital and Newton. In subsequent years several mathematicians worked on the solution of particular examples of these variational problems. A landmark in the theory came in 1732 when Euler published a general method of solving a class of variational problems.

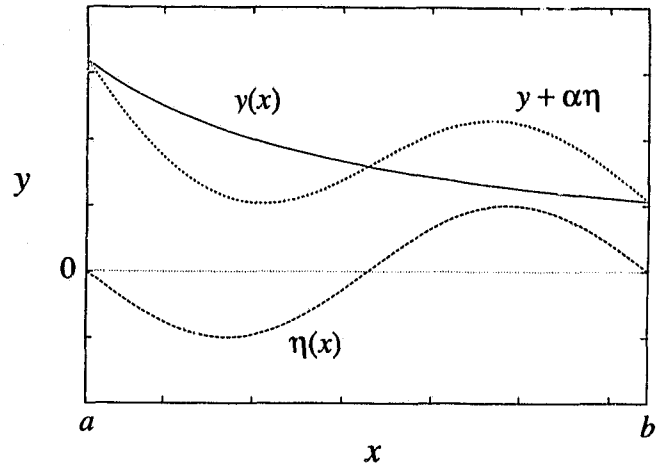


Figure 2.2 Variation of a Function

2.2.1 Euler's Equation

Euler sought to determine the conditions that a function $y(x)$ must satisfy in order to provide a weak relative extremum of the functional

$$J = \int_a^b F(x, y, y')dx \tag{2.2}$$

where the function F is assumed to be continuous and to have continuous partial derivatives to second order inclusive. Let the extremum be achieved on $y(x)$. Consider the function

$$y + \alpha\eta(x) \tag{2.3}$$

where α is a scalar and η is an arbitrary smooth function subject only to the conditions

$$\eta(a) = \eta(b) = 0 \tag{2.4}$$

See Figure 2.2. The increment of the functional J is given by,

$$\Delta J = \int_a^b F(x, y + \alpha\eta, y' + \alpha\eta')dx - \int_a^b F(x, y, y')dx \tag{2.5}$$

Performing a Taylor series expansion yields

$$\Delta J = \alpha \left(\frac{dJ}{d\alpha}\right)_{\alpha = 0} + \frac{\alpha^2}{2!}\left(\frac{d^2J}{d\alpha^2}\right)_{\alpha = 0} + \ldots \tag{2.6}$$

The first term in the Taylor series expansion is called the first variation of the functional. The second term is the second variation, and so on. For small α, the increment δJ can be approximated by the first variation. Since, by assumption, the extremum of the functional is achieved on $y(x)$, the increment may not be negative.

$$\Delta J \approx \delta J = \alpha \left(\frac{dJ}{d\alpha}\right)_{\alpha = 0} \geq 0 \tag{2.7}$$

Since α is arbitrary, a necessary condition for an extremum is

$$\left(\frac{dJ}{d\alpha}\right)_{\alpha = 0} = 0 \tag{2.8}$$

Then

$$\frac{dJ}{d\alpha} = \int_a^b \frac{d}{d\alpha} F(x, y + \alpha\eta, y' + \alpha\eta')dx \tag{2.9}$$

$$\frac{dJ}{d\alpha} = \int_a^b \left[\frac{\partial F}{\partial y}\frac{d}{d\alpha}(y + \alpha\eta) + \frac{\partial F}{\partial y'}\frac{d}{d\alpha}(y' + \alpha\eta')\right]dx \tag{2.10}$$

$$\frac{dJ}{d\alpha} = \int_a^b \left(\frac{\partial F}{\partial y}\eta + \frac{\partial F}{\partial y'}\eta'\right) dx \tag{2.11}$$

Integrating the second term by parts gives,

$$\int_a^b F_{y'}\eta' dx = F_{y'}\eta\Big|_a^b - \int_a^b \eta \frac{d}{dx}F_{y'} dx \tag{2.12}$$

$\eta(x)$ vanishes at points a and b. Therefore,

$$\int_a^b F_{y'}\eta' dx = -\int_a^b \eta \frac{d}{dx}F_{y'} dx \tag{2.13}$$

$$\delta J = \int_a^b \left(F_y - \frac{d}{dx}F_{y'}\right) \alpha\eta dx \tag{2.14}$$

It is clear that if $y(x)$ provides an extremum of the functional it is necessary that

$$F_y - \frac{d}{dx}F_{y'} = 0 \tag{2.15}$$

This second order differential equation is known as Euler's equation and solutions to the equation are called extremals. Euler was able to generalize the functional of equation 2.2 to include higher order derivatives of the function $y(x)$. This functional and the resulting Euler equations are as follows.

$$J = \int_a^b F(x, y, y', y'', \ldots, y^{(n)}) dx \tag{2.16}$$

$$F_y - \frac{d}{dx}F_{y'} + \frac{d^2}{dx^2}F_{y''} - \ldots = 0 \tag{2.17}$$

It is important to remember that Euler's equation is only a necessary condition for an extremum. Extremals may yield a local maximum, a local minimum, or a saddle point condition. If Euler's equation is not satisfied by any function in the space being considered then an extremum does not exist for the considered function space.

2.2.2 Contributions of Lagrange

J. L. Lagrange created the *"method of variations"* for the solution of variational problems. His method was first described in a letter to Euler in 1755 and was published in 1760 and 1761. The published work included the solution to variational problems with moving endpoints.

The method of variations is based on the first variation of the functional J. Consider the simplest functional form

$$J = \int_a^b F(x, y, y')dx \tag{2.18}$$

Assume that the extremal is $y(x)$ of Figure 2.3 and that $y(x) + h(x)$ is its variation.

The increment of the functional is

$$\Delta J = J(y+h) - J(y)$$

$$= \int_{x_0+\delta x_0}^{x_1+\delta x_1} F(x, y+h, y'+h')dx - \int_{x_0}^{x_1} F(x, y, y')dx \tag{2.19}$$

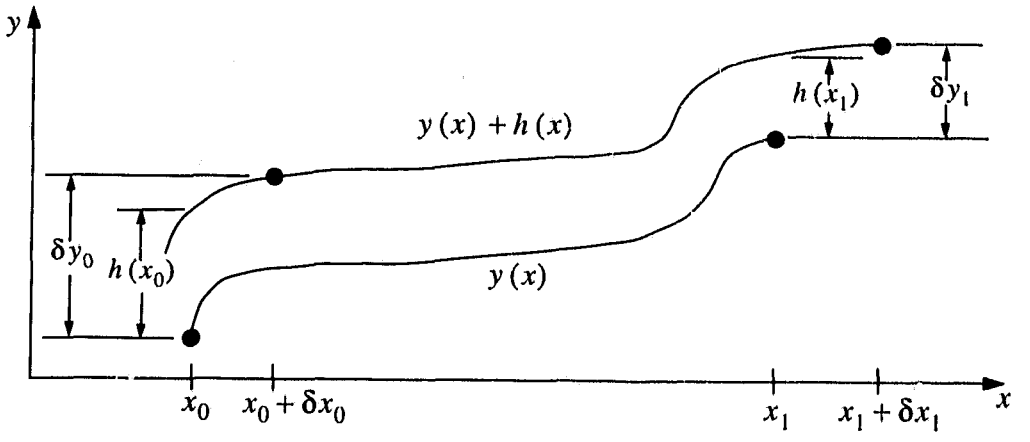


Figure 2.3 Variation of a Function with Variable Endpoints

$$\Delta J = \int_{x_0}^{x_1} [F(x, y+h, y'+h') - F(x, y, y')]\, dx +$$

$$\int_{x_1}^{x_1+\delta x_1} F(x, y+h, y'+h')dx - \int_{x_0}^{x_0+\delta x_0} F(x, y+h, y'+h')dx \tag{2.20}$$

The first variation of the functional is given by the linear portion of the increment.

$$\delta J = \int_{x_0}^{x_1} [F_y h + F_{y'} h']\, dx + F|_{x=x_1}\delta x_1 - F|_{x=x_0}\delta x_0 \tag{2.21}$$

Integrating the second term of the integrand by parts and substituting the following relations apparent from Figure 2.3

$$h(x_0) = \delta y_0 - y'\delta x_0 \tag{2.22}$$

$$h(x_1) = \delta y_1 - y'\delta x_1 \tag{2.23}$$

gives

$$\delta J = \int_{x_0}^{x_1} (F_y - \frac{d}{dx}F_{y'})\, h dx + F_{y'}\big|_{x=x_1} \delta y_1 - F_{y'}\big|_{x=x_0} \delta y_0 +$$

$$(F - y'F_{y'})\big|_{x=x_1} \delta x_1 + (F - y'F_{y'})\big|_{x=x_0} \delta x_0 \tag{2.24}$$

As before the necessary condition for an extremum is that

$$\delta J = 0 \tag{2.25}$$

Note that the integral term of equation 2.24 is the same result achieved by Euler for fixed endpoint conditions. The remaining terms are necessary for the inclusion of variable endpoints. Assume that the endpoints are constrained to move along two curves $\varphi(x)$ and $\psi(x)$, and let function $y(x)$ be the function which extremizes the functional equation 2.18. For extremal $y(x)$ the integral term in equation 2.24 vanishes. Using a linear approximation at the endpoints one may write,

$$\delta y_0 = \varphi' \delta x_0 \qquad \delta y_1 = \psi' \delta x_1 \tag{2.26}$$

Therefore

$$\delta J = (F_{y'}\psi' + F - y'F_{y'})\big|_{x_1} \delta x_1 - (F_{y'}\varphi' + F - y'F_{y'})\big|_{x_0} \delta x_0 \tag{2.27}$$

Since δx_0 and δx_1 are independent,

$$(\varphi' F_{y'} + F - y'F_{y'}\big|_{x=x_0}) = 0 \tag{2.28}$$

$$(\psi' F_{y'} + F - y'F_{y'}\big|_{x=x_1}) = 0 \tag{2.29}$$

Equations 2.28 and 2.29 are called transversality conditions.

The culmination of Lagrange's work is the solution to what is now referred to as the general Lagrange problem. This is an important result since many other problems can be reduced to a problem of this form. The general Lagrange problem is to find the vector function $\hat{y}(x)$ which extremizes the functional

$$J = \int_a^b F(x, \hat{y}, \hat{y}')dx \tag{2.30}$$

Subject to the following conditions.

$$\varphi_j(x, \hat{y}, \hat{y}') = 0 \qquad j=1,2,\ldots n \tag{2.31}$$

Note that the functional, equation 2.30, depends on several functions $\hat{y}(x)$. In this case the extremum must satisfy the system of Euler equations

$$\frac{\partial F}{\partial y_i} - \frac{d}{dx}\frac{\partial F}{\partial y'_i} = 0 \qquad i=1,2,\ldots m \tag{2.32}$$

where m is the dimension of the vector $\hat{y}(x)$. Lagrange was able to include the conditional equations 2.31 in his extremization by introducing a new function,

$$H = F + \sum_{j=1}^{n} \lambda_j(x)\varphi_j(x, \hat{y}, \hat{y}') \tag{2.33}$$

where $\lambda_j(x)$ are as yet unknown functions, and seeking the extremal of the functional

$$J^* = \int_a^b H dx \tag{2.34}$$

by customary methods. For this problem $m+n$ functions must be determined. The required $m+n$ equations are the m Euler equations 2.32 and the n coupling equations 2.31.

Two types of problems which may be reduced to the general Lagrange problem are problems with functionals dependent on higher derivatives and isoperimetric problems. Consider the functional

$$J = \int_a^b F(x, y, y', y'')dx \tag{2.35}$$

The problem of finding the extremal of this functional may be reduced to the Lagrange problem by making the substitutions $y' = z$ and $y'' = z'$. The problem now is to find the extremal of the functional

$$J = \int_a^b F(x, y, y', z')dx \tag{2.36}$$

under the condition

$$y' - z = 0 \tag{2.37}$$

Isoperimetric problems have integral constraints. To reduce problems of this type to the general Lagrange problem the integral constraints are transformed to differential equations. Consider the following integral constraint. The integral

$$J_1 = \int_a^b K(x, y, y')dx \tag{2.38}$$

is required to attain the value J_{1_0} on the extremum of a given functional

$$J = \int_a^b F(x, y, y')dx \tag{2.39}$$

Denoting

$$\varphi(x) = \int_a^b K(x, y, y')dx \tag{2.40}$$

the intermediate function

$$H = F + \lambda(x)(\varphi' - K) \tag{2.41}$$

can be generated. The problem is now to extremize functional, equation 2.34, subject to the coupling equation

$$\varphi' - K(x, y, y') = 0 \tag{2.42}$$

The methods of Euler and Lagrange employ only the first variation of the functional and therefore are unable to categorize extremals as maxima, minima, or saddle conditions. Also the conditions which come from these methods are necessary but not sufficient conditions for an extremum.

2.2.3 Legendre Conditions

Legendre derived the criteria for differentiating between a minimum and a maximum in 1786 by studying the second variation of the functional

$$J = \int_a^b F(x, y, y')dx \tag{2.43}$$

Legendre noted that since the first variation vanished at an extremum the sign of the increment would be given by the second variation

$$\Delta J = \delta J + \delta^2 J + \ldots \tag{2.44}$$

where

$$\delta^2 J \equiv \frac{\alpha^2}{2}\left(\frac{d^2 J}{d\alpha^2}\right) \tag{2.45}$$

Therefore $y(x)$ yields a minimum of the functional if $\delta^2 J > 0$ and a maximum if $\delta^2 J < 0$. Let the variation of the extremum $y(x)$ be given as before by equation 2.3. Now

$$\begin{aligned}\frac{d^2 J}{d\alpha^2} &= \int_a^b \frac{d^2}{d\alpha^2} F(x, y + \alpha\eta, y' + \alpha\eta')dx \\ &= \int_a^b (F_{yy}\eta^2 + 2F_{yy'}\eta\eta' + F_{y'y'}\eta'^2)\, dx\end{aligned} \tag{2.46}$$

Integrating the second term of the integrand by parts and noting that $\eta(a) = \eta(b) = 0$ gives

$$\frac{d^2 J}{d\alpha^2} = 2\int_a^b (P\eta^2 + R\eta'^2)\, dx \tag{2.47}$$

where

$$P = \frac{1}{2}\left(F_{yy} - \frac{dF_{yy'}}{dx}\right) \tag{2.48}$$

and

$$R = \frac{1}{2}F_{y'y'} \tag{2.49}$$

Since $\eta(x)$ is an arbitrary function it is necessary that

$$F_{y'y'} \geq 0 \tag{2.50}$$

for a minimum, and

$$F_{y'y'} \leq 0 \tag{2.51}$$

for a maximum. The above conditions are only necessary conditions. The Legendre conditions permit distinction between a maximum and a minimum only if they exist and are reached on an extremal, but do not provide sufficient conditions for the existence of the extremum.

2.2.4 Field Theory and Sufficient Conditions

Field Theory

The derivation of sufficient conditions for an extremum requires the investigation of families of extremals. The study of families of extremals is part of field theory. Recall the simplest functional

$$J = \int_a^b F(x, y, y')dx \tag{2.52}$$

The solutions to the Euler equation for this functional generate a family of curves

$$y = y(x, C_1, C_2) \tag{2.53}$$

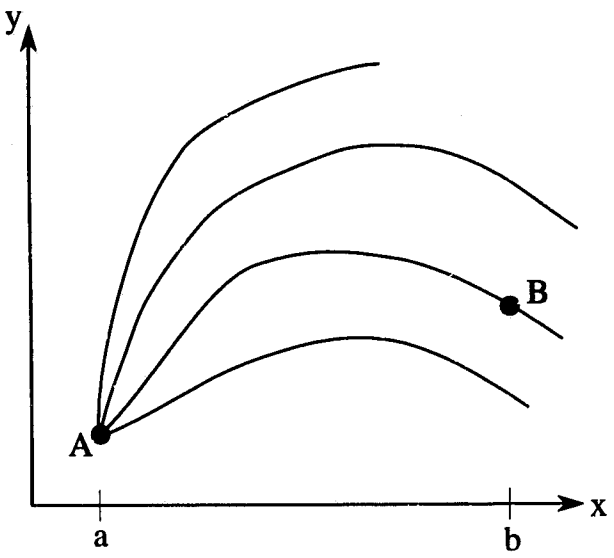


Figure 2.4 A Pencil of Extremals

which depends on the integration constants C_1 and C_2. These constants are determined by the conditions that the extremal passes through known points A and B. If only one constant is fixed, say C_1 from the condition of the extremal passing through point A, a pencil of extremals passing through the point A will result. See Figure 2.4. The pencil of extremals is called a field on the interval (a, b). A field is defined as follows. If a family of curves dependent on one parameter disposed in some domain D in such a way that one and only one curve will pass through each point of the domain, then this family of curves generates a field in the domain D.

Jacobi Condition

Now consider the case of a weak extremum where both zero and first order distances are small. Recall that the sign of the increment of the functional agrees with the sign of the second variation when going from the extremal to a nearby curve. The Legendre condition says that

$$F_{y'y'} \leq 0 \tag{2.54}$$

is necessary to make the second variation nonpositive. Legendre attempted to show that compliance with the strengthened condition

$$F_{y'y'} < 0 \tag{2.55}$$

is sufficient for the second variation to be negative. Legendre reasoned as follows. From equation 2.47 the second variation is given by

$$\delta^2 J = \alpha^2 \int_a^b (P\eta^2 + R\eta'^2)\, dx \tag{2.56}$$

Then for any differentiable function $\omega(x)$,

$$\int_a^b (\eta^2\omega' + 2\eta\eta'\omega)\, dx = \int_a^b \frac{d}{dx}(\eta^2\omega)\, dx = 0 \tag{2.57}$$

Attaching equation 2.57 to the expression for the second variation gives

$$\delta^2 J = \alpha^2 \int_a^b [R\eta'^2 + 2\eta\eta'\omega + (P + \omega')\eta^2]\, dx \tag{2.58}$$

Choose the function $\omega(x)$ so that the integrand forms a perfect square. To form a perfect square $\omega(x)$ must satisfy the differential equation

$$R(P + \omega') = \omega^2 \tag{2.59}$$

Substituting for $(P + \omega')$ in equation 2.58 gives

$$\delta^2 J = \alpha^2 \int_a^b R\left[\eta' + \frac{\eta\omega}{R}\right]^2 dx \tag{2.60}$$

It is apparent that the second variation will agree with the sign of R and hence the sign of $F_{y'y'}$. The flaw in this argument, as Lagrange pointed out, is that equation 2.59 may not even have a solution. Making the change of variable

$$\omega = -\left(\frac{u'}{u}\right) R \tag{2.61}$$

where u is a new unknown function, equation 2.59 is transformed into the second order linear differential equation

$$-\frac{d}{dx}(Ru') + Pu = 0 \tag{2.62}$$

If $u(x)$, the solution to equation equation 2.62, does not vanish on (a, b), then a solution of equation 2.59 also exists. The condition that the solution to equation 2.62 does not vanish on (a, b) is called the Jacobi condition. Together, the Jacobi condition and the strengthened Legendre condition ($F_{y'y'} > 0$ for a minimum and $F_{y'y'} < 0$ for a maximum) are sufficient for a weak relative extremal of the functional on (a, b).

To see how this result relates to a field theory, consider the following derivation of the Jacobi equation. Let $h(x)$ represent the difference in ordinate of two infinitesimally close extremals $y(x)$ and $y(x) + h(x)$. Since $y(x) + h(x)$ is an extremal it must satisfy the Euler equation. Therefore

$$F_y(x, y + h, y' + h') - \frac{d}{dx} F_{y'}(x, y + h, y' + h') = 0 \tag{2.63}$$

Performing a Taylor series expansion and keeping only first order terms in h gives

$$F_{yy}h - F_{yy'}h - \frac{d}{dx}(F_{y'y'}h' + F_{y'y}h) = 0 \tag{2.64}$$

which is equivalent to

$$Ph - \frac{d}{dx}(Rh') = 0 \tag{2.65}$$

Equation 2.65 is of course the Jacobi equation (see equation 2.62) where the unknown function $u(x)$ is given by $h(x)$, the difference in ordinate between two infinitesimally near extremals. The solution to equation 2.65 vanishes at an intersection of neighboring extremals. Therefore, for the Jacobi condition to be satisfied the extremals must form a field on the interval (a, b). From the above development it is apparent that there are two means to test the Jacobi condition. One method is to analytically find the solution to the Jacobi equation. The other is to construct the field of extremals. If the desired extremal passing through points A and B does not intersect infinitesimally nearby extremals then the Jacobi condition is satisfied.

Weierstrass Condition

To determine the condition for a strong extremum again consider the increment of the functional

$$\Delta J = \int_H F(x, y, y')dx - \int_{extr} F(x, y, y')dx \tag{2.66}$$

where the integral term subscripted by $extr$ is the functional on the extremal passing through points A and B in Figure 2.5 and the term subscripted by H is the function on an

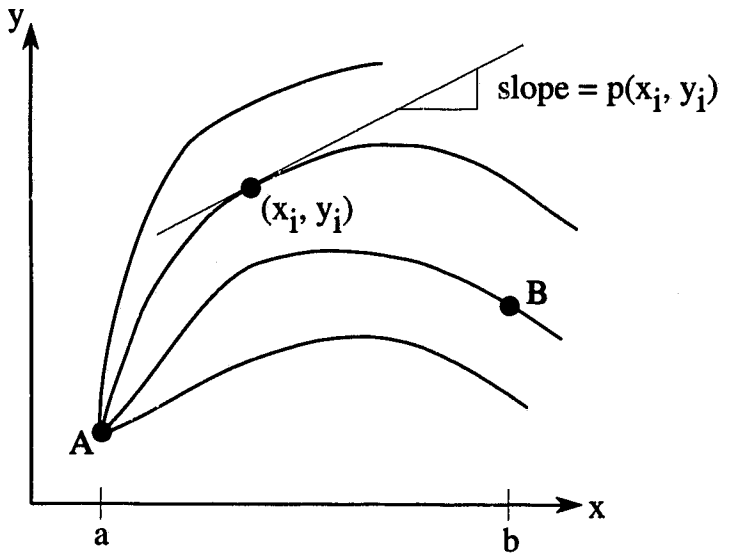


Figure 2.5 Slope of a Field of Extremals

extremal in zero order closeness. Assume that the Jacobi condition is satisfied on the extremals and let there be a field of extremals whose derivative at each point in the field is given by $p\,(x, y)$. Consider the integral

$$\int_a^b [F(x, y, p) + (y' - p)\, F_p(x, y, p)]\, dx \tag{2.67}$$

On the extremal $y(x)$ this integral becomes the simple functional given by equation 2.52. Equation 2.67 may be written,

$$\int_A^B ([F(x, y, p) - pF_p(x, y, p)]\, dx + F_p(x, y, p) dy) \tag{2.68}$$

Since this equations integrand is a total differential, the integral is independent of the path of integration. Noting that

$$F_p(y' - p) = 0 \tag{2.69}$$

on the extremal, equation 2.66 may be written

$$\Delta J = \int_H F(x, y, y')dx - \int_{extr} [F(x, y, p) - (y' - p) F_p] dx \tag{2.70}$$

It has just been shown that the second integral is independent of the path of integration. Therefore

$$\Delta J = \int_H [F(x, y, y') - F(x, y, p) - (y' - p) F_p] dx \tag{2.71}$$

or

$$\Delta J = \int_H E dx \tag{2.72}$$

where the function

$$E(x, y, y', p) = F(x, y, y') - F(x, y, p) - (y' - p) F_p \tag{2.73}$$

is called the Weierstrass equation. To achieve a strong minimum of the simplest functional, equation 2.52, it is sufficient that for any y' the inequality

$$E \geq 0 \tag{2.74}$$

and the Jacobi condition be satisfied in the neighborhood of the extremal. The inequality of the opposite sign is sufficient for a strong maximum. This condition is known as the Weierstrass condition. If the function $F(x, y, y')$ can be differentiated three times with respect to y' the Weierstrass condition can be simplified to

$$F_{y'y'} \geq 0 \tag{2.75}$$

Unlike the Legendre condition this inequality must be satisfied not only on the extremal but also in its neighborhood

Many mathematicians continued to work on variational problems. Weierstrass (1865) and Erdmann (1875) studied extremals with break points and established conditions which an extremal should satisfy at these break points. A more complete theory of discontinuous extremals was developed by Razmadze (1890-1929) and later by Krotov (1961). Variational problems with constraints imposed on the desired functions and/or their derivatives were examined by a number of researchers in the 19th century. In 1913, Garnett gave a general formulation of the calculus of variations for such closed domain problems. In the most general case, the extremum is composed of curve sections from the solution to the Euler equations and segments of the domain boundary. Variational methods were not widely used in control engineering practice until the development of rocketry. Control problems, such as minimum fuel consumption trajectories, provided the impetus for a renewed interest in variational methods and spurred the development of a new field of study; optimum control theory.

2.2.5 The Maximum Principle

One approach to the optimal control problem was advanced by a group of Soviet mathematicians, Pontryagin, Boltyanski, Gamkrelidze, and Mishchenko. The result of their investigations [59] is called the maximum principle. The maximum principle is applicable to systems whose behavior is described by differential equations of the form

$$\dot{\vec{x}} = \vec{f}[\vec{x}(t), \vec{u}(t), t]$$

$\vec{x}(t_0)$ known initial condition

$$t_0 \le t \le t_f \tag{2.76}$$

where $\vec{x}(t)$ is the n-dimensional state vector, $\vec{u}(t)$ the m-dimensional control vector and t is time. Due to the type of problems out of which the maximum principle arose, the notation and terminology differ from that used in classical variational calculus. Unlike problems in the calculus of variations, the state coordinates and controls are separated. This is found to be particularly useful in cases where constraints are placed on the control but not on the state. A scalar performance index of the following form is specified.

$$J = \Phi[\vec{x}(t_f), t_f] + \int_{t_0}^{t_f} L[\vec{x}(t), \vec{u}(t), t]\, dt \tag{2.77}$$

Here, Φ is a penalty term which is a function of the final state of the system and the final time and L represents an integrable term to be minimized (maximized). For example, choosing $L = 1$ and minimizing J defines the minimum time control problem. The optimal control problem may now be stated as follows.

> Find the control vector $\vec{u}(t)$ for $t_0 \le t \le t_f$, such that the system described by state equations 2.76 is controlled in such a way as to make the performance index J a minimum (maximum).

In order for the control to be optimal, in the sense that J is minimized, a set of necessary conditions must be obeyed. Adjoining the system equations to the performance index using n Lagrange multipliers $\vec{\lambda}(t)$ gives

$$J = \Phi[\vec{x}(t_f), t_f] + \int_{t_0}^{t_f} \{L[\vec{x}(t), \vec{u}(t), t] + \vec{\lambda}^T(t)\{f[\vec{x}(t), \vec{u}(t), t] - \dot{\vec{x}}(t)\}\}\, dt \tag{2.78}$$

The variables $\vec{\lambda}(t)$ are commonly referred to as the costate or influence variables. Define the scalar function H called the Hamiltonian as

$$H[\vec{x}(t), \vec{u}(t), t] = L[\vec{x}(t), \vec{u}(t), t] + \vec{\lambda}^T f[\vec{x}(t), \vec{u}(t), t] \tag{2.79}$$

The performance index now becomes,

$$J = \Phi[\vec{x}(t_f), t_f] + \int_{t_0}^{t_f} \{H[\vec{x}(t), \vec{u}(t), t] - \vec{\lambda}^T(t)\dot{\vec{x}}(t)\}\, dt \tag{2.80}$$

Integrate the right most term in equation 2.80 by parts to yield,

$$\begin{aligned} J = {} & \Phi[\vec{x}(t_f), t_f] - \vec{\lambda}^T(t_f)\vec{x}(t_f) + \vec{\lambda}^T(t_0)\vec{x}(t_0) \\ & + \int_{t_0}^{t_f} \{H[\vec{x}(t), \vec{u}(t), \vec{\lambda}(t)] + \dot{\vec{\lambda}}^T(t)\vec{x}(t)\}\, dt \end{aligned} \tag{2.81}$$

Consider the first variation in J due to variations in the control vector.

$$\delta J = \left[\left(\frac{\partial \Phi}{\partial \vec{x}} - \vec{\lambda}^T\right)\delta\vec{x}\right]_{t=t_f} + \left[\vec{\lambda}^T \delta\vec{x}\right]_{t=t_0} + \int_{t_0}^{t_f} \left[\left(\frac{\partial H}{\partial \vec{x}} + \dot{\vec{\lambda}}^T\right)\delta\vec{x} + \frac{\partial H}{\partial \vec{u}}\delta\vec{u}\right] dt \tag{2.82}$$

To remove the need to determine the variations in the state $\delta\vec{x}(t)$ produced by variations in the control vector $\delta\vec{u}(t)$ the coefficients of $\delta\vec{x}$ are eliminated by choosing the costate equations

$$\dot{\vec{\lambda}}^T = -\frac{\partial H}{\partial \vec{x}} \tag{2.83}$$

with the boundary conditions

$$\vec{\lambda}^T(t_f) = \frac{\partial \Phi}{\partial \vec{x}} \tag{2.84}$$

The first variation of the performance index is now

$$\delta J = \vec{\lambda}^T(t_0)\delta\vec{x}(t_0) + \int_{t_0}^{t_f} \frac{\partial H}{\partial \vec{u}} \delta\vec{u} dt \tag{2.85}$$

For an extremum it is necessary that $\delta J = 0$ for arbitrary $\delta\vec{u}(t)$. This will only occur if

$$\frac{\partial H}{\partial \vec{u}} = 0 \qquad t_0 \le t \le t_f \tag{2.86}$$

The problem may now be formulated as,

$$\begin{aligned} \dot{\vec{x}} &= f(\vec{x}, \vec{u}, t) \\ \dot{\vec{\lambda}} &= -\left(\frac{\partial H}{\partial \vec{x}}\right)^T \end{aligned} \tag{2.87}$$

with $\vec{u}(t)$ given by,

$$\frac{\partial H}{\partial \vec{u}} = 0 \tag{2.88}$$

and boundary conditions

$$\vec{x}(t_0) \quad \text{given}$$

$$\vec{\lambda}(t_f) = \left(\frac{\partial \Phi}{\partial \vec{x}}\right)^T \tag{2.89}$$

Therefore, solving the optimal control problem requires the solution of $2n$ differential equations with some boundary conditions specified at the initial time t_0 and some at the terminal time t_f. The above problem is a two point boundary value problem and in all but the simplest cases requires the use of numerical techniques to solve.

In the above development, no constraints were placed on the control vector. Pontryagin and his collaborators were able to generalize the above method to provide a necessary condition when the control vector is constrained. The resulting maximum principle may be stated as follows.

> Let $\vec{u}(t)$, $t_0 \le t \le t_f$ be an admissible control such that starting with initial conditions $\vec{x}_0$, the trajectory passes the point $\vec{x}(t_f)$ at some time t_f. If $\vec{u}(t)$ is optimal in the sense of minimizing the performance index J, then there exists a nonzero, continuous vector $\vec{\lambda}(t)$ which satisfies the following equations.

$$\dot{\vec{x}}(t) = \left(\frac{\partial H}{\partial \vec{\lambda}}\right)^T$$

$$\dot{\vec{\lambda}}(t) = -\left(\frac{\partial H}{\partial \vec{x}}\right)^T \tag{2.90}$$

> In addition,
>
> 1. For all t in the interval $[t_0, t_f]$, the Hamiltonian H attains it supremum with respect to $\vec{u}$. This is expressed mathematically by,

$$M(\vec{\lambda}, \vec{x}) = H(\vec{\lambda}, \vec{x}, \vec{u}) \tag{2.91}$$

> where

$$M(\vec{\lambda}, \dot{\vec{x}}) = \sup_{\dot{\vec{u}}} H(\vec{\lambda}, \dot{x}, \dot{u}) \tag{2.92}$$

2. At $t = t_f$, $\vec{\lambda}(t_f)$, $M[\vec{\lambda}(t_f), \dot{\vec{x}}(t_f)] = 0$.
3. The vector $\vec{\lambda}(t_f)$ is orthogonal to the tangent hyperplane of the smooth manifold or surface containing $\dot{x}(t)$.

As with Euler's equation, the maximum principle only provides necessary conditions for an extremum. The most important value of the maximum principle is its facility of use in cases where there is no extremal and the control is made up of sections of the boundary of the control domain. This makes the maximum principle particularly useful for systems where the functionals and coupling equations are linear and constraints are imposed only on the controls.

2.2.6 Dynamic Programming

At about the same time Pontryagin was working on his maximum principle, the dynamic programming technique was described by Bellman [5] and expanded upon for the solution of control problems in [6]. Bellman's approach is different from the variational approach in that he begins with a hypothesis called the "*principle of optimality*" and then derives a partial differential equation. The solution to this equation determines a set of curves on which the extremum may be achieved. This equation is analogous to Euler's equation in the variational approach. The principle of optimality may be stated as follows.

> Regardless of the initial state or the initial control decision, the remaining control decisions must constitute an optimal policy with respect to the state resulting from the initial decision.

The theory of dynamic programming was developed for a much broader class of problems than just those described by differential equations. As a consequence the principle of opti-

mality is very general. The principle of optimality has not been proved in its most general form although it has been shown to hold for causal processes in which systems described by differential equations are included.

Dynamic programming is an extremal field approach. It determines the form of the optimal control from a large number of initial points to the terminal manifold. The extremal field is determined by minimizing the performance index

$$J = \Phi [\dot{x}(t_f), t_f] + \int_{t_0}^{t_f} L [\dot{x}(\tau), \dot{u}(\tau), \tau] \, d\tau \tag{2.93}$$

for the dynamic system described by the state equations

$$\dot{\dot{x}} = \dot{f}(\dot{x}, \dot{u}, t) \tag{2.94}$$

subject to the terminal boundary conditions

$$\Psi [\dot{x}(t_f), t_f] = 0 \tag{2.95}$$

Let the optimal return be

$$J^* (\dot{x}, t) = min \left\{ \Phi [\dot{x}(t_f), t_f] + \int_{t_0}^{t_f} L [\dot{x}, \dot{u}, \tau] \, d\tau \right\} \tag{2.96}$$

with the boundary condition that

$$J^* (\dot{x}, t) = \Psi (\dot{x}, t) \tag{2.97}$$

on the terminal manifold defined by equation 2.95. Suppose that the system proceeds for some small time Δt from an initial point $(\dot{x}, t)$, under control $\dot{u}(t)$ which is not optimal. The new state of the system is

$$\dot{x} + \dot{f}(\dot{x}, \dot{u}, t)\Delta t \tag{2.98}$$

Assume that the control from this point onward is optimal. A first order approximation of the performance index is therefore

$$J' = J^* [\dot{x} + \dot{f}(\dot{x}, \dot{u}, t)\Delta t, t + \Delta t] + L(\dot{x}, \dot{u}, t)\Delta t \tag{2.99}$$

Since optimal control was not used during the Δt time step, the resulting return function must be greater than or equal to the optimal return function. The return will only be optimal if $\dot{u}(t)$ is chosen during the time interval Δt to minimize equation 2.99.

$$J^* (\dot{x}, t) = \min_{\dot{u}(t)} \{J^* [\dot{x} + \dot{f}(\dot{x}, \dot{u}, t)\Delta t, t + \Delta t] + L(\dot{x}, \dot{u}, t)\Delta t\} \tag{2.100}$$

Assuming that $J^*(\dot{x}, t)$ is continuous and has continuous first and second partial derivatives in the region of interest of $\dot{x} - t$ space, a Taylor series expansion of equation 2.100 gives

$$J^*(\dot{x}, t) = \min_{\dot{u}(t)} \{J^*(\dot{x}, t) + \frac{\partial J^*}{\partial \dot{x}} \dot{f}(\dot{x}, \dot{u}, t)\Delta t + \frac{\partial J^*}{\partial t}\Delta t + L(\dot{x}, \dot{u}, t)\Delta t\} \tag{2.101}$$

Since J^* in not explicitly dependent on $\dot{u}$, equation 2.101 may be written by passing to the limit as Δt tends to zero.

$$-\frac{\partial J^*}{\partial t} = \min_{\dot{u}} \{L(\dot{x}, \dot{u}, t) + \frac{\partial J^*}{\partial \dot{x}} \dot{f}(\dot{x}, \dot{u}, t)\} \tag{2.102}$$

Equation 2.102 is called Bellman's equation. It is a first order, nonlinear, partial differential equation whose boundary condition is given by equation 2.97. In practice it is more common to use an algorithm based directly on the principle of optimality rather than to attempt the solution of Bellman's equation.

Approximate solutions to continuous problems can be achieved by replacing them with appropriate discrete problems. It is assumed that the control is piecewise constant in time. The state differential equations become the state difference equations,

$$\hat{x}(k+1) = \hat{g}\,[\hat{x}(k), \hat{u}(k), k] \tag{2.103}$$

where $\hat{x}$ is the state vector and $\hat{u}$ the control vector. Here, k is the stage variable and often represents the discretized time variable. The performance index is given by,

$$J = \sum_{k=0}^{K} L\,[\hat{x}(k), \hat{u}(k), k] \tag{2.104}$$

where L is the cost for a single stage. Constraints may be placed on both state and control vectors. The constraints on the state vector may be functions of the stage and constraints on the control vector may be functions of both the state and stage. Define the minimum cost function from state $\hat{x}$ at stage k to the terminal manifold at final stage K as follows.

$$\begin{aligned} I(x, k) &= \min_{\hat{u}(j)} \{ \sum_{j=k}^{K} L\,[\hat{x}(j), \hat{u}(j), j] \} \\ j &= k, k+1, \ldots, K \end{aligned} \tag{2.105}$$

The sum can be separated into two parts by splitting off the $j = k$ term of the summation.

$$I(\dot{x}, k) = \min_{\dot{u}(k)} \{ L[\dot{x}, \dot{u}, k] + \min_{\dot{u}(j)} \left[\sum_{j=k+1}^{K} L[\dot{x}(j), \dot{u}(j), j] \right] \}$$
$$j = k+1, \ldots K \tag{2.106}$$

This equation may be rewritten,

$$I(\dot{x}, k) = \min_{\dot{u}} \{ L(\dot{x}, \dot{u}, k) + I[g(\dot{x}, \dot{u}, k), k+1] \} \tag{2.107}$$

Equation 2.107 is just a mathematical expression for Bellman's principle of optimality. It states that the minimum cost in going from state $\dot{x}$ at stage k to the terminal manifold is achieved by choosing the control that minimizes the cost to be levied at the current stage plus the cost of going from the state at stage $k+1$, resulting from the chosen control, to the terminal manifold. Since $I(\dot{x}, k)$ and $u(\dot{x}, k)$ are determined in terms of $I(\dot{x}, k+1)$ the problem is solved backwards from the terminal manifold. The terminal boundary condition is given by,

$$I(\dot{x}, k) = \min_{\dot{u}} \{ L(\dot{x}, \dot{u}, k) \} \tag{2.108}$$

A digital computer is usually used to solve equation 2.107 in the following manner.

1. Quantize each state variable x_i $i = 1, \ldots, n$, into N_i levels and each of the control variables u_j $j = 1, \ldots, m$ into M_j levels.
2. Find $I(\dot{x}, K)$ for all quantized states which satisfy the constraints by evaluating $L(\dot{x}, \dot{u}, k)$ for all feasible $\dot{u}$ and doing a direct comparison to find the minimum. As often happens, no control is required at the final stage. In this case $I(\dot{x}, K)$ is simply $L(\dot{x}, K)$.
3. At stage $k = K-1$, for each quantized state, each quantized control is applied and the resulting state $g(\dot{x}, \dot{u}, K-1)$ is computed. The minimum cost of the resulting state is computed by interpolating between the stored values of $I(\dot{x}, K)$. $L(\dot{x}, \dot{u}, K-1)$ is then computed. The sums of

these values for each quantized control are then compared and the smallest value stored in $I(\hat{x}, K-1)$. The optimal control $u^*(\hat{x}, K-1)$ is the value for which the minimum was obtained.

4. The procedure continues moving backwards through stages, computing $I(\hat{x}, k)$ and $\hat{u}^*(\hat{x}, k)$ in terms of $I(\hat{x}, k+1)$ until $k = 0$.
5. The optimal control sequence is recovered by performing a forward pass; interpolating between the stored $\hat{u}^*(\hat{x}, k)$ values. The initial control $\hat{u}^*(\hat{x}_0, 0)$ is read directly. The next state is calculated as $\hat{g}\,[\hat{x}_0, \hat{u}^*(\hat{x}_0, 0), 1]$. The required control at the next stage is evaluated by interpolating between the $\hat{u}^*(\hat{x}, 1)$ values at quantized $\hat{x}$. The procedure continues until the entire control sequence is recovered at $k = K$.

Note that this procedure determines the optimal control for any initial state starting at any stage. A family of optimal control decisions has been generated.

The maximum principle and Bellman's equation are closely related. In fact, the problems addressed by the maximum principle may be considered a subclass of those dealt with by dynamic programming. Again consider the system state equations 2.76 and the performance index, equation 2.77. First treat time as an additional state variable by appending it to the n-dimensional state vector $\hat{x}(t)$.

$$x_{n+1} = t; \quad \dot{x}_{n+1} = f_{n+1}; \quad x_{n+1}(0) = 0 \tag{2.109}$$

Consider the adjoint variables

$$\vec{\lambda} = \left(\frac{\partial J^*}{\partial x_1}, \frac{\partial J^*}{\partial x_2}, \ldots, \frac{\partial J^*}{\partial x_n}\right) \tag{2.110}$$

where J^* is given by equation 2.100. Since J^* is not explicitly dependent on $\hat{u}$, $\frac{\partial J^*}{\partial t}$ may be moved inside the term to be minimized in equation 2.102. Noting that

$$sup(-Z) = -min(Z) \tag{2.111}$$

for any Z, Bellman's equation may be written

$$\max_{\vec{u}} \{L(\vec{x}, \vec{u}, t) + \vec{\lambda}^T \vec{f}\} = 0 \tag{2.112}$$

From equation 2.79 the Hamiltonian is

$$H = L(\vec{x}, \vec{u}, t) + \vec{\lambda}^T \vec{f} \tag{2.113}$$

Therefore,

$$\max_{\vec{u}} H = 0 \tag{2.114}$$

which is the maximum principle.

Chapter 3
Literature Survey

The problem of computing optimal trajectories for robotic manipulators has been a subject of research since the late 1960's. Stepanenko [80], first formulated the minimal energy consumption problem in 1970, noting that energy consumption depended essentially on the kinematic properties of the manipulator and the dissipative characteristics of its actuators. In this paper, actuators are classified based on their energy consumption characteristics in the regimes of static load and negative work. In the static load regime the manipulator is held stationary against an applied force; for example gravity. The negative works regime is characterized by a decrease in potential of the system. Minimum energy is not the only performance criteria considered in the literature. In fact, the minimum time cost criteria is more common. Other performance criteria which have also been considered are, weighted time/energy, reaction forces, control effort and obstacle avoidance. In the paragraphs which follow, the methods in the literature which have been applied to problems of the above type are briefly discussed.

3.1 Minimum Time Performance Criteria

3.1.1 Variational Approaches

Perhaps the first attempt at determining the minimum motion time trajectory between two points in joint space for a robotic manipulator was given by Kahn and Roth [34][35]. Using the classical variational approach of the maximum principle, they derived the twelve nonlinear ordinary differential equations of the two point boundary value problem which arise for a three link manipulator. Further, they were able to show that the control solution to this problem must be bang-bang. That is, at any given time, at least one of the joint controls will be at its saturation or limit point. The solution to the two point boundary problem proved difficult due to its ill-conditioned nature. As a suboptimal alternative, Kahn and Roth linearized and decoupled the equations of motion and analytically derived the optimal control law for the three resulting double integrator systems. It was shown that the responses generated by the suboptimal control strategy resulted in trajectories with the same general character of the optimal solution for many prescribed moves. However, the linearized control scheme can lead to significant error for some motions with a significantly degraded motion time performance. Other attempts to solve optimal motion problems for robotic manipulators include [86][25][51][21][16]. In [16], Chen and Derochers demonstrate that the minimum-time problem is singular. As demonstrated by Jacobson [33], the minimum-time control problem can be converted to a non-singular form by adding an energy term to the performance index. As successive iterations of the optimization algorithm are completed, the weight multiplying the "energy" term is reduced. Bessonnet and Lallemand [7], use a mixed time control effort performance index to overcome the problem of singularity. A weighting factor on the control effort portion of the performance index is reduced at each iteration of the algorithm. Even with these additional parameters, the authors note that a good initial estimated solution is required if the algo-

rithm is to converge. Examples in the above literature are restricted to one or two link arms. Even for problems of this low dimensionality, solution of the two point boundary value problem depends on a reasonably accurate starting estimate of the boundary conditions or requires partitioning the nonlinear system into linear and nonlinear parts and then slowly introducing the nonlinear terms over several iterations using an adjustable weighting factor. The highly nonlinear coupling of more complex manipulators makes the solution of the two point boundary value problem extremely difficult, if not impossible. More recently, Fourquet, [24] uses analytical results provided by optimal control theory to investigate the structure of solutions to the time-optimal point-to-point motion control problem. The goal was to use this information to reduce the complexity of numerical search routines by restricting the classes of functions searched for the optimum.

3.1.2 Prespecified Path

More success has been achieved using more direct approaches on a simplified version of the minimum time problem. For this problem, the manipulator path is assumed to be specified, either in Cartesian or joint space, by a set of knot points connected by interpolating segments; usually straight lines or splines.[48][49][14][44][11][12][13][20][70] [73][74][75][56][72][41][39][43][26] The minimum travelling time along the specified path is sought subject to constraints on velocity and acceleration. This is in effect a one dimensional problem. Assuming a inverse kinematic solution for the manipulator in question exists, it is just necessary to determine the velocity with which the end effector moves to provide minimum travel time. This will of course be as fast as possible subject to constraints.

Luh and Lin addressed the minimum-time problem along a path specified in Cartesian space.[49] While recognizing that the real physical constraints on the manipulator are

the applied forces and torques at the joints, they cited the difficulty, due to the highly non-linear coupled nature of the system, in converting bounds on forces and torques into those on accelerations and velocities of each joint as the reason for specifying constraints on accelerations and velocities in Cartesian space. These constraints were determined from experimental data. The required motion along the preplanned path subject to the Cartesian constraints was formulated as a nonlinear programming problem. The solution was found by reformulating the problem and solving it as a sequence of linear programing problems with all nonlinearities iteratively linearized.

Bobrow, Dubowsky and Gibson took a more direct approach to this minimum time problem. The nonlinear dynamic and torque constraints are transformed into nonlinear state dependant constraints on the acceleration along the parametrically prescribed path. The acceleration is then viewed as the control. The functions $f(x, \dot{x})$ and $g(x, \dot{x})$ bound the end effector acceleration $\ddot{x}$, which must not violate the constraints on actuator torque for the given x and $\dot{x}$.

$$f(x, \dot{x}) \leq \ddot{x} \leq g(x, \dot{x}) \tag{3.1}$$

It is possible that, for some values of x and $\dot{x}$, $f(x, \dot{x}) > g(x, \dot{x})$. In these cases no $\ddot{x}$ will satisfy the bounds given by equation 3.1. In these cases, the actuators can no longer hold the manipulator on the specified trajectory without violating at least one of the constraints. The time optimal solution is achieved by choosing the acceleration to make the velocity as large as possible without violating the constraints. It is demonstrated that the time is minimized when the acceleration takes on either its largest or smallest possible value. Finding the optimal control therefore amounts to finding the points at which the acceleration switches between its maximum and minimum values. An algorithm which finds these switching points in the parametric phase plane of Figure 3.1 is provided. An example is

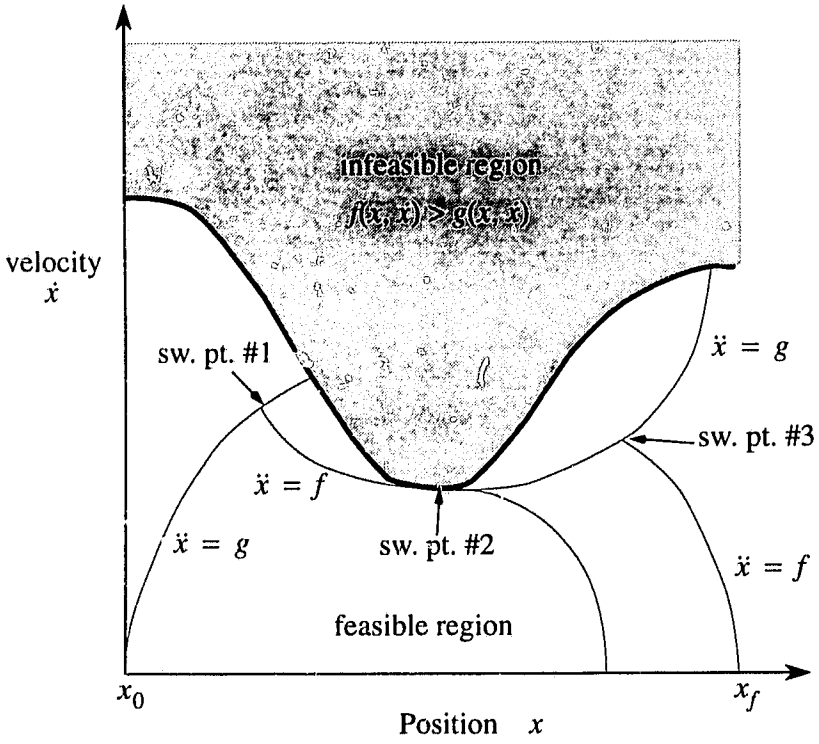


Figure 3.1 Parametric Path Variable Phase Plane

given for a three link arm. Dubowsky and Shiller [.'0] applied this technique to a six link manipulator and extended the technique to include payload and gripper constraints [70]. Shin and McKay [73][74][75], and Pfeiffer and Johanni [56], proposed a similar approache, which differed in the method used for determining the switching points. Shin and McKay noted that when the effects of viscous friction at the joints were considered, islands of inadmissibility, appeared in Bobrow's admissible regions. Pfeiffer and Johanni were able to characterize points of tangency to the limit curve. Tangency occurs at either a "naturally" tangent point, where the acceleration is unique, or at a "critical" point where the acceleration can be selected from a finite range. The assumption of minimum acceleration or deceleration may fail at "critical' points where the maximum acceleration causes the trajectory to cross the limit curve and violate at least one constraint. The above algo-

rithms get trapped at such points. Shiller and Lu [72] developed a modified algorithm to avoid critical points or reshape the path to control the location of the critical points.

3.1.3 Unspecified Path

Some researchers have proposed methods to solve the minimum time problem when the path is not specified before hand. Sahar and Hollerbach [66] tessellated joint space into a grid. The requirement for time optimality and the dynamics of the arm constrain the number of velocities possible at each position node. Therefore, there is no need to tessellate velocity space. A tree of all possible state-space paths is constructed and then searched for the minimum-time path using any of a number or search techniques; for example best first or branch and bound. The authors claim that since each actuator is always at its torque bound, there are at most $2n$ possible transitions from a given state to a neighboring position. A time scaling algorithm determines, for each transition, the velocity of the neighboring node and its transition time. The technique was demonstrated for a two-link manipulator. A geometric increase in the size of the search space for manipulators with higher degrees of freedom makes the use of this technique impractical. The assumption that each actuator is at is torque bound is also invalid. One can only guarantee that at least one of the actuators will be at its torque bound at any one time.

Rajan [62], parameterized what is called configuration space by fitting cubic splines through a set of n-tuple knot points representing the 'n' joint angles of an 'n' degree of freedom manipulator. For a path described by a given set of knot points, he used the algorithm of Bobrow to compute the minimum time trajectory. The path is then changed by varying the knot points and the process repeated. To determine the path with minimum time the knot space is searched. This is carried out first by discretizing the knot space and searching through the discrete points to localize the minimum and then using

gradient descent to find the minimum more precisely. For large numbers of knot points a global minimum would be hard to achieve since the discrete search through a parameter space of many dimensions would be time consuming. Shiller and Dubowsky [71], and Bobrow [13] present similar algorithms for finding the minimum time trajectory. Shiller and Dubowsky perform a hierarchical search for the best path in tessellated space. Five levels of pruning are used to remove all paths but the best paths in regions of local optima. These paths are then used as starting points for a local path optimization. Bobrow represents the geometric path of the manipulator as a set of parametric B-spline interpolation functions. The constraints are represented by cubic interior penalty functions and an unconstrained minimization is performed using the Davidon-Fletcher-Powell search technique [23] with the method of Golden Sections for the one dimensional search.

Shin and McKay [77] derive two criteria for selecting near-minimum time geometric paths for a manipulator. The first is based on minimization of the lower bound of the manipulator's traversal time and the other based on the minimization of the product of path length and curvature. In both cases the near-optimal path is found to be a geodesic in inertia space. The geodesics can be constructed by solving a two point boundary value problem. The advantage of the proposed method over the direct application of the maximum principle is that the boundary conditions for finding the geodesics are easier to apply than are those which arise when using the maximum principle. They demonstrate by example that their derived approximate solutions usually require less time than Cartesian straight-line paths and joint interpolated paths.

Podhorodeski and Cleghorn [58] present a direct search technique to determine minimum time trajectories comprised of cubic splines in joint space. Manipulator tasks are specified as a sequence of required end effector locations and orientations in task

space. Enforcement of joint constraints is achieved by scaling infeasible manipulation kinematics and dynamics through the expansion of total motion time.

3.2 Minimum Energy Performance Criteria

The number of publications dealing with minimum energy performance criteria is significantly less than that for the minimum time problem. Often the actuators are assumed to be simple torque sources and "minimum energy" is taken to mean "minimize the time integral of the sum of the squares (or sum of absolute values) of the control torques" [84][38]. Real actuators are more complex than this and their dissipative characteristics must be taken into account when computing energy consumption for a specified trajectory. Vukobratovic and Kircanski [85], Shin and McKay [76] and Singh and Leu [78] each present methods for optimizing energy or time/energy performance criteria along specified paths is space. Each of the methods use a dynamic programming approach. In [85], the travel time is given, the path is discretized in terms of time and along each discrete path segment the velocity in considered constant. In [76], the path is represented parametrically by a scalar parameter. The equations representing the manipulator dynamics, constraints and performance index are all expressed in this parameter and its time derivative. The dynamic programming approach is then applied to a grid spanned by the parametric variable and its time derivative. In [78], the path is discretized in Cartesian space and the acceleration along each discrete path segment is assumed constant. Field [21][22] assumes no prespecified path but assumes quasi-static conditions of operation. A dynamic programming approach is applied. Chou and Song [17] also assume quasi-static conditions. Their approach is to analyze a property called geometric work. Geometric work is defined as the sum of the absolute values of all energy consumed by the actuators minus the absolute value of the work done by the system on the environment. They note that the major contributor to geometric work for most actuators is the energy dissipated

when the actuator acts as a brake. The manipulator's joint space is analyzed to find paths of zero geometric work. These maps are then applied to path planning for minimum energy consumption. A dynamic programming technique for finding the optimal paths in presented. The quasi-static assumption will be invalidated if time constraints are placed on the motion. For example a manipulator may be required to complete a specified move within a certain time frame to coordinate with other equipment. Also, the energy dissipation of some actuation systems have a time dependent component. Depending on the relative contribution of this component to the overall energy consumption, motion will not necessarily be slow enough to use a quasi-static assumption. This dissertation addresses the problem of minimizing the energy consumption for a transition between two given states, taking the into full account the robot dynamics and the dissipative characteristics of the actuators.

Chapter 4
Energy Consumption in Robotic Systems

Determining minimum energy trajectories for robotic manipulators requires the development of energy consumption models for robotic systems. Since robotic systems are interdisciplinary in that they require modelling of subsystems that are mechanical, electrical and perhaps hydraulic, and the consumption of energy is of primary interest, bond graphs are an ideal tool with which to model these systems. A bond graph is a dual-signal-flow diagram representation of a physical system. They are especially useful for systems whose component parts are the topic of study for a variety of engineering disciplines. These include electrical, mechanical, hydraulic, chemical and thermal systems. Bond graphs are able to represent each of these types of systems in a systematic and unified manner. This ability is derived from the use of energy flow through a system as the parameter to bind the model of a physical system together. Appendix A provides a brief introduction to bond graph terminology and notation. Other useful bond graph references include, Paynter [55], Karnopp and Rosenberg [36][37], and Thoma [81][82] for general usage and application, Breedveld [15], for multibond graph notation, and Bos and Tiernego [8][9] for modelling of multibody systems. As pointed out by Thoma [82], it is bes-

to build up a bond graph model of a complex physical system one piece at a time, taking time to check the correctness of each subsystem model. This is the approach used here. First, a single link is modelled as a rigid body moving in space. A method is presented for connecting links together to form a multibody chain. This method is applied to a two link manipulator example. The equations of motion are extracted from the bond graph and are shown to be the same as those which could be obtained by either a Newton-Euler or Lagrangian formulation. A model for a three link arm is presented. Next, models of actuator systems are considered, first in a general manner, and then in the development of an energy consumption model for a common robotic actuator: a direct current servo motor with a gear reduction unit. Finally these elements are put together to develop an energy consumption model for the first three links of the Reis V15 industrial manipulator so that results achieved using the path planning methods developed in this dissertation may be experimentally validated.

4.1 Modelling a Robotic Mechanism

4.1.1 Bond Graph Model of A Single Link

Figure 4.1 shows a single link in a kinematic chain modelled as a body moving in space with respect to an inertial coordinate frame 0_{xyz}. Using a frame assignment which follows Craig [18], a body-fixed frame, B_{xyz}, is attached to the link such that the frame's z-axis is coincident with the joint axis of the link. A second body-fixed frame, G_{xyz}, is attached at the link's center of gravity. This frame's axes are chosen to align with the principle axes of inertia of the body. Point P on the link is the point at which the link is connected to the next body in the kinematic chain. Define the vector $\vec{r}^{\,k}_{i,j}$ to be the position of point 'i' with respect to point 'j' written with respect to a frame oriented the same as

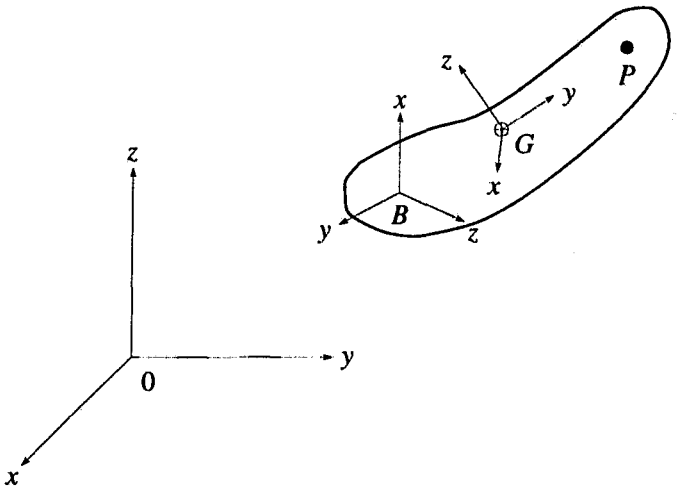


Figure 4.1 Free Moving Body

frame 'k'. The position of point P with respect to the origin of the inertial base frame 0, written in the inertial base frame is therefore given by,

$$\vec{r}^{\,0}_{P,0} = \vec{r}^{\,0}_{B,0} + \vec{r}^{\,0}_{P,B}. \tag{4.1}$$

The velocity of point P with respect to the base frame, obtained by differentiating equation 4.1, is,

$$\dot{\vec{r}}^{\,0}_{P,0} = \dot{\vec{r}}^{\,0}_{B,0} + \dot{\vec{r}}^{\,0}_{P,B} = \dot{\vec{r}}^{\,0}_{B,0} + \vec{\omega}^{0}_{B,0} \times \vec{r}^{\,0}_{P,B} \tag{4.2}$$

where $\vec{\omega}^{0}_{B,0}$ is the angular velocity of the body-fixed frame with respect to the base frame written in base frame coordinates. Let R^{j}_{i} define a rotational coordinate transformation matrix which describes frame 'i' with respect to frame 'j'. Then equation 4.2 may be written in matrix notation as,

$$\dot{\vec{r}}^{\,0}_{P,0} = \dot{\vec{r}}^{\,0}_{B,0} + R^{0}_{B}\,(X(\vec{r}^{\,B}_{P,B})\vec{\omega}^{B}_{B,0}) \tag{4.3}$$

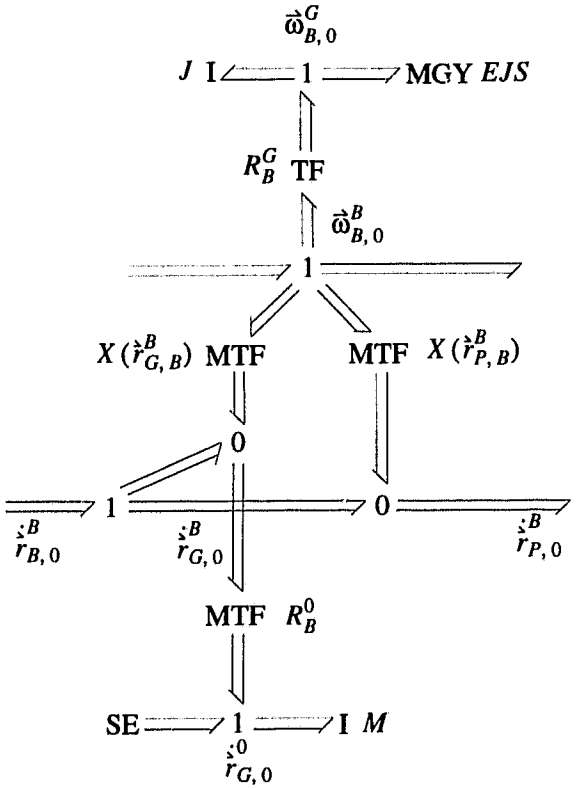


Figure 4.2 Multibond Graph Representing the Dynamics of a Free Moving Body

where $X(\dot{r}^{B}_{P,B})$ is the skew symmetric matrix,

$$X(\dot{r}^{B}_{P,B}) = \begin{bmatrix} 0 & r^{Bz}_{P,B} & -r^{By}_{P,B} \\ -r^{Bz}_{P,B} & 0 & r^{Bx}_{P,B} \\ r^{By}_{P,B} & -r^{Bx}_{P,B} & 0 \end{bmatrix} \tag{4.4}$$

The motion consists of a velocity due to the translation of the body $\dot{r}^{0}_{B,0}$ and a velocity due to the rotation of the body $X(\dot{r}^{B}_{P,B})\vec{\omega}^{B}_{B,0}$. A similar relationship can be derived for the body's center of gravity, point G, by replacing the P subscript in equation 4.3 by a G subscript. These velocity relationships for the body are embodied in the junction structure of the multibond graph in Figure 4.2.

The dynamics of the body are represented by adding inertia elements. The mass matrix M, is connected as an I-element to the 1-junction representing the translational velocity of the mass in base frame. An effort source is also added here to represent the gravitational force. The rotary inertia matrix J, is connected as an I element to the 1-junction representing the rotational velocities in the body frame. The coupling forces which act between these velocities are represented by the modulated gyrator MGY. In Figure 4.2, the rotational velocities are first transformed to a second body frame at the center of gravity whose axes are aligned with the body's principle axes. Although not necessary, performing this transformation allows the rotary inertia matrix J to be represented in diagonal form and allows the simplest form of Euler's equations of motion to be used to describe the gyrator MGY. The general form of the bond graph for a multibody interconnected system can be developed from this bond graph of a single body.

4.1.2 Bond Graph Model of a Multibody Chain

Figure 4.2, shows the bond graph representation for body 'i'. The upper 1-junction and bonds represent the angular velocity of the body in the body coordinate frame. The lower 1-junction and bonds represent the translational velocity of the body. The vertical branches of bonds represent the interaction between translation and rotation. The branch with the attached I-element represents the interaction with the mass center. The second branch provides the velocity for the body point P where the coupling with body 'i+1' is to occur. Individual body models may then be connected together to form a bond graph for the multibody system. Figure 4.3 shows how one link is connected to the next via a pair of modulated transformers and a pair of 0-junctions, where appropriate angular and/or linear velocities are added in to represent inputs from actuators connecting the two bodies.

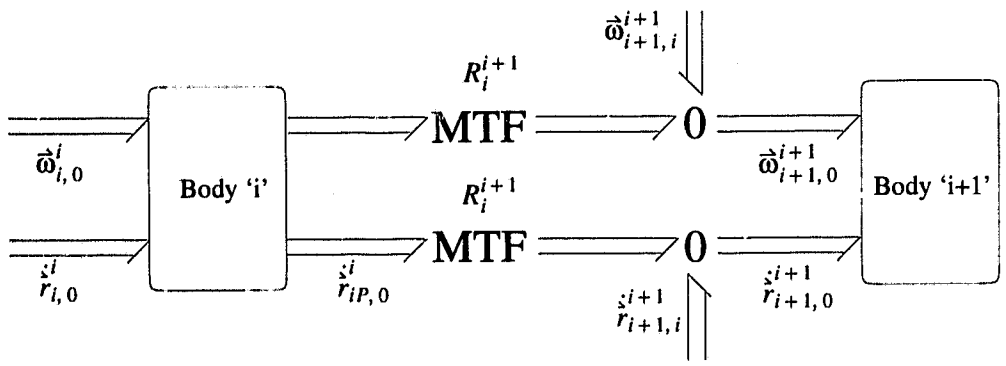


Figure 4.3 Connecting Bodies to Form a Multibody System

4.1.3 Bond Graph Model of a Two-Link Manipulator

The basic structure of Figure 4.3 may be used to construct the bond graph representation of the two link arm given in Figure 4.4. The arm has two bodies driven by two revolute actuators. Since the first body does not translate, the bond graph for this particular multibody system can be simplified from the general case. This simplification is apparent in the word bond graph where there is no connection between the translational velocities of the two bodies. Note the further simplifications in the Body 1 portion of the bond graph model. Since the mass center lies on the first body's axis of rotation, elements representing translational motion are not required. The joint actuators are represented as effort sources. These will be replaced by actuator models in later sections.

The parameters in the bond graph are as follows. The rotary inertia of the first body and the rotary and linear inertia of body two are given by,

$$J_1^{1G} = \begin{bmatrix} J_{1x}^{1G} & 0 & 0 \\ 0 & J_{1y}^{1G} & 0 \\ 0 & 0 & J_{1z}^{1G} \end{bmatrix} \qquad J_2^{2G} = \begin{bmatrix} J_{2x}^{2G} & 0 & 0 \\ 0 & J_{2y}^{2G} & 0 \\ 0 & 0 & J_{2z}^{2G} \end{bmatrix} \qquad M_2 = \begin{bmatrix} m_2 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_2 \end{bmatrix} \tag{4.5}$$

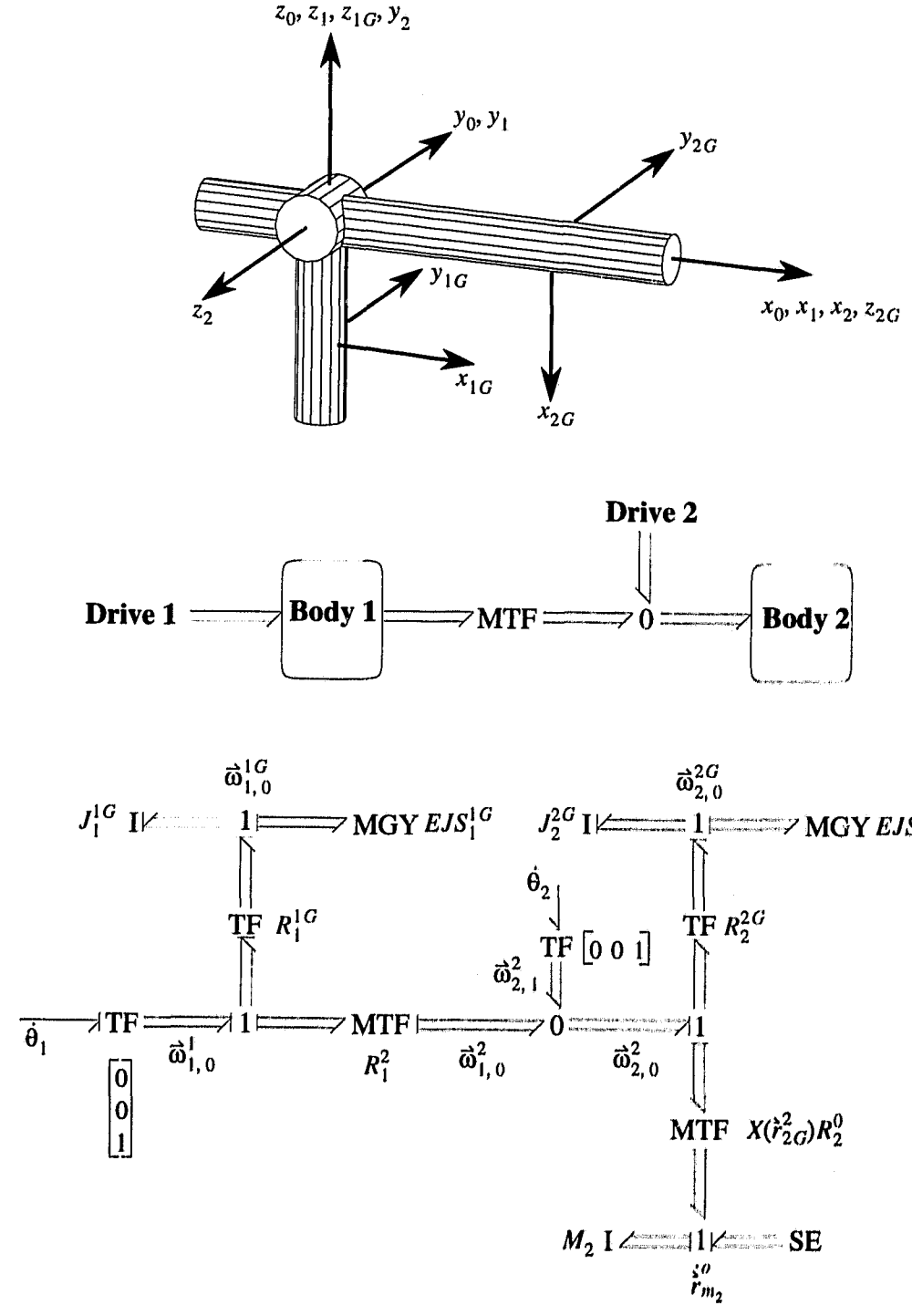


Figure 4.4 Bond Graph Model of a Two-Link Arm

The Eulerian junction structures used to model centripetal and Coriolis effects are,

$$EJS_1^{1G} = \begin{bmatrix} 0 & J_{1z}^{1G}\omega_{1z}^{1G} & -J_{1y}^{1G}\omega_{1y}^{1G} \\ -J_{1z}^{1G}\omega_{1z}^{1G} & 0 & J_{1x}^{1G}\omega_{1x}^{1G} \\ J_{1y}^{1G}\omega_{1y}^{1G} & -J_{1x}^{1G}\omega_{1x}^{1G} & 0 \end{bmatrix} \tag{4.6}$$

$$EJS_2^{2G} = \begin{bmatrix} 0 & J_{2z}^{2G}\omega_{2z}^{2G} & -J_{2y}^{2G}\omega_{2y}^{2G} \\ -J_{2z}^{2G}\omega_{2,z}^{2G} & 0 & J_{2x}^{2G}\omega_{2x}^{2G} \\ J_{2y}^{2G}\omega_{2y}^{2G} & -J_{2x}^{2G}\omega_{2x}^{2G} & 0 \end{bmatrix} \tag{4.7}$$

The transformers required to convert between frames of reference are,

$$R_1^{1G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_2^{2G} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \qquad R_1^2 = \begin{bmatrix} c_2 & -s_2 & 0 \\ 0 & 0 & -1 \\ s_2 & c_2 & 0 \end{bmatrix}$$

$$R_2^0 = \begin{bmatrix} c_1c_2 & s_1c_2 & s_2 \\ -c_1s_2 & -s_1s_2 & c_2 \\ s_1 & -c_1 & 0 \end{bmatrix} \qquad X(r_{2G}^2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & r_{2G_x}^2 \\ 0 & -r_{2G_x}^2 & 0 \end{bmatrix} \qquad \begin{aligned} s_1 &= \sin(\theta_1) \\ c_1 &= \cos(\theta_1) \\ s_2 &= \sin(\theta_2) \\ c_2 &= \cos(\theta_2) \end{aligned} \tag{4.8}$$

Assigning causality requires that one of the inertia elements get differential causality, a possible source of instability for numerical simulation. In this example, the inertia element representing the translational inertia of Body 2 is assigned differential causality. This mixed differential integral causality indicates that a dependency exists between the inertia elements. This dependency arises regularly in systems subject to a large number of kinematic constraints. Integral causality may be achieved by generating the equivalent bond graph which results from transforming the inertia elements in differential causality

to those in integral causality. The transformation procedure should be restricted to that part of the bond graph where all of the dependencies occur. This portion of the bond graph was referred to by Allen as the mechanism[2]. For the two link arm, the entire bond graph of Figure 4.4 constitutes the mechanism. In the equivalent bond graph, all inertias will be combined into one multiport I-element connected to a junction structure. Capacitor, resistor, source and gyrator elements are treated in a like manner.

Since this system has two degrees of freedom, there are two independent velocities. The choice of independent velocities, $\dot{\vec{q}}_i$, is not unique but the form of the resulting equations does depend on the choice made. For the two link arm model, the most obvious choice is

$$\dot{\vec{q}}_i = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}. \tag{4.9}$$

The velocities associated with the inertia elements, $\dot{\vec{q}}_I$, are found from the bond graph to be

$$\dot{\vec{q}}_I = \begin{bmatrix} \vec{\omega}_{1,0}^{1G} \\ \vec{\omega}_{2,0}^{2G} \\ \dot{\vec{r}}_{m_2,0}^{0} \end{bmatrix}. \tag{4.10}$$

The inertial velocities can be written as a function of the independent velocities by tracing through the transformations in the bond graph from the inertial velocities to the independent velocities. That is,

$$\dot{\vec{q}}_I = \begin{bmatrix} \vec{\omega}_{1,0}^{1G} \\ \vec{\omega}_{2,0}^{2G} \\ \dot{\vec{r}}_{m_2,0}^{0} \end{bmatrix} = \begin{bmatrix} \left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_1^{1G} \begin{bmatrix} \dot{\theta}_1 \end{bmatrix}\right)^T \\ \left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_1^2 R_2^{2G} \begin{bmatrix} \dot{\theta}_1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_2^{2G} \begin{bmatrix} \dot{\theta}_2 \end{bmatrix}\right)^T \\ \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_1^2 X(\dot{\vec{r}}_{2G}^{2}) R_2^0 \begin{bmatrix} \dot{\theta}_1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} X(\dot{\vec{r}}_{2G}^{2}) R_2^0 \begin{bmatrix} \dot{\theta}_2 \end{bmatrix}^T \end{bmatrix} \tag{4.11}$$

$$\dot{\vec{q}}_I = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ -c_2 & 0 \\ 0 & -1 \\ s_2 & 0 \\ s_1 c_2 r_{2G_x}^2 & c_1 s_2 r_{2G_x}^2 \\ -c_1 c_2 r_{2G_x}^2 & s_1 s_2 r_{2G_x}^2 \\ 0 & -c_2 r_{2G_x}^2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.12}$$

$$\dot{\vec{q}}_I = T_i^I \dot{\vec{q}}_i \tag{4.13}$$

Transforming the multiport inertia containing all inertial elements to the independent velocities is equivalent to the transformation of a multiport inertia over a modulated transformer. Allen [2] shows that this results in a virtual inertia, $\tilde{I}$, and a gyristor GR. From Figure 4.5, the equations for the modulated transformer are

$$\dot{\vec{q}}_I = T_i^I \dot{\vec{q}}_i \tag{4.14}$$

$$\vec{e}_i = (T_i^I)^T \vec{e}_I. \tag{4.15}$$

The equation for the I-element is

Figure 4.5 Transforming a Multiport Inertia Over a Modulated Transformer

$$\dot{e}_I = I_n \frac{d\dot{q}_I}{dt} \tag{4.16}$$

where

$$I_n = diag\left[J_{1x}^{1G}\ J_{1y}^{1G}\ J_{1Z}^{1G}\ J_{2x}^{2G}\ J_{2y}^{2G}\ J_{2z}^{2G}\ m_2\ m_2\ m_2\right] \tag{4.17}$$

is a diagonal matrix of inertia elements. Substituting equation 4.14 into equation 4.16 gives

$$\dot{e}_I = I_n T_i^I \frac{d\dot{q}_i}{dt} + I_n \frac{dT_i^I}{dt}\dot{q}_i. \tag{4.18}$$

From equation 4.15 and equation 4.18 the independent effort variables are expressed as

$$\dot{e}_i = (T_i^I)^T I_n T_i^I \frac{d\dot{q}_i}{dt} + (T_i^I)^T I_n \frac{dT_i^I}{dt}\dot{q}_i. \tag{4.19}$$

The coefficient,

$$\tilde{I} = (T_i^I)^T I_n T_i^I \tag{4.20}$$

is called the transformed inertia. The second coefficient,

$$GR = (T_i^I)^T I_n \dot{T}_i^I \tag{4.21}$$

relates effort and flow as occurs in multiport gyrators and resistors. This element is called a gyristor. For simplicity, assume that the links can be represented by point masses. Then all rotational inertias become zero. In practical application, this simplification would not be required since the equations of motion would be left in the matrix form obtained from the bond graph and the computer would perform the required transformation operations. However, since the intent here is to show the equivalence of the bond graph technique with other methods of formulating the equations of motion, it is necessary to continue symbolically. Substituting the appropriate expressions into equations 4.20 and 4.21 gives,

$$\tilde{I} = \begin{bmatrix} m_2 (r_{2G_x}^2)^2 c_2^2 & 0 \\ 0 & m_2 (r_{2G_x}^2)^2 \end{bmatrix} \tag{4.22}$$

and

$$GR = \begin{bmatrix} \dfrac{-(m_2 r_{2G_x}^2 \sin(2\theta_2)\,\dot{\theta}_2)}{2} & \dfrac{-(m_2 r_{2G_x}^2 \sin(2\theta_2)\,\dot{\theta}_1)}{2} \\ \dfrac{m_2 r_{2G_x}^2 \sin(2\theta_2)\,\dot{\theta}_1}{2} & 0 \end{bmatrix} \tag{4.23}$$

The two Eulerian junction structures, EJS_1^{1G} and EJS_2^{2G} may be expressed in terms of the independent velocities by noting, from the bond graph, that

$$\vec{\omega}_{1,0}^{1G} = ([0\ 0\ 1] R_1^{1G})^T \dot{\theta}_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \tag{4.24}$$

and

$$\vec{\omega}_{2,0}^{2G} = \left(\left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_1^2 \right) R_2^{2G} \dot{\theta}_1 + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R_2^{2G} \dot{\theta}_2 \right)^T = \begin{bmatrix} -c_2 \dot{\theta}_1 \\ -\dot{\theta}_2 \\ s_2 \dot{\theta}_1 \end{bmatrix}. \tag{4.25}$$

Substituting into equation 4.6, an overall Eulerian junction structure for the mechanism may be written as,

$$EJS = \begin{bmatrix} EJS_1^{1G} & 0 \\ 0 & EJS_2^{2G} \end{bmatrix} \tag{4.26}$$

where

$$EJS_1^{1G} = \begin{bmatrix} 0 & J_{1z}^{1G} \dot{\theta}_1 & 0 \\ -J_{1z}^{1G} \dot{\theta}_1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad EJS_2^{2G} = \begin{bmatrix} 0 & J_{2z}^{2G} s_2 \dot{\theta}_1 & J_{2y}^{2G} \dot{\theta}_2 \\ -J_{2z}^{2G} s_2 \dot{\theta}_1 & 0 & -J_{2x}^{2G} c_2 \dot{\theta}_1 \\ -J_{2y}^{2G} \dot{\theta}_2 & J_{2x}^{2G} c_2 \dot{\theta}_1 & 0 \end{bmatrix}. \tag{4.27}$$

This Eulerian junction structure is then transformed to the independent velocities by using the rotational portion of T_i^I. This submatrix will be denoted $T_{i,rot}^I$ and is given by the first six rows of T_i^I.

$$\tilde{EJS} = (T_{i,rot}^I)^T \, EJS \, T_{i,rot}^I \tag{4.28}$$

Of course, since the point mass assumption is in effect EJS and as a result $\tilde{EJS}$, are both zero matrices. Together, GR and $\tilde{EJS}$ account for all centripetal and Coriolis effects.

Finally, the effort source elements are combined into a single multiport and related to the chosen independent velocities. The effort variables associated with these effort sources are denoted,

$$\dot{e}_{SE} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ 0 \\ 0 \\ -m_2 g \end{bmatrix}. \tag{4.29}$$

The associated flow variables are expressed in terms of the independent velocities with the matrix T_i^{SE}.

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{x}_{m_2}^0 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 & 0 \\ 0 & \dot{\theta}_2 \\ ([0\ 0\ 1]R_1^2 X(r_{2G}^2)R_2^0)^T\dot{\theta}_1 & ([0\ 0\ 1]X(r_{2G}^2)R_2^0)^T\dot{\theta}_2 \end{bmatrix} \tag{4.30}$$

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{x}_{m_2}^0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ s_1 c_2 r_{2G_x}^2 & c_1 s_2 r_{2G_x}^2 \\ -c_1 c_2 r_{2G_x}^2 & s_1 s_2 r_{2G_x}^2 \\ 0 & -c_2 r_{2G_x}^2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.31}$$

$$\dot{q}_s = T_i^{SE}\dot{q}_i \tag{4.32}$$

The bond graph for the equivalent system is shown in Figure 4.6. Note that the multiport inertia has integral causality. The dynamic equations for the system are gleaned from the bond graph by summing the effort variables at the 1-junction.

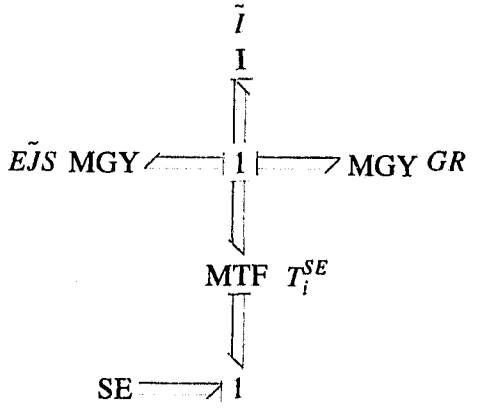


Figure 4.6 Equivalent System Bond Graph

$$-\grave{e}_{\tilde{I}} - \grave{e}_{GR} - \grave{e}_{E\tilde{J}S} + (T_i^{SE})^T \grave{e}_{SE} = 0 \tag{4.33}$$

$$-\tilde{I}\ddot{q}_i - GR\dot{q}_i - E\tilde{J}S\dot{q}_i + (T_i^{SE})^T \grave{e}_{SE} = 0 \tag{4.34}$$

Substituting the element expressions from above and solving for the joint torques yields,

$$\tau_1 = m_2 (r_{2G_x}^2)^2 (\cos(\theta_2))^2 \ddot{\theta}_1 - m_2 r_{2G_x}^2 \sin(2\theta_2)\dot{\theta}_1\dot{\theta}_2 \tag{4.35}$$

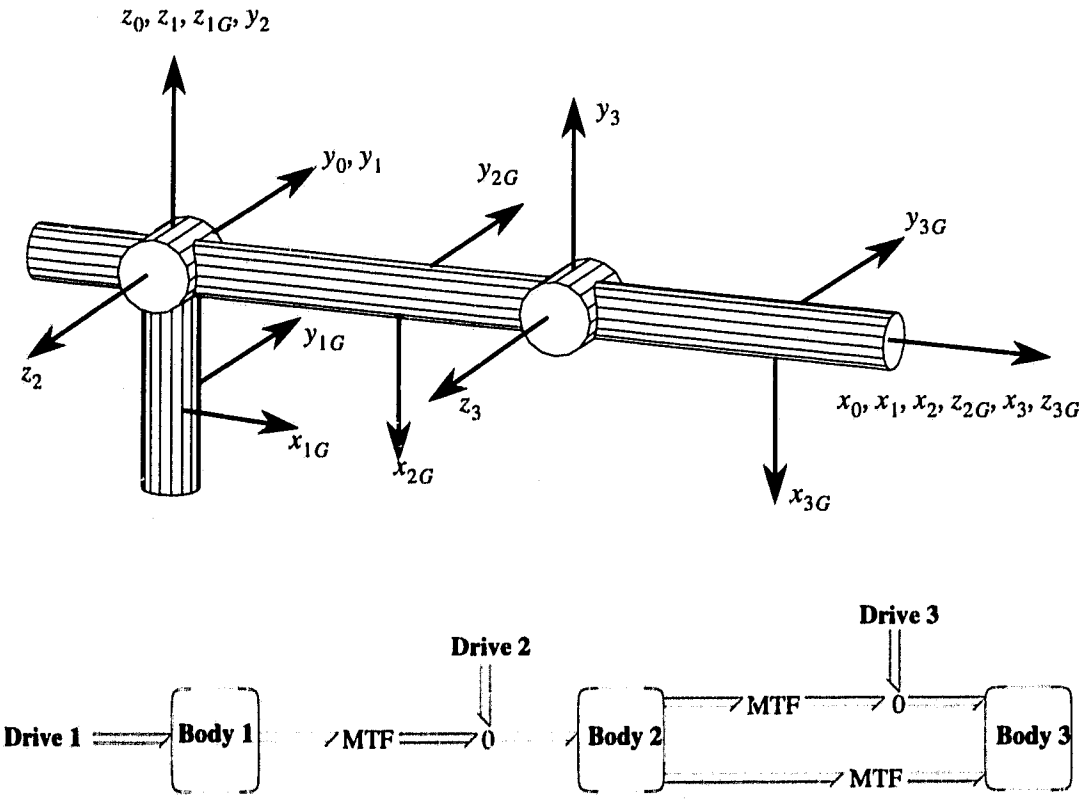
$$\tau_2 = m_2 (r_{2G_x}^2)^2 \ddot{\theta}_2 + \frac{1}{2} m_2 r_{2G_x}^2 \sin(2\theta_2)\dot{\theta}_1^2 + m_2 g r_{2G_x}^2 \cos(\theta_2) \tag{4.36}$$

Identical results are achieved with either a Lagrangian or Newton-Euler formulation. For example, [69] and [42] each contain a worked example which differ only in link length.

4.1.4 Bond Graph Model of a Three-Link Manipulator

The gross motions, which consume the majority of the energy in most robotic systems, require at least a three-link model. The bond graph model of the previous section can be extended to include an arbitrary number of links. Figure 4.7 shows a three link arm with frame assignment and its associated word bond graph. The full bond graph model in Figure 4.8 is constructed by adding an additional link to the end of the two link arm modelled in the previous section. The parameters of the bond graph are as follows. The rotational inertias of each link, expressed in a body frame aligned with the principle axes of inertia are,

Figure 4.7 Frame Assignment and Word Bond Graph of a Three Link Arm



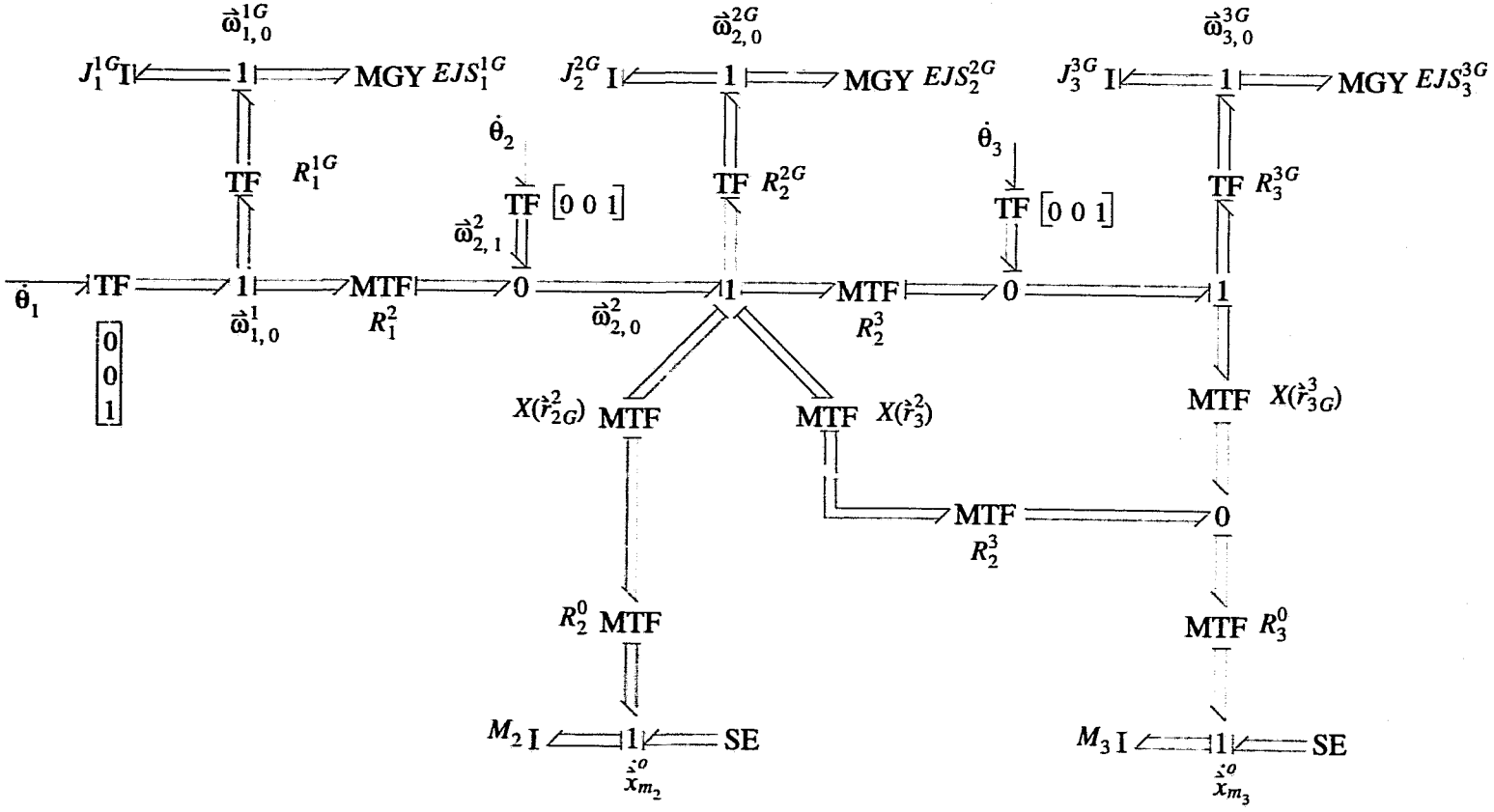


Figure 4.8 Bond Graph Model of a Three Link Arm

$$J_1^{1G} = \begin{bmatrix} J_{1x}^{1G} & 0 & 0 \\ 0 & J_{1y}^{1G} & 0 \\ 0 & 0 & J_{1z}^{1G} \end{bmatrix} \qquad J_2^{2G} = \begin{bmatrix} J_{2x}^{2G} & 0 & 0 \\ 0 & J_{2y}^{2G} & 0 \\ 0 & 0 & J_{2z}^{2G} \end{bmatrix} \qquad J_3^{3G} = \begin{bmatrix} J_{3x}^{3G} & 0 & 0 \\ 0 & J_{3y}^{3G} & 0 \\ 0 & 0 & J_{3z}^{3G} \end{bmatrix} \tag{4.37}$$

The link masses are represented by the matrices,

$$M_2 = \begin{bmatrix} m_2 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_2 \end{bmatrix} \qquad M_3 = \begin{bmatrix} m_3 & 0 & 0 \\ 0 & m_3 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \tag{4.38}$$

Coupling terms are modelled by the Eulerian Junction structures,

$$EJS_1^{1G} = \begin{bmatrix} 0 & J_{1z}^{1G}\omega_{1z}^{1G} & -J_{1y}^{1G}\omega_{1y}^{1G} \\ -J_{1z}^{1G}\omega_{1z}^{1G} & 0 & J_{1x}^{1G}\omega_{1x}^{1G} \\ J_{1y}^{1G}\omega_{1y}^{1G} & -J_{1x}^{1G}\omega_{1x}^{1G} & 0 \end{bmatrix} \tag{4.39}$$

$$EJS_2^{2G} = \begin{bmatrix} 0 & J_{2z}^{2G}\omega_{2z}^{2G} & -J_{2y}^{2G}\omega_{2y}^{2G} \\ -J_{2z}^{2G}\omega_{2,z}^{2G} & 0 & J_{2x}^{2G}\omega_{2x}^{2G} \\ J_{2y}^{2G}\omega_{2y}^{2G} & -J_{2x}^{2G}\omega_{2x}^{2G} & 0 \end{bmatrix} \tag{4.40}$$

$$EJS_3^{3G} = \begin{bmatrix} 0 & J_{3z}^{3G}\omega_{3z}^{3G} & -J_{3y}^{3G}\omega_{3y}^{3G} \\ -J_{3z}^{3G}\omega_{3,z}^{3G} & 0 & J_{3x}^{3G}\omega_{3x}^{3G} \\ J_{3y}^{3G}\omega_{3y}^{3G} & -J_{3x}^{3G}\omega_{3x}^{3G} & 0 \end{bmatrix}. \tag{4.41}$$

The required transformation matrices are,

$$R_1^{1G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_2^{2G} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \qquad R_3^{3G} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.42}$$

$$X(r_{2G}^2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & r_{2G_x}^2 \\ 0 & -r_{2G_x}^2 & 0 \end{bmatrix} \qquad X(r_{3G}^3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & r_{3G_x}^3 \\ 0 & -r_{3G_x}^3 & 0 \end{bmatrix} \qquad X(r_3^2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & r_{3_x}^2 \\ 0 & -r_{3_x}^2 & 0 \end{bmatrix} \tag{4.43}$$

$$R_1^2 = \begin{bmatrix} c_2 & -s_2 & 0 \\ 0 & 0 & -1 \\ s_2 & c_2 & 1 \end{bmatrix} \qquad R_2^3 = \begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_2^0 = \begin{bmatrix} c_1c_2 & s_1c_2 & s_2 \\ -c_1s_2 & -s_1s_2 & c_2 \\ s_1 & -c_1 & 0 \end{bmatrix}$$

$$R_3^0 = \begin{bmatrix} c_1c_{23} & s_1c_{23} & s_{23} \\ -c_1s_{23} & -s_1s_{23} & c_{23} \\ s_1 & -c_1 & 0 \end{bmatrix} \qquad \begin{aligned} s_{23} &= \sin(\theta_2 + \theta_3) \\ c_{23} &= \cos(\theta_2 + \theta_3) \end{aligned} \tag{4.44}$$

If the joint velocities $\dot{\theta}_1$, $\dot{\theta}_2$ and $\dot{\theta}_3$ are chosen as the independent velocities the equations of motion are given by equation 4.34, where

$$\tilde{I} = (T_i^I)^T I_n T_i^I \tag{4.45}$$

$$T_i^I = \begin{bmatrix} ([0\ 0\ 1]R_1^{1G})^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^{2G})^T & ([0\ 0\ 1]R_2^{2G})^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_3^{3G})^T \\ ([0\ 0\ 1]R_1^2X(r_{2G}^2)R_2^0)^T & ([0\ 0\ 1]X(r_{2G}^2)R_2^0)^T & [0\ 0\ 0]^T \\ t_{51} & t_{52} & ([0\ 0\ 1]X(r_{3G}^3)R_3^0)^T \end{bmatrix}$$

$$t_{51} = ([0\ 0\ 1]R_1^2R_2^3X(r_{3G}^3)R_3^0 + [0\ 0\ 1]R_1^2X(r_3^2)R_2^3R_3^0)^T$$

$$t_{52} = ([0\ 0\ 1]R_2^3X(r_{3G}^3)R_3^0 + [0\ 0\ 1]X(r_3^2)R_2^3R_3^0)^T \quad \textbf{(4.46)}$$

$$I_n = diag\left[J_1^{1G}\ J_2^{2G}\ J_3^{3G}\ M_2\ M_3\right] \quad \textbf{(4.47)}$$

$$GR = (T_i^I)^T I_n \dot{T}_i^I \quad \textbf{(4.48)}$$

$$\tilde{EJS} = (T_{i,rot}^I)^T EJS\ T_{i,rot}^I \quad \textbf{(4.49)}$$

$$EJS = diag\left[EJS_1^{1G}\ EJS_2^{2G}\ EJS_3^{3G}\right] \quad \textbf{(4.50)}$$

$$T_{i,rot}^I = \begin{bmatrix} ([0\ 0\ 1]R_1^{1G})^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^{2G})^T & ([0\ 0\ 1]R_2^{2G})^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_3^{3G})^T \end{bmatrix} \quad \textbf{(4.51)}$$

$$\dot{e}_{SE} = \left[\tau_1\ \tau_2\ \tau_3\ 0\ 0\ -m_2g\ 0\ 0\ -m_3g\right] \quad \textbf{(4.52)}$$

$$T_i^{SE} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ ([0\ 0\ 1]R_1^2X(r_{2G}^2)R_2^0)^T & ([0\ 0\ 1]X(r_{2G}^2)R_2^0)^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^3X(r_{3G}^3)R_3^0)^T & ([0\ 0\ 1]R_2^3X(r_{3G}^3)R_3^0)^T & ([0\ 0\ 1]X(r_{3G}^3)R_3^0)^T \end{bmatrix} \quad (4.53)$$

4.2 Robot Actuation

It can be seen from the bond graphs in the previous sections that the mechanism of a robotic manipulator is an energy conversion system. Energy is exchanged with the environment through the drives. Energy supplied by the drives is stored as potential and kinetic energy in the manipulator. Energy may also be returned to the drive system when the mechanism potential or kinetic energy decreases. The present models contain no dissipative elements. Robotic systems do, of course, dissipate energy. This energy is dissipated in the system's actuators which until this point have been modelled as effort sources. This section discusses energy dissipation in actuator systems; first in a general manner by classifying actuators according to there dissipative characteristics. Subsequently, a more detailed model of a particular common actuation system is developed. This model can be substituted for the effort source inputs to the manipulator models to create an overall model of the robotic system.

4.2.1 Actuator Classification

Stepanenko [80] proposed an actuator classification scheme based on the dissipative characteristics of actuator systems in two modes of operation; static load and negative work. In the static load regime the manipulator does not move but must support its own weight and any externally applied forces. The negative work regime implies an overall decrease in the system potential energy. The actuator classification is summarized in

Table 1 Actuator Classification

Type	Work Regime	
	static	**negative**
A1	Y	Y
A2	Y	N
B1	N	Y
B2	N	N

Table 1. Actuators which consume energy under static load are called type A; those which don't are called type B. These basic types are subdivided into those which consume energy in the regime of negative work, subtype 1 and those which don't subtype 2. For example, an actuator which consumed energy under static load and not under negative work would be classified type A2.

Type A1 actuators consume energy regardless of the work regime in which they operate. An example is a hydraulic actuator with fluid pressure provided by a fixed displacement hydraulic pump. The pump runs continuously to maintain a pressure at the input port to a servo-valve which regulates the actuator. Pressure generated in excess of the set level is bled back to the tank through a relief valve.

An electric servomotor with a transmission which is not self-braking is an example of a type A2 actuator. In the static load regime the motor supplies a torque to maintain its equilibrium position and therefore dissipates energy in its windings. In the regime of negative work, the load is permitted to fall. Energy released from the decreasing potential is most often dissipated through a resistor, but in a more sophisticated systems may be recovered for later use. (e.g. regenerative braking)

Type B actuators can be constructed by using self-locking transmissions. If the transmission is on the self-locking boundary, then under static load, the actuator remains stationary without the use of the motor. In the regime of negative work, only a small amount of power is required to move the actuator. Such an actuator would be classified type B2. If the transmission were well within the self-locking boundary, significant energy would be required to drive the actuator in the regime of negative work. This would constitute a type B1 actuator.

The actuator classification scheme presented above categorizes idealized actuators. Real actuators can only be approximated by such a classification scheme. The following section looks at the model of a common actuation system for robotic manipulators.

4.2.2 Direct Current Servomotor Actuation

A common actuation system for robotic manipulators is a direct current (DC) servomotor with a gear reducer on the output. Figure 4.9 shows a schematic representation of such an actuator. The input to the actuator is a voltage v_i. An operational amplifier is used prior to the servo-amplifier to provide armature current control. The armature current, i_a, is measured by a low resistance R_s.

$$i_a = \frac{v_s}{R_s} \tag{4.54}$$

Since the operational amplifier has a very high input impedance, its input current is negligibly small. Therefore,

$$\frac{v_i - v_{op}}{R_1} + \frac{v_s - v_{op}}{R_2} = 0 \tag{4.55}$$

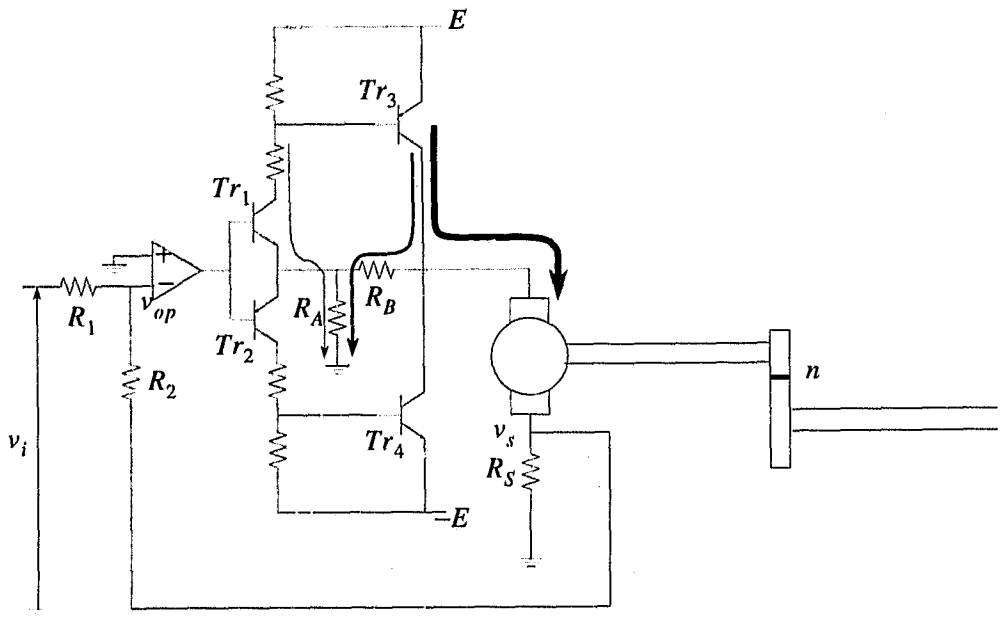


Figure 4.9 DC Servo Actuator

Let the voltage gain of the operational amplifier, servo-amplifier combination be A. Then the sensed voltage is given by,

$$v_s = -Av_{op}. \tag{4.56}$$

Solving for v_{op} and substituting into equation 4.55 gives,

$$\frac{v_i}{R_1} + \frac{v_s}{R_1 A} + \frac{v_s}{R_2} + \frac{v_s}{R_2 A} = 0 \tag{4.57}$$

Solving for the sensed voltage gives,

$$v_s = \frac{-v_i\left(\frac{R_2}{R_1}\right)}{1 + \left(\frac{1}{A}\right)\left(1 + \frac{R_2}{R_1}\right)} \tag{4.58}$$

The amplifier gain, A, is very high (typically > 10^5). If R_2 and R_1 are chosen to be of the same order of magnitude, the sensed voltage is, with little error,

$$v_s = -\frac{R_2}{R_1} v_i \tag{4.59}$$

From equation 4.54 and equation 4.59, the armature current can be expressed in terms of the input voltage.

$$i_a = -\frac{R_2}{R_1 R_s} v_i \tag{4.60}$$

For a separately excited or permanent magnet d.c. motor, the torque output of the motor is given by

$$\tau = K_m i_a \tag{4.61}$$

where K_m is the motor constant. The motor torque is then applied to the load through a gear reducer,

$$\tau_l = \frac{1}{n} \tau \tag{4.62}$$

where n is the ratio of the output to input speeds of the gear reducer.

In this actuator system, energy is dissipated by electric losses in the electric circuitry, iron losses in the motor core and mechanical losses in the motor and transmission. Almost all of the electrical losses occur in the transistors in the final amplifier stage (Tr_3 and Tr_4) and as I^2R losses in the armature circuit. Electrical currents through resistors in earlier stages of the servo-amplifier are much smaller than the armature current. Therefore the I^2R losses due to these currents will be neglected.

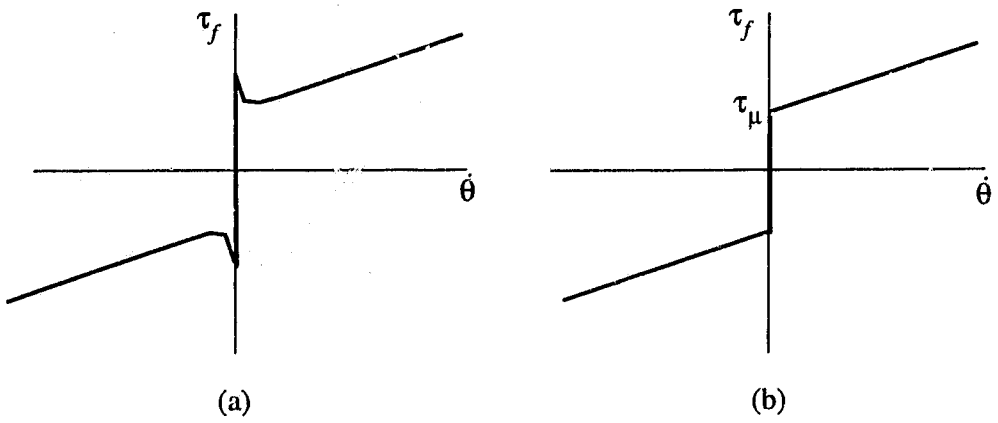


Figure 4.10 Joint Friction Characteristic

Iron losses include magnetic hysteresis and eddy current generation in the armature core. To be determined accurately, these losses must be based on empirical data. However, conventional analyses assume a loss due to hysteresis proportional to the rotational speed of the motor and eddy current loss proportional to the square of the rotational speed.

Mechanical losses include windage and friction. Windage is the air drag on the armature and is generally considered to be proportional to the square of the rotational speed of the armature. Windage is a minor loss except in large or high speed machines. Joint friction is the major source of mechanical energy dissipation. A typical joint friction characteristic has the form shown in Figure 4.10(a). As can be seen in the figure, static friction must be overcome before motion starts. The friction force drops sharply once motion starts. As speed increases, a viscous friction term, which is proportional to velocity, becomes a factor. A reasonable piecewise approximation to the friction characteristic, for the purpose of computing energy losses is shown in Figure 4.10(b). Its equation is given by,

$$\tau_f = \tau_\mu \operatorname{sgn}\dot{\theta} + K_f\dot{\theta} \tag{4.63}$$

where τ_μ is the static friction torque and K_f is a proportionality constant for viscous friction.

Figure 4.11 shows a bond graph model of the DC servo actuator system. The servo-amplifier portion of the actuator is represented as a flow source which supplies current i_a, proportional to the input voltage v_i. I^2R losses in the armature and sensor resistor are represented by the R-element R_a. Electrical losses in the final amplifier stage transistors are represented by the R-element R_{Tr}. The power dissipated in the transistor is given by the product of the collector current i_c, and the voltage across the collector emitter v_{ce}. The collector current is very nearly equal to the armature current and v_{ce} is equal to the difference between the supply voltage E and the armature voltage v_a. Therefore,

$$R_{Tr} = \frac{v_{ce}}{i_c} \approx \frac{E - v_a}{i_a} = \frac{E - i_a R_a}{i_a} \tag{4.64}$$

The conversion of electrical current into motor torque is represented by the gyrator GY with motor constant K_m. Mechanical friction is represented by R-element R_f. Iron losses may also be lumped into this R-element. The rotary inertia of the armature and transmission are modelled by I-element J_a. The transformer, TF, represents the gear reducer and the capacitive C-element models transmission compliance. The actuator model may now be attached at the drive points of the mechanism bond graph models to provide a model of the entire system.

The completed model is used to simulate the response of the robot given the input voltage v_i to each actuator. The energy supplied to the system is the time integral of the product of armature voltage v_a and the armature current i_a. Note that energy supplied is not equivalent to energy dissipated. Energy is only dissipated in the resistive elements of the model. If needed the dissipation can be calculated by integrating the products of the

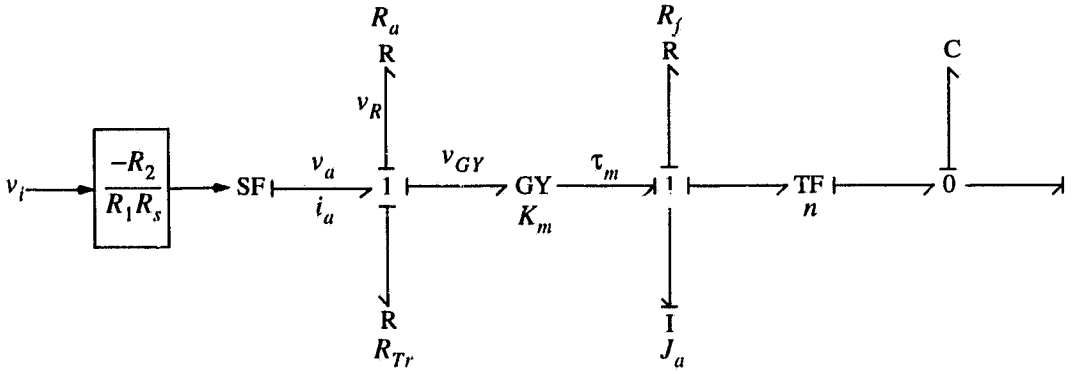


Figure 4.11 Bond Graph Model of a DC Servo Actuator

effort and flow variables in each R-element. The remainder of the supplied energy is stored in the kinetic and potential energy of the mechanism. Note that the energy flow in the flow source can be negative. This occurs when stored energy (kinetic and potential) is released by the mechanism and returned to the servo circuitry. In virtually all actuator systems in use today this energy is dissipated by a bleed off resistor in the servo circuitry. A more sophisticated servo-drive could store this returned energy for later use. If energy released by the mechanism is dissipated, the simplest way to evaluate the system's energy consumption is to integrate the energy supplied to the system when the energy supply is positive. If the energy supply is negative, integrate zero.

The bond graph model described above is suitable for evaluating a particular system design. If, on the other hand, the interest is in planning an optimal energy trajectory, the input to the model, is a trajectory whose energy consumption requires evaluation, not the servo input voltage v_i. In this situation the mechanism is treated as a state dependent load on the actuators. The bond graph of Figure 4.12 has been modified to reflect this change by replacing the 0-junction and connected elements with an effort source whose magnitude is determined by solution of the inverse dynamics of the manipulator mechanism. The armature current is given by,

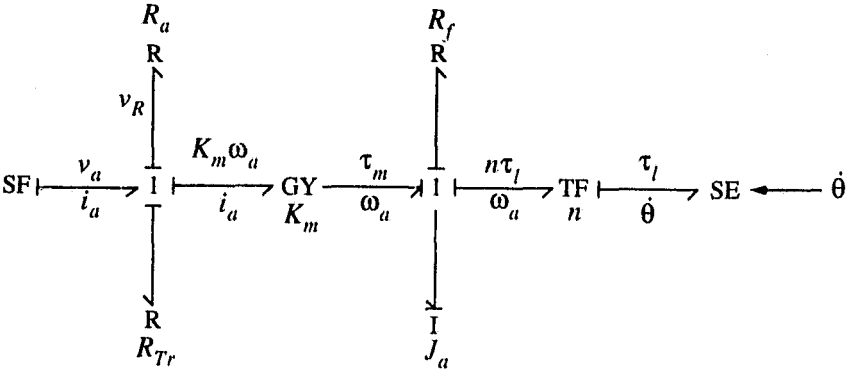


Figure 4.12 Bond Graph to Compute Energy Consumption of a Specified Trajectory

$$i_a = \frac{\tau_m}{K_m} \tag{4.65}$$

where τ_m is given by summing the effort variables at the right most 1-junction.

$$\tau_m = J_a\dot{\omega}_a + n\tau_l + R_f(\omega_a) \tag{4.66}$$

where $R_f(\omega_a)$ is a nonlinear function of the armature angular velocity representing the joint friction. The armature voltage is determined by summing the effort sources at the left most 1-junction.

$$v_a = (R_a + R_{Tr})\, i_a + K_m\omega_a \tag{4.67}$$

The power supplied is the product of v_a and i_a. The energy consumed for the specified trajectory is the time integral of

$$\frac{(1 + \text{sgn}\,(i_a v_a))\, i_a v_a}{2}. \tag{4.68}$$

4.3 Energy Consumption Model for the Reis V15 Industrial Manipulator

In order to validate any method developed for determining optimal energy consumption paths it is necessary to test that the paths produced by the optimal path planning algorithm provide some improvement over conventionally planned trajectories when tested on an actual arm. For this reason it is necessary to generate an approximate energy consumption model for a real manipulator. Paths planned using this model will then be executed on the actual manipulator and the real energy consumption measured. This energy consumption may then be compared to the energy consumption for a conventionally planned path to gage the success of the optimal path planning algorithm.

The manipulator available to this researcher is a Reis V15 industrial manipulator. This arm is a six degree of freedom revolute joint manipulator. Each joint is actuated by a DC servomotor with a harmonic drive gear reducer. The kinematic structure of the first three links is identical to that of the arm modelled in Section 4.1.4. The final three joints form a spherical wrist. For the following reasons, only the first three joints will be modelled. The majority of the energy dissipation occurs in the gross motion joints; in this instance the first three joints. Restricting the model to three links keeps it relatively simple. Also, hardware to provide motor current measurements necessary to determine the manipulator's actual energy consumption is only available on the servo drives of the first three motors of this particular manipulator.

4.3.1 Mechanism Model

The bond graph representation of the Reis V15 manipulator is identical to the bond graph developed for the three link manipulator in Section 4.1.4 except for an additional branch to represent the translation of the mass centre of the first link. This branch is neces-

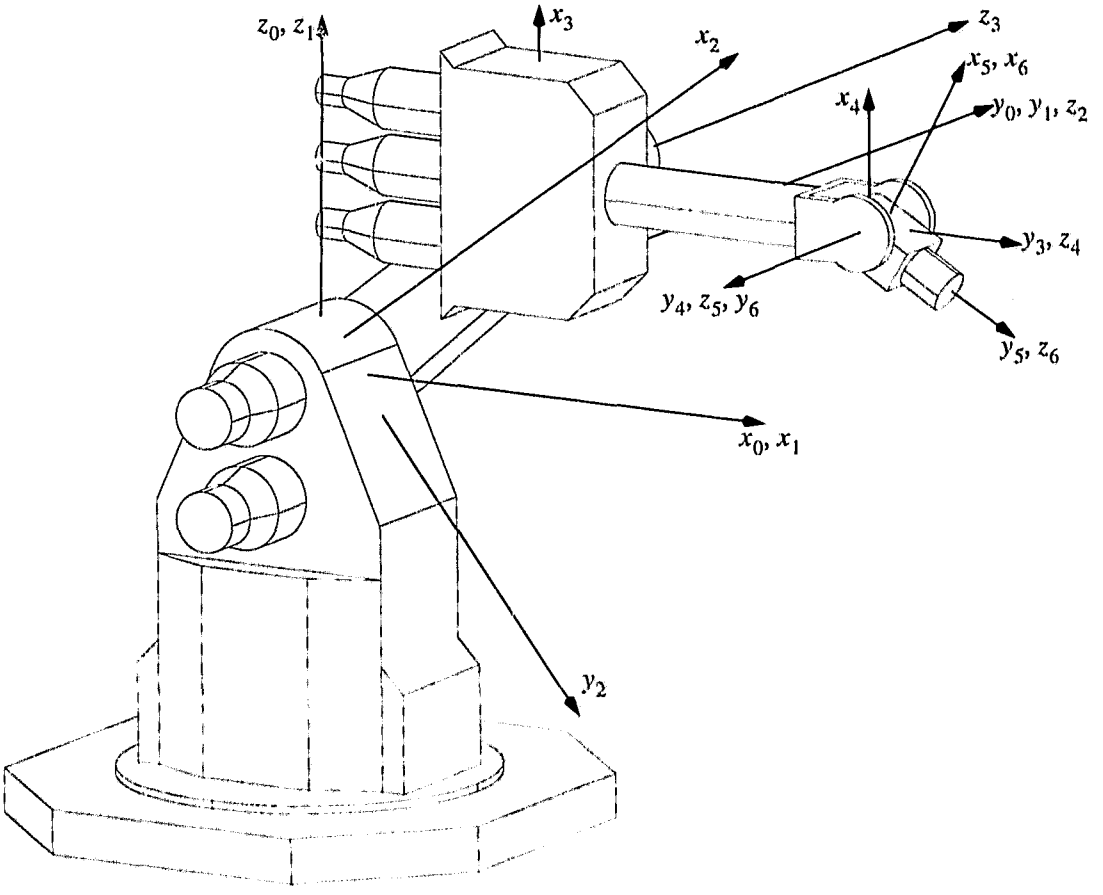


Figure 4.13 Reis V15 Manipulator

sary since the mass distribution of the real arm does not place the mass centre of the first link on the axis of rotation of the first joint. The Reis V15 manipulator with frame assignment is shown Figure 4.13. The bond graph model is shown in Figure 4.14. The mass and coupling parameters are given by equations 4.37 through 4.41 with the addition of a matrix to represent the mass of the first link

$$M_1 = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_1 & 0 \\ 0 & 0 & m_1 \end{bmatrix}. \tag{4.69}$$

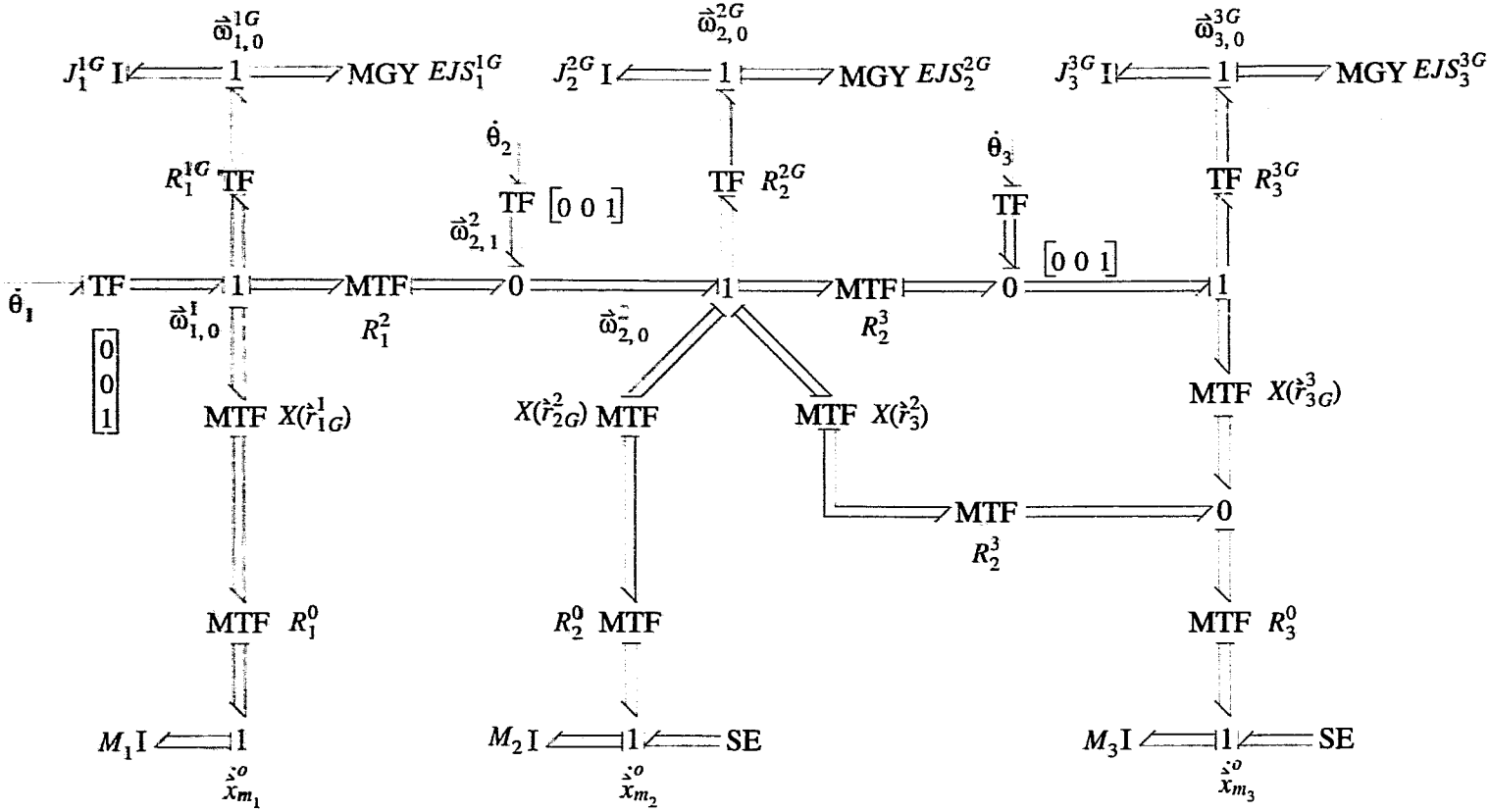


Figure 4.14 Bond Graph Model for First Three Links of the Reis V15 Manipulator

The frame assignment for the manipulator is such that the axes of the body fixed frames located at each link's mass centre are aligned with the axes of the body fixed frames located at each link's joint with the previous link. Therefore,

$$R_1^{1G} = R_2^{2G} = R_3^{3G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.70}$$

The more complex mass distribution also requires changes to transformation matrices used to form the cross product of a links angular velocity with a vector from the origin of the links body fixed frame located at the joint to the link's mass centre. For the general case these transformation matrices are,

$$X(r_{1G}^1) = \begin{bmatrix} 0 & r_{1G_z}^1 & -r_{1G_y}^1 \\ -r_{1G_z}^1 & 0 & r_{1G_x}^1 \\ r_{1G_y}^1 & -r_{1G_x}^1 & 0 \end{bmatrix} \qquad X(r_{2G}^2) = \begin{bmatrix} 0 & r_{2G_z}^2 & 0 \\ -r_{2G_z}^2 & 0 & r_{2G_x}^2 \\ 0 & -r_{2G_x}^2 & 0 \end{bmatrix}$$

$$X(r_{3G}^2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & r_{3G_x}^2 \\ 0 & -r_{3G_x}^2 & 0 \end{bmatrix} \tag{4.71}$$

The cross product transformation from link two to link three $X(r_3^2)$ and the rotational transformations between the links remain unchanged and are given in equation 4.42. The equations of motion are again given by equation 4.34 with elements of the equation described by equations 4.37 through 4.53. Note however, that since a mass has been added to the bond graph to represent the translation of the mass centre of first link the expression for the transformed inertia, equation 4.45, gains three rows. The transformed inertia matrix is now correctly represented as,

$$\tilde{I} = (T_i^I)^T I_n T_i^I \tag{4.72}$$

where

$$T_i^I = \begin{bmatrix} ([0\ 0\ 1]R_1^{1G})^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2 R_2^{2G})^T & ([0\ 0\ 1]R_2^{2G})^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2 R_2^3 R_3^{3G})^T & ([0\ 0\ 1]R_2^3 R_3^{3G})^T & ([0\ 0\ 1]R_3^{3G})^T \\ ([0\ 0\ 1]X(r_{1G}^1)R_1^0)^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2 X(r_{2G}^2)R_2^0)^T & ([0\ 0\ 1](X(r_{2G}^2))R_2^0)^T & [0\ 0\ 0]^T \\ t_{61} & t_{62} & ([0\ 0\ 1]X(r_{3G}^3)R_3^0)^T \end{bmatrix}$$

$$t_{61} = ([0\ 0\ 1]R_1^2 R_2^3 X(r_{3G}^3)R_3^0 + [0\ 0\ 1]R_1^2 X(r_3^2)R_2^3 R_3^0)^T$$

$$t_{62} = ([0\ 0\ 1]R_2^3 X(r_{3G}^3)R_3^0 + [0\ 0\ 1]X(r_3^2)R_2^3 R_3^0)^T \tag{4.73}$$

and

$$I_n = diag\left[J_1^{1G}\ J_2^{2G}\ J_3^{3G}\ M_1\ M_2\ M_3\right] \tag{4.74}$$

Since accurate mass properties or an accurate geometrical model were not available from the robot's manufacturer, the inertial parameters for the first three links are based on a simplified geometric model of the arm. As such they are rough estimates. These model uncertainties will likely prevent good agreement between the simulated and actual energy consumptions for a particular trajectory. However, this approximate model should be sufficient to allow an optimal path planning algorithm to generate significantly better trajectories than nonoptimal path planning methods. Estimates for all inertial parameters are derived in Appendix B and summarized in Table 2.

Table 2 Link Inertial Parameters

	Link		
i	1	2	3
$\vec{r}^{\,i}_{iG,i}$ (m)	$[0\ -0.119\ -0.143]$	$[0.37\ 0\ -0.18]$	$[-0.0314\ 0\ 0]$
m_i (kg)	52.3	18.3	39.9
J_i^{iG} (kgm^2)	$\begin{bmatrix} 3.61 & 0 & 0 \\ 0 & 3.32 & 0 \\ 0 & 0 & 0.60 \end{bmatrix}$	$\begin{bmatrix} 0.231 & 0 & 0 \\ 0 & 0.892 & 0 \\ 0 & 0 & 0.982 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 3.29 & 0 \\ 0 & 0 & 3.69 \end{bmatrix}$

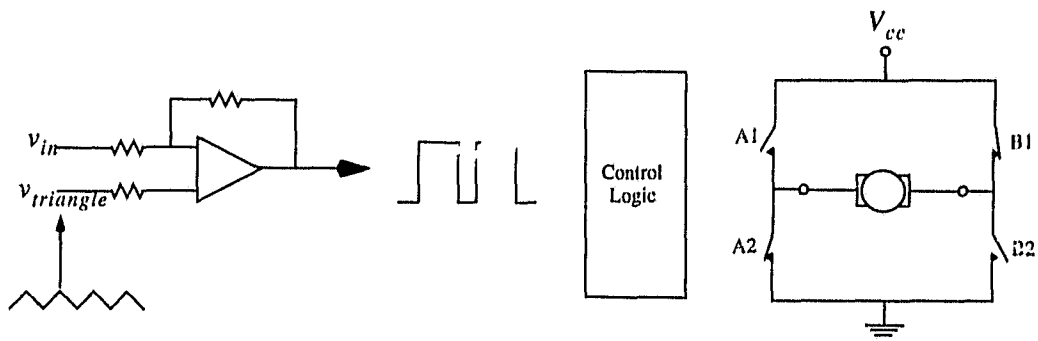


Figure 4.15 Pulse Width Modulated Drive

4.3.2 Actuator Model

As mentioned, the actuators for the Reis V15 manipulator are DC servomotors with harmonic drive gear reducers. A model for a similar drive systems is shown in Figure 4.9 and described in Section 4.2.2. The Reis V15 actuator differs at the amplifier stage. Rather than using a two stage linear amplifier as depicted in Figure 4.9 a pulse-width-modulated (PWM) drive is used. A schematic representation of a PWM drive is shown in Figure 4.15. The input voltage v_{in}, is compared to the voltage from a triangular waveform $v_{triangle}$. If v_{in} is larger than $v_{triangle}$ the input to the control logic is 5 V, otherwise it is 0 V. The control logic circuitry switches the transistors in the H-bridge such

that the A1 / B2 transistor pair are closed and the B1 / A2 transistor pair are open when the input to the control logic is 5 V. When the input to the control logic is 0 V the B1 /A2 pair are closed and the A1 / B2 are open. The net effect is that the voltage across the motor switches between $\pm V_{cc}$ at a duty cycle dependent on the input voltage v_{in} and at the frequency of the triangle waveform. Since the motor has an inherent resistance and inductance (a 5 mH. choke coil is added in series with the motor armature to increase the inductance), the current does not change instantaneously. Rather the motor current is proportional to the duty cycle of the input voltage. There is some current ripple, but if the switching frequency is high enough the effect of current ripple is small. The Reis V15 servo drives use a switching frequency of 9000 Hz., which, although low by current standards, is sufficiently high that the resulting current ripple has no measurable effect on the manipulator performance. To compensate for the effect of the armature back emf on the current in the motor the Reis V15 servo cards sense a voltage proportional to the instantaneous motor current, subtract this voltage from the input voltage and pass the resulting signal through a PID control circuit before comparing with the triangle waveform voltage. This PID control loop in effect makes the input voltage to the PWM amplifier appear as a motor current setpoint.

One of the main advantages of using a PWM drive is that since the transistors in the H-bridge circuit are operating in their switching region (as opposed to their linear region) the voltage drop across the collector emitter circuit and hence the power loss is comparatively small compared to a linear amplifier. Power losses come from three sources in a PWM drive. The first source is commonly referred to as the quiescent or no-load power dissipation. It is the product of the no load current flowing in the bridge and the supply voltage V_{cc}. For the Reis V15 manipulator, V_{cc} is 130 V and the no load current is typically in the region of 15 mA. The second source of power loss is the conductive

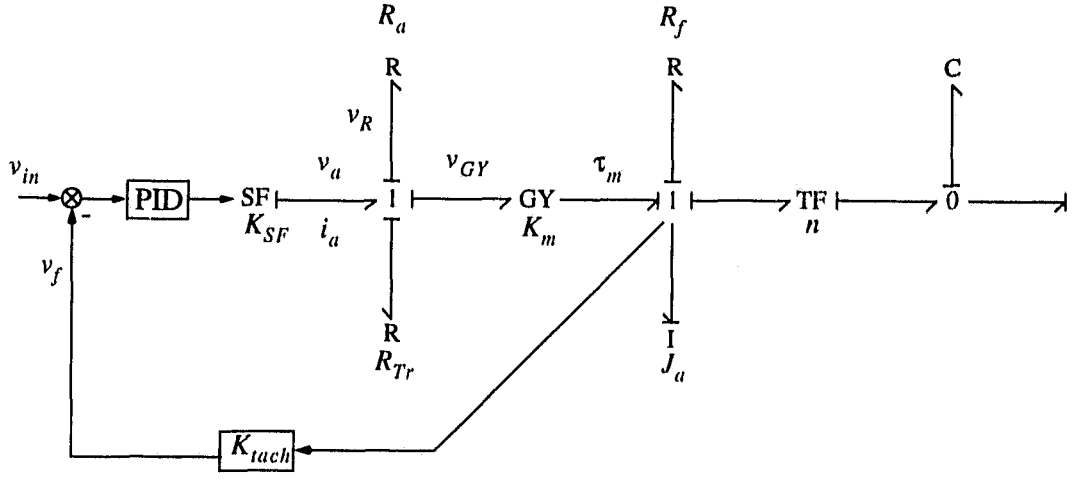


Figure 4.16 Bond Graph Model of the Reis V15 DC Servo Actuator

power losses in the switches (transistors). For the purposes of the conductive power loss the switches appear as resistors of approximately 0.5 Ω. The final source of energy loss is power dissipated during switching. This power loss is a function of how quickly the transistors are able to switch and the operating frequency of the PWM drive. For switching frequencies less than 50 kHz, which is the case for the Reis robot, the conductive dissipation term dominates [65].

From the point of view of energy dissipation, the bond graph model of Figure 4.12 can still be used to model a DC servo even if PWM drive is used. The only difference is that the expression for R_{Tr} changes. Instead of the expression given by equation 4.64, R_{Tr} becomes a constant value of 1 Ω, 0.5 Ω for each transistor conducting at any given time. The armature voltage in the bond graph v_a must be interpreted as the average armature voltage since at any given time the armature voltage when using a PWM drive will be $\pm V_{cc}$. This does not present a problem as far as the calculation of energy consumption is concerned. A bond graph model of the Reis V15's servo actuators is shown in Figure 4.16.

Note that there is an additional feedback signal through gain K_{tach} from the 1-junction corresponding the motor's angular velocity. This tachogenerator voltage is compared to the servo input voltage. The difference is run through an analog PID controller. Therefore the voltage inputs to the Reis V15 servo cards appear as motor angular velocity commands. The output of the PID block controls the level of current in the motor armature. Estimated parameter values for the V15 manipulator's actuators are shown in Table 3.

Table 3 Reis V15 Actuator Parameters

	Joint #		
	1	**2**	**3**
R_a (Ω)	2.0	2.0	2.0
R_{Tr} (Ω)	1.0	1.0	1.0
K_m $(\frac{Nm}{A})$	0.29	0.29	0.29
J_{ai} (kgm^2)	8.72×10^{-4}	8.72×10^{-4}	8.19×10^{-4}
τ_μ (Nm)	0.33	0.33	0.33
K_f (Nms)	2.9×10^{-4}	2.9×10^{-4}	2.9×10^{-4}
n	0.01	0.01	0.01
K_P	25	25	25
K_I	0	0	0
K_D	0	0	0
K_{tach} $(\frac{Vs}{rad})$	0.024	0.024	0.024
K_{DAC} (V)	0.0049	0.0049	0.0049
K_{SF} $(\frac{A}{V})$	4.064	4.064	4.064

The Reis V15 manipulator uses Mavilor MO-800 DC motors to actuate its first three joints. The combined resistance of the armature and choke coil for each joint's motor, as measured across pins ten and eleven of each of the Z1, Z2 and Z3 connectors respectively is $R_a = 2.0\ \Omega$. These connectors are located inside the Reis servo driver cabinet beneath the servo amplifier cards.

The motor constant can be estimated from the motor rating data supplied by the manufacturer. At the rated current of 9.2 A, the motors should supply the rated torque of 2.65 Nm, yielding a motor constant of $K_m = 0.288$ Nm/A. The value was experimentally verified by Pittens [57] where the motor constant for the first joint was found to be 0.293 Nm/A. For the current model the motor constant will be assumed to be $K_m = 0.29$ Nm/A for each of the first three joints of the manipulator.

The armature inertia is given by the motor manufacturer as $7.91 \times 10^{-4}\ kgm^2$. In addition each motor drives the wave generator input of a harmonic drive gear reducer. For the first two joints the manufacturer specifies an input inertia of $8.10 \times 10^{-5}\ kgm^2$. The harmonic drive on the third joint has an input inertia of $2.79 \times 10^{-5}\ kgm^2$. The total estimated rotary inertia for each motor, harmonic drive combination is therefore $J_a = 8.72 \times 10^{-4}\ kgm^2$ for the first two joints and $8.19 \times 10^{-4}\ kgm^2$ for the third joint.

In [57], Pittens, observed that joint friction was predominantly Coulomb. This is confirmed using data from the motor manufacturer. The motor manufacturer lists a damping constant of 0.133 $\frac{in \cdot lb}{1000\ \text{rpm}}$. If this is doubled to account for additional viscous losses in the other components of the transmissions (especially the harmonic drive) one gets a viscous damping coefficient of $K_f = 2.9 \times 10^{-4}\ Nms$. The nominal values of the friction torques for the motors and harmonic drives are reported by the manufacturers to

be 0.398 $in \cdot lb$ and 2.54 $in \cdot lb$ respectively. Combined this gives a static friction torque of $\tau_\mu = 0.33\ Nm$. This will provide a starting estimate for the friction model.

Each harmonic drive provides a speed reduction of $n = 0.01$.

The proportional, integral and derivative gains are determined from a jumper setting and digital readout on each servo amplifier card. For the tests conducted here jumper W15 was in. This disconnects the integral and derivative portion of the analog velocity PID controller on the servo cards, effectively making the integral and derivative gains zero. The digital readout on each of the servo cards gives an integer proportional gain figure of 1500. From the frequency response curve of the PID controller given in annex 1 of the servo amplifiers' manual [32], this corresponds to a proportional gain between 20 and 50. A few trial simulations showed $K_P = 25$ to be a reasonable choice.

The tach gain is a combination of the tachogenerator constant of 10 V/1000 rpm. or 0.0955 Vs/rad and a voltage divider gain. The tach voltage is connected in the feedback path with a voltage divider as shown in Figure 4.17. With the potentiometer tuned as measured and indicated the ratio of the feedback voltage to the tach voltage is,

$$\frac{v_f}{v_{tach}} = \frac{28.3\Omega - 21.2\Omega}{28.3\Omega} = 0.25. \tag{4.75}$$

The overall tachometer gain is therefore,

$$K_{tach} = 0.25 \times 0.0955\ \frac{Vs}{rad} = 0.024\ \frac{Vs}{rad}. \tag{4.76}$$

The input to the servo actuator model of Figure 4.11, v_{in}, is a voltage in the range of $\pm 10\ V$, whereas the output from the digital portion of the controller is an integer number. This integer is converted to a voltage by a twelve bit D/A converter. Nominally, the 20

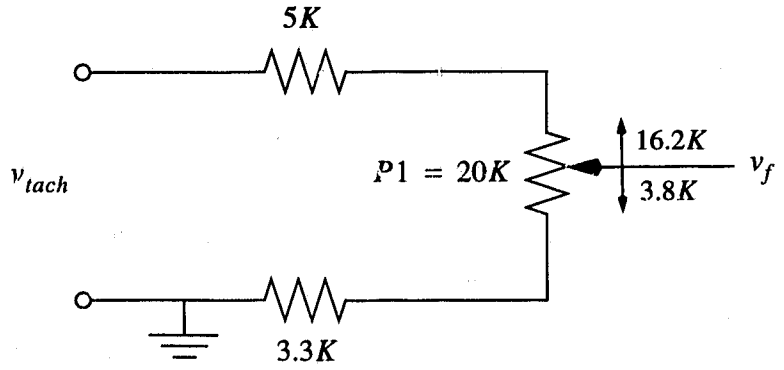


Figure 4.17 Tachogenerator Connection in the Feedback Path

volt range of the input is divided into 4096 discrete values giving an effective gain from digital controller output to servo card input of $K_{DAC} = 0.0049$.

Finally, in [57], Pittens gives a plot relating the armature current to the voltage input to the current control portion of the servo amplifier card. This input voltage corresponds to v_i, the input to the flow source in Figure 4.16. The slope of this linear relationship yields $K_{SF} = 4.064\ A/V$.

The Reis V15's drive system has the added complication that the drives for joints two and three are not kinematically independent. Since the drive motor for joint three is mounted on the first joint and drives the third joint through a transmission which passes through the second link, joints two and three are kinematically coupled. For example, rotating the second joint one full revolution, while holding joint three's motor fixed, would cause joint three to rotate by the gear ratio 'n' times one revolution. This coupling is shown in the bond graph of Figure 4.18 where models for each of the Reis V15's first three joints are shown with the associated kinematic coupling. The full dynamic model of the robot mechanism, actuators and servoamplifiers is formed by connecting the outputs

marked with the joint torques τ_1, τ_2 and τ_3 in Figure 4.18 to the joint inputs of the mechanism model of Figure 4.14.

In the actuator models of Figure 4.18, the compliance (C - element) has been neglected. In [57], Pittens shows that the first resonance frequency from this compliance is approximately 15 Hz. Neglecting this compliance will not have a significant effect on the calculation of energy consumption for this manipulator but will greatly reduce the cost of computing the model since it reduces the number of state variables by half. Removing the compliance elements and associated 0-junctions by themselves creates a problem with preferred causality in the bond graph. It would be impossible to have the actuator model producing the desired output efforts (torques) required as input to the mechanism model and retain inertial causality on the inertial elements J_a. This is because these inertia elements are no longer independent. The causal problem is resolved by transforming these inertias across their respective gear reducers to the mechanism model. The resulting input to the mechanism at the first joint is shown in Figure 4.19. The configuration for the second and third joints are identical. Note that the effective inertia, after transformation across the gear reducer is the armature inertia J_{ai} divided by the square of the gear ratio n^2. This adds three dependent inertias to the mechanism model. As a result the transformation matrix from the velocities associated with the inertia elements and the selected independent velocities gains three rows and the transformed inertia of equation 4.72 becomes,

$$\tilde{I} = (T_i^I)^T I_n T_i^I \tag{4.77}$$

where

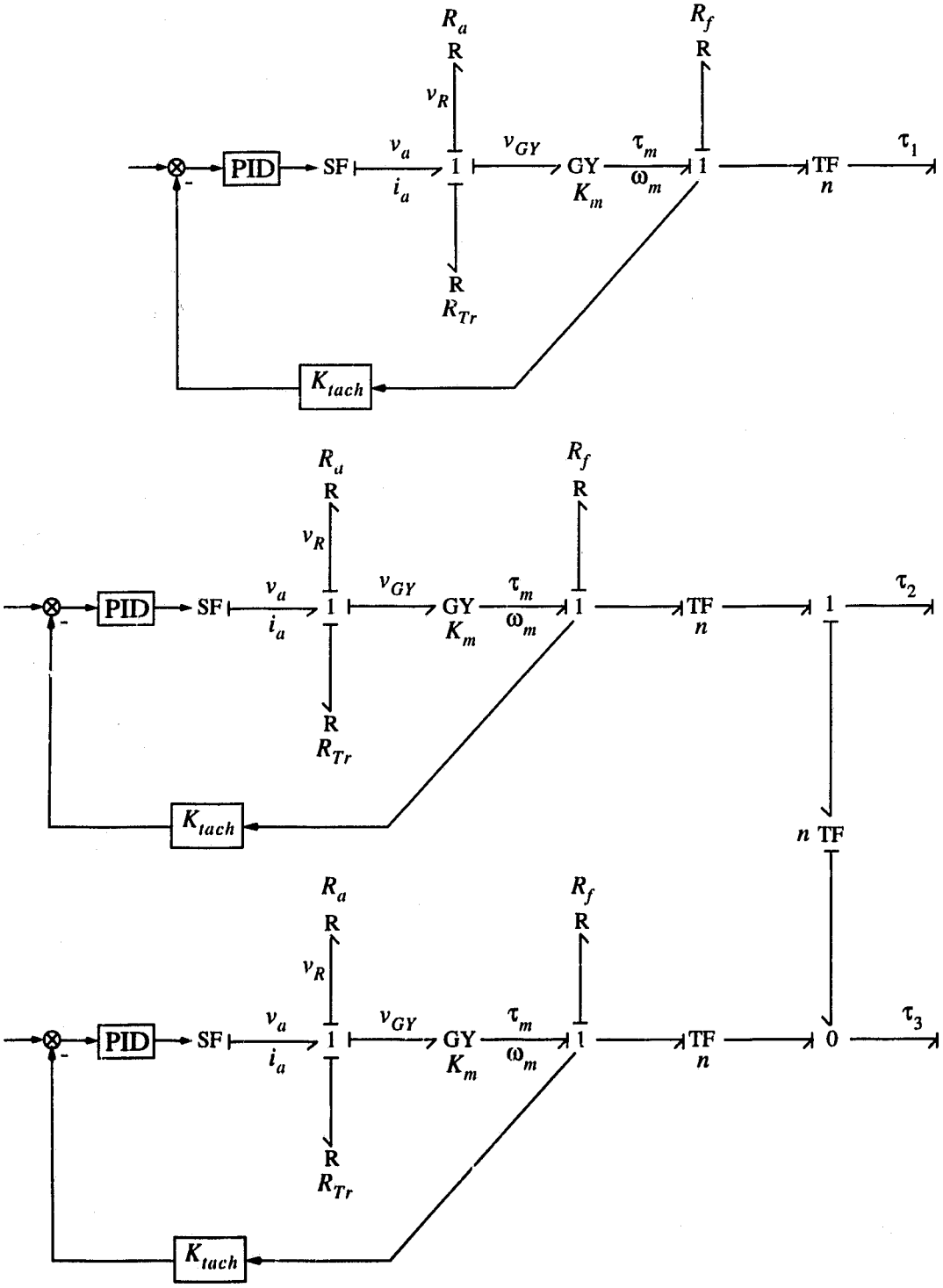


Figure 4.18 Bond Graph Model of the First Three Reis V15 DC Servo Actuators

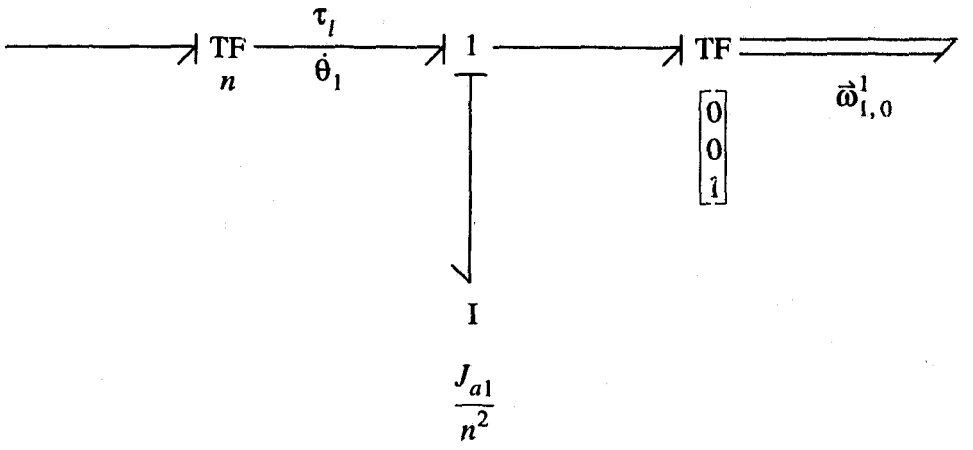


Figure 4.19 First Joint Armature Inertia Transformed Across Gear Reducer

$$T_i^I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ ([0\ 0\ 1]R_1^{1G})^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^{2G})^T & ([0\ 0\ 1]R_2^{2G})^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_2^3R_3^{3G})^T & ([0\ 0\ 1]R_3^{3G})^T \\ ([0\ 0\ 1]X(r_{1G}^1)R_1^0)^T & [0\ 0\ 0]^T & [0\ 0\ 0]^T \\ ([0\ 0\ 1]R_1^2X(r_{2G}^2)R_2^0)^T & ([0\ 0\ 1]X(r_{2G}^2)R_2^0)^T & [0\ 0\ 0]^T \\ t_{61} & t_{62} & ([0\ 0\ 1]X(r_{3G}^3)R_3^0)^T \end{bmatrix}$$

$$t_{61} = ([0\ 0\ 1]R_1^2R_2^3X(r_{3G}^3)R_3^0 + [0\ 0\ 1]R_1^2X(r_3^2)R_2^3R_3^0)^T$$

$$t_{62} = ([0\ 0\ 1]R_2^3X(r_{3G}^3)R_3^0 + [0\ 0\ 1]X(r_3^2)R_2^3R_3^0)^T \qquad (4.78)$$

and

$$I_n = diag\left[J_{a1}\ J_{a2}\ J_{a3}\ J_1^{1G}\ J_2^{2G}\ J_3^{3G}\ M_1\ M_2\ M_3\right]. \tag{4.79}$$

The net effect, since the joint velocities have been selected as the independent velocities, is that the armature inertias are simply added to the first second and third diagonal elements of the transformed inertia matrix $\tilde{I}$ respectively. The structure of the equations of motion for the manipulator remains unchanged.

4.3.3 Control System Model

If a perfect model for the mechanism and actuators is available, the assumption that the manipulator moves along a prescribed trajectory is valid. From this trajectory and the mechanism model, the required joint torques to cause the specified motion could be computed. From the actuator model, the necessary motor currents and the energy consumption required could then be computed. A perfect model is not available. This is why the robot uses a feedback controller to compensate for unmodelled dynamics and disturbances. For the Reis V15 manipulator, part of this controller is embedded in the motor servo cards in the form of an analog velocity feedback loop as shown in Figure 4.16. To control position, the joints angles are fed back and compared to desired values. The difference or error signal is then passed through a PI controller. This part of the control loop is done on a digital computer. The sampling rate used is 100 Hz. The proportional and integral gains are 5000 and 100 respectively, for each of the first three joints. Both the digital and analog portions of the controller have a significant impact on the dynamic behavior of the robot and therefore its energy consumption. The controller must therefore be taken into account when planning optimal trajectories for real manipulators. The approach used with a real manipulator is to treat the specified trajectory as the input setpoint to the manipulator. The controller tracks this setpoint trajectory and the entire system model is

used to estimate the resulting energy consumption. The result of this simulation is then used as the basis for the selection of one trajectory over another in the optimization process.

Finally, real actuator systems exhibit saturation. For the Reis V15, the motor currents may not exceed 18.2 amperes and the outputs of the digital portion of the control law are clipped at 1000. The first constraint is built into the analog hardware of the servo cards and may be mimicked in the computer model by clipping the current values when they exceed the maximum value. This condition is also flagged since running the motors for more than a fraction of a second at this high current level will cause the motors to overheat and subsequently be shut down. Trajectories which exceed this current limit will be deemed inadmissible and will be treated as such by the optimization algorithm. The second constraint is simply part of the digital portion of the control algorithm which may be directly copied into the model.

4.3.4 Special Considerations When Modelling Friction

From all appearances, the friction model of equation 4.63 looks simple. Implementing this type of friction model in a computer simulation requires some special considerations. This is due to the step discontinuity at zero velocity required to model static friction. Consider a mass, subject to a net external force, F_{ext}, which slides on a surface subject to friction. Consider the friction force, F_f, to be made of two components; static friction, F_s, which acts only when the velocity is zero, and dynamic friction, F_d, which is some, possibly nonlinear, function of velocity. The effective force acting on the mass, F_{eff}, is equal to the sum of the external and friction forces.

Dynamic friction always acts in the direction opposite the velocity vector. If the velocity vector happens to be in the opposite direction of the external force vector, F_{ext},

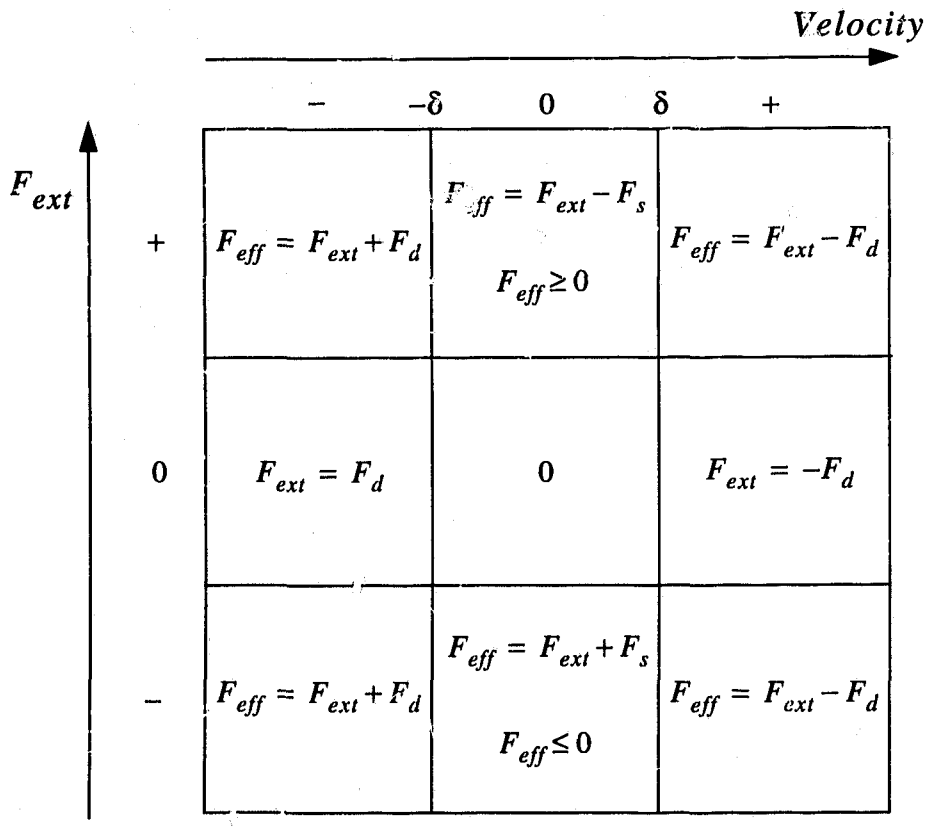


Figure 4.20 Calculation of Effective Force

the magnitude of F_{eff} will actually be the sum of the external and friction forces. Static friction (stiction) always acts in a direction opposite the external force vector. Stiction only occurs when the velocity is zero. Stiction can only null the external force, not cause a reversal in the effective force F_{eff}. As such, its magnitude is equal to the external force up to a prescribed limit.

A chart showing the nine possible conditions under which the effective force is calculated is shown in Figure 4.20. Note from the figure that a prescribed velocity band $\pm\delta$ around the zero velocity has been designated. This band will allow proper switching between static and dynamic friction in the simulation. Since the digital simulation only

evaluates the state of the continuous system model at discrete time intervals, and since the velocity is calculated using machine precision floating point arithmetic, it is practically impossible to arrive at a velocity of exactly zero. The correct choice of the velocity threshold size is essential for an accurate simulation. If the threshold is too small, the simulation will oscillate about the null position. If the threshold is too large, the simulation will stutter step into a stop condition before the velocity drops to an insignificant value. An effective method of estimating the width of the zero velocity band is to calculate the width the band should have in order to ensure that the mass moving with near zero velocity, subject to only the dynamic friction force, would not drift, in one time step, by such an amount that it would be subject to a restoring force. This is done by computing the distance the mass would move under zero friction from standing start with an external force equivalent only to the dynamic friction is applied. The velocity threshold is determined by dividing this distance by the simulation step size.

A final consideration again comes from the nature of digital simulation. Even when the velocity has fallen into the zero velocity threshold region, this small velocity is still being integrated in the simulation to determine position. To prevent the mass from creeping, the velocity must be reset to zero any time it enters the threshold area.

The same arguments apply to the friction model for the Reis V15 actuators except that rotational units such as angular velocity and friction torques are used. The procedure for determining the threshold size does require knowledge of the physics of the situation. For the Reis V15 robot, a threshold size of 0.01 rad/s was estimated from the actuator model.

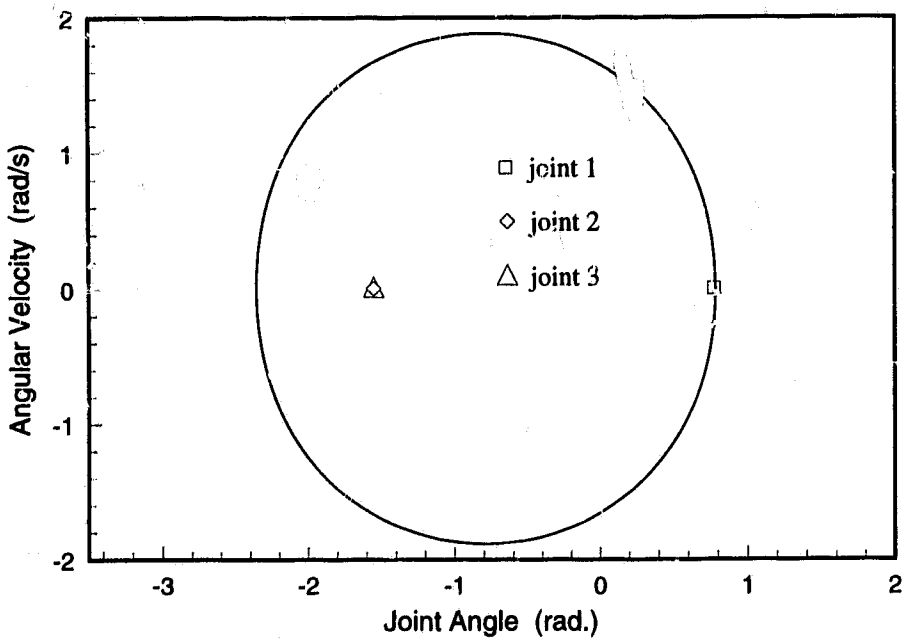


Figure 4.21 Test Trajectory #1

4.3.5 Verification of the Reis V15 Model

To check that the proposed model provides an adequate representation of the Reis V15 robot, both the model and the robot are commanded to follow a set of trajectories. The trajectories of the model and the real manipulator may be in good agreement even if the model is poor. This is because the feedback loops in the controller keep the motion on track despite model errors. The motor current, which is an important factor in determining the energy consumption, is much more sensitive to modelling errors. Therefore, the calculated motor currents from the model are compared to those of the real robot to evaluate the accuracy of the model. Since the robot's inertia is dominated by the motor and harmonic drive inertias and the joint friction model is considered to be the most uncertain part of the overall system model, the three test trajectories selected, shown in Figures 4.21 through 4.23, move only individual joints. The initial and terminal points of the trajectory, are marked by the symbols as indicated in the figure legends.

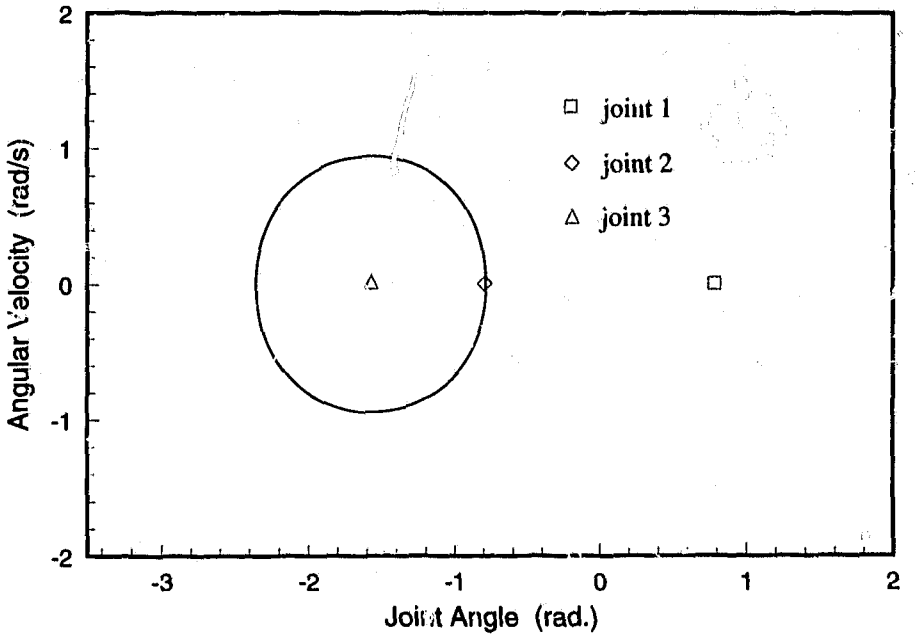


Figure 4.22 Test Trajectory #2

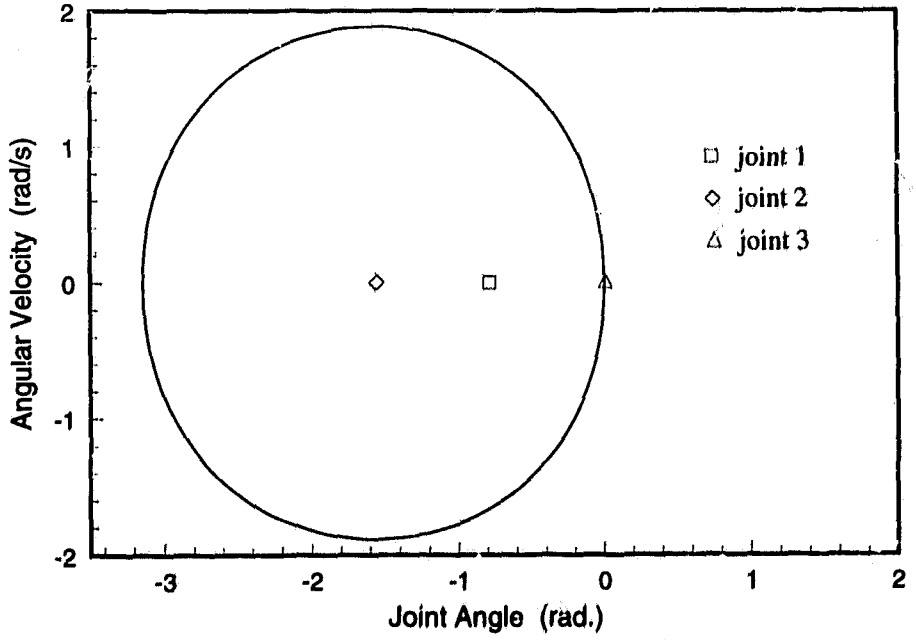


Figure 4.23 Test Trajectory #3

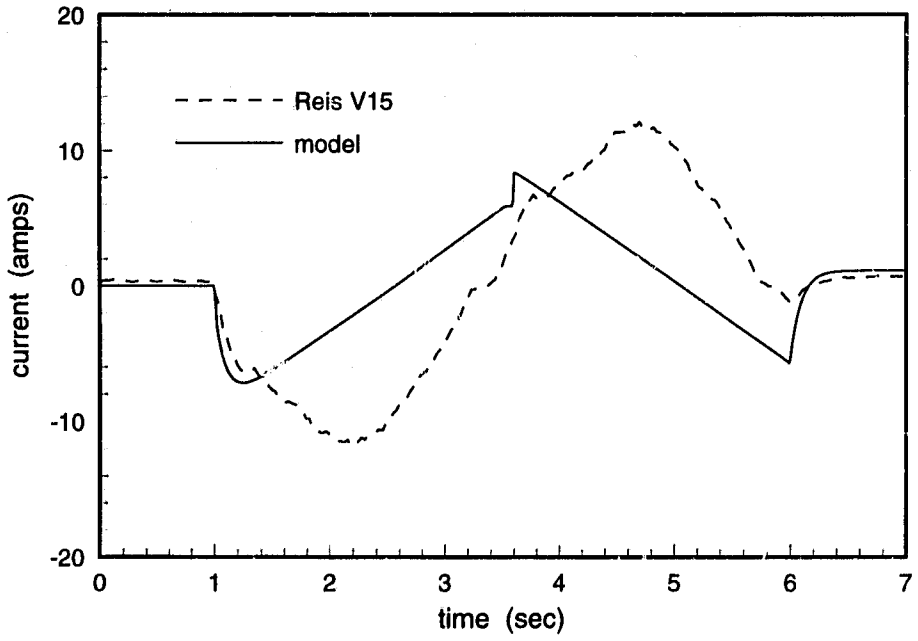


Figure 4.24 Joint 1 Motor Current for Test Trajectory #1

For the first test trajectory, the current drawn by the motor in the first joint, for both the model and the real manipulator, is shown in Figure 4.24. The shapes of the model and Reis V15 curves are similar. Note that the current peaks on the system model correspond to side peaks on the Reis V15 current data. Past these model current peaks the current drawn by the real robot continues to increase. These are regions where the magnitude of the velocity is a maximum. This indicates that viscous friction component has been underestimated.

Figure 4.25 compares the current in the motor of the second joint for both the model and the robot when the robot is asked to follow the second test trajectory. The agreement is good, with again the real robot exhibiting more rounded extrema indicating an underestimation of the viscous friction load.

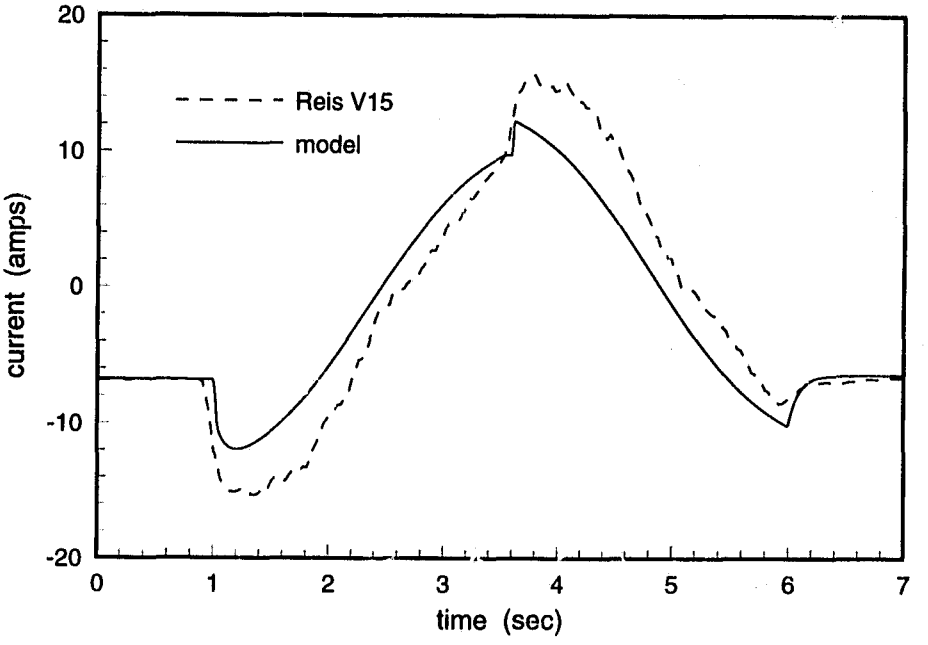


Figure 4.25 Joint 2 Motor Current for Test Trajectory #2

Figure 4.26 shows the current drawn by the motor of the third joint when the robot and model track the third test trajectory. The model errors here are similar to those found in the first and second joints when the manipulator was tracking the first and second test trajectories respectively.

The modelling errors in the motor current draw for the first three joints is, for the most part, due to errors in the friction model, particularly an underestimation of the viscous friction component. The present friction model takes the form of equation 4.63, with a Coulomb friction component and a viscous friction component which depends linearly on the angular velocity of the motor. Testing different parameters in the friction model indicated that a friction term proportional to the square of the velocity term has a significant effect. The square term would represent the effects of windage on the motor armature

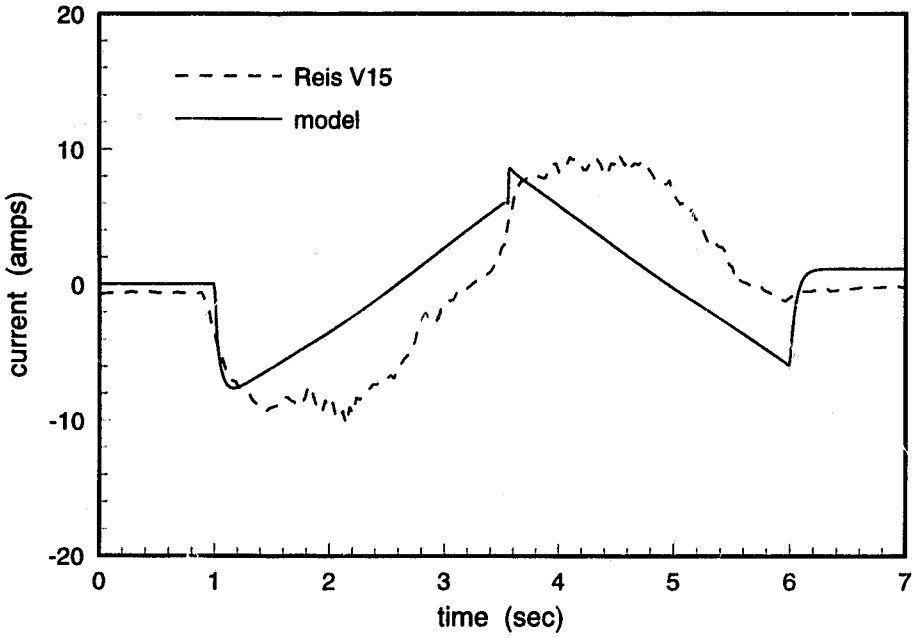


Figure 4.26 Joint 3 Motor Current for Test Trajectory #3

and harmonic drive wave generator and iron losses in the motor. A term proportional to the square of the angular velocity was added to yield a friction model of the following form.

$$\tau_f = \tau_\mu \operatorname{sgn}\dot{\theta} + K_f\dot{\theta} + K_{f2}\dot{\theta}^2 \tag{4.80}$$

After a few sets of parameters were tried the friction model parameters of Table 4 were settled upon. With the revised model, the plots of Figures 4.24 through 4.26 are repeated

Table 4 Reis V15 Friction Model Parameters

	Joint #		
	1	**2**	**3**
τ_μ (Nm)	0.33	0.80	0.33
K_f (Nms)	0.001	0.007	0.0008
K_{f2} Nms^2	0.00008	0.0001	0.00005

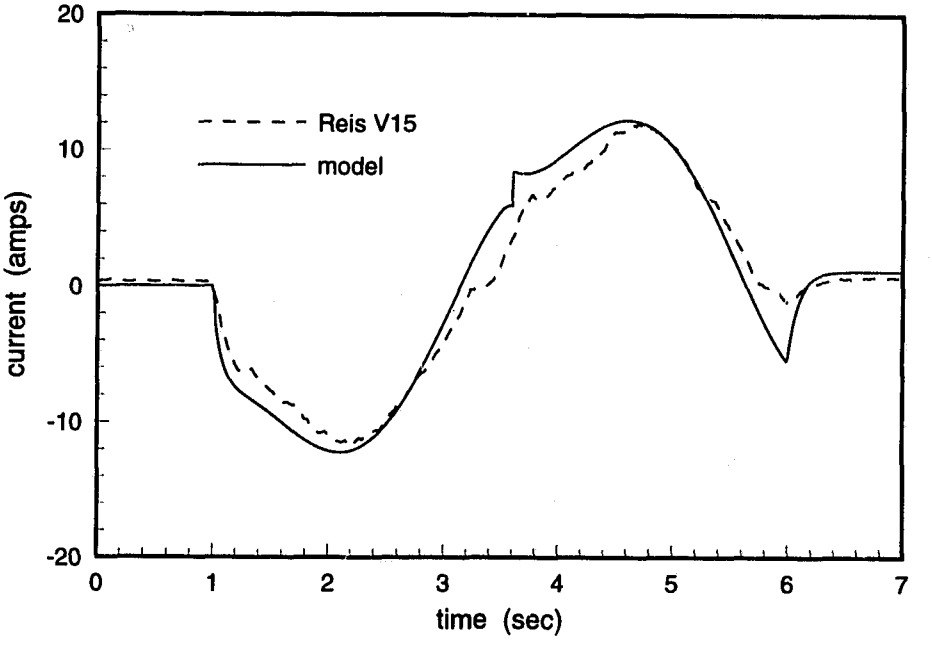


Figure 4.27 Joint 1 Motor Current for Test Trajectory #1 - Improved Model

in Figures 4.27 through 4.29. These figures show a dramatic improvement in the accuracy with which the model current approximates the collected data. The model continues to exhibit the pronounced current peak on the first and third joints as the trajectory reaches its terminal point. This difference is again due to the friction model. For the real robot the friction is likely significantly higher than the dynamic Coulomb friction when a joint is stationary and then drops sharply as the joint begins to move. The model assumes that the level of static friction is equivalent to the Coulomb friction. This can be adjusted in the computer model but is an unnecessary complication since it will have only a small effect on the calculation of the overall energy consumption for the trajectory. If the model were being used as part of a model-based control law, this effect would be harder to neglect.

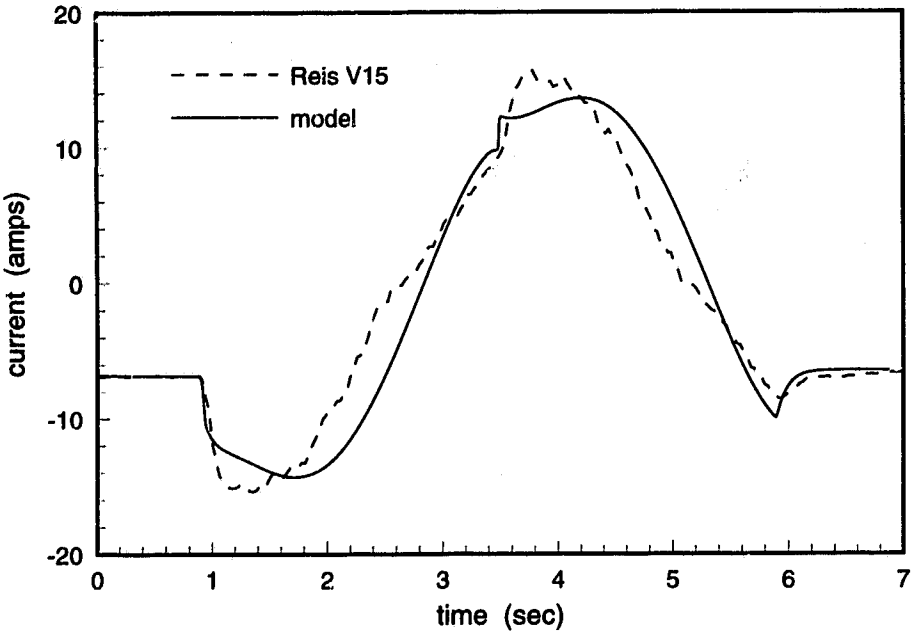


Figure 4.28 Joint 2 Motor Current for Test Trajectory #2 - Improved Model

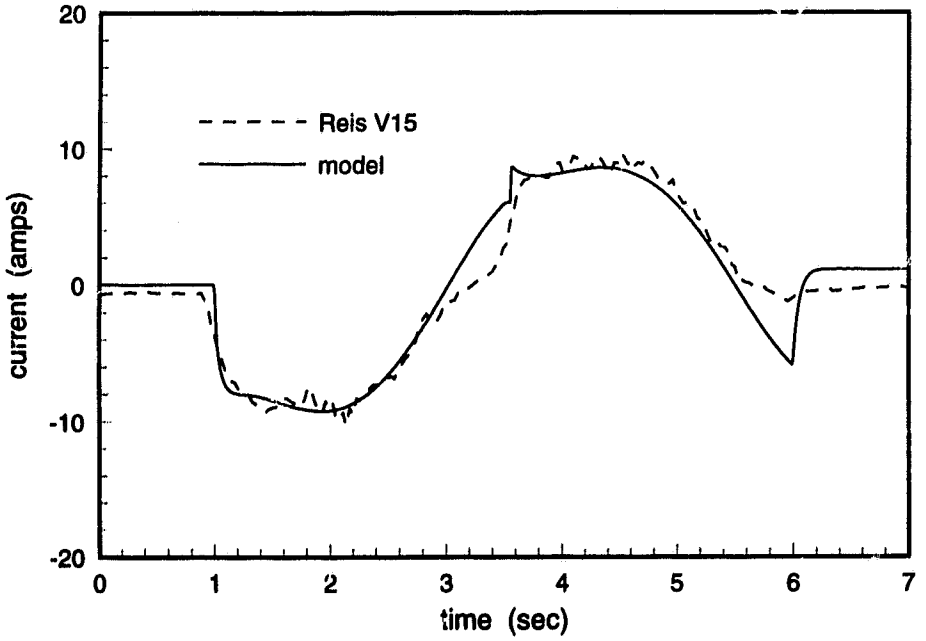


Figure 4.29 Joint 3 Motor Current for Test Trajectory #3 - Improved Model

Chapter 5
Optimization Approaches

A trajectory for a robotic manipulator between given initial and final states is sought such that a performance index is minimized. As stated, this is a functional optimization problem. One function f^* (the optimal trajectory), from a space S of functions f (candidate trajectories), is sought, such that a functional J (performance index) is minimized. The solution to such a problem can be formulated by using the classical variational approach of the maximum principle. However, as is pointed out in Chapter 3, previous researchers have achieved little success with this approach due to the ill-conditioned nature of the two point boundary value problem which arises from this formulation. If the candidate trajectories can be represented by a finite number of parameters then this optimization problem can be transformed into a function minimization and the broad body of methods for the optimization of functions can be brought to bear.

One method of creating a parameterized trajectory is to choose a fixed number of points through which the trajectory must pass and then use some algorithm to interpolate between these points. A function optimization routine may then be applied over the multivariable space spanned by the interpolation points. There are many interpolation functions

which may be selected, including polynomials and rational functions of various order. The choice of interpolation function depends on the situation. One should select a function which is as simple as possible but still reflects the physics of the problem. Selecting a method for parameterizing a trajectory sets the class of functions over which an optimization routine will search. Different parameterization methods will produce better or worse results depending on how closely the selected function class can represent the true optimal trajectories. A second consideration, which must be dealt with in concert with the selection of an interpolating function, is which representation of the manipulator trajectory should be parameterized. These options are discussed next. The selection of an interpolating function is left until Section 5.1.2.

5.1 Trajectory Representations

Three common representations of manipulator motion are, configuration space, phase space and a time history of each joint angle.

5.1.1 Configuration Space

Configuration space contains n-tuples which represent the 'n' joint angles for an 'n' degree of freedom manipulator. A curve in configuration space represents a manipulator "path". A path is distinguished from a manipulator trajectory by the fact that it contains no specific time information. At any point on the path, the joint angles, or configuration of the manipulator is described, but the time at which that configuration occurs relative to other configurations on the path is not specified. Clearly a configuration space representation by itself is insufficient to plan optimal trajectories. Configuration space representations can be used as part of a two stage optimization process whereby an inner loop suboptimization process finds the optimum trajectory given a fixed path, and an outer loop

optimization process varies the path to find the overall optimum trajectory. As mentioned in Chapter 3, this is the approach of Rajan [62] to determine minimum time trajectories. This approach could also be used for the minimum energy problem where the methods given in [85][76][78] would be candidate methods for the inner loop suboptimization process. To choose this representation over a representation capable of fully describing the trajectory there would have to be some benefit to solving the optimization problem in two stages. In reality, the outer loop problem of determining the best path in configuration space is of the same order of difficulty as determining the optimal trajectory in one stage from a representation capable of describing the complete trajectory.

5.1.2 Time History of Joint Angles

Trajectories may be represented by describing the time histories of each joint angle. Hereafter, this will be referred to as the time domain representation. For a robotic manipulator, it is necessary for the interpolating functions to have at least a continuous first derivative since discontinuities in joint velocity would imply the ability of the motors to apply an infinite torque. The simplest interpolation function which guarantees a continuous first derivative is the piecewise cubic spline.

C^1 Piecewise Cubic Spline Interpolation

Given a time series of data points, $\theta_j(t_1)\ldots\theta_j(t_n)$, for each joint j such that,

$$t_1 < t_2 < \ldots < t_n \tag{5.1}$$

construct a piecewise cubic curve which passes through each of the data points. Each pair of data points is connected by a distinct cubic polynomial curve segment. These curve segments may be expressed on each interval t_i to t_{i+1} by substituting a local parameterization variable,

$$s = \frac{t - t_i}{t_{i+1} - t_i}. \tag{5.2}$$

Clearly s varies between zero and one over each interval. Using the local variable s, the curve segment is represented by the cubic polynomial,

$$Y_i(s) = a_i + b_i s + c_i s^2 + d_i s^3 \tag{5.3}$$

with

$$Y_i(0) = \theta_i = a_i \tag{5.4}$$

and

$$Y_i(1) = \theta_{i+1} = a_i + b_i + c_i + d_i. \tag{5.5}$$

Since there are four coefficients to determine, two more equations are required. These equations are provided by specifying the derivative of the curve at each knot point t_i. Therefore,

$$Y'_i(0) = D_i = b_i \tag{5.6}$$

and

$$Y'_i(1) = D_{i+1} = b_i + 2c_i + 3d_i. \tag{5.7}$$

Equations 5.4 through 5.7 are solved to yield equations for the four coefficients.

$$
\begin{aligned}
a_i &= \theta_i \\
b_i &= D_i \\
c_i &= 3(\theta_{i+1} - \theta_i) - 2D_i - D_{i+1} \\
d_i &= 2(\theta_i - \theta_{i+1}) + D_i + D_{i+1}
\end{aligned}
\tag{5.8}
$$

Since D_i is the derivative at the left end of the ith interval and at the right end of the $(i-1)\,th$ interval the resulting spline curve will have C^1 continuity. In practice, robotic actuators are not capable of providing instantaneous changes in the applied joint torques or forces. Even with high bandwidth actuators which can change their outputs nearly instantaneously (from the point of view of the lower bandwidth mechanism), allowing discontinuities in the joint torques or forces can lead to excessive wear of components. This problem can be addressed by requiring the piecewise cubic spline curve to have a continuous second derivative. Adding this requirement removes the ability to arbitrarily choose the trajectory derivatives at the interpolation points. This new class of trajectories through which an optimization algorithm will search is a subclass of the one just described. While it does not contain all of the trajectories of the original class, it does have the benefit that the dimension of the space the optimization routine must work with will be halved.

C^2 Piecewise Cubic Spline Interpolation

To ensure C^2 continuity, at each internal joint D_i is chosen such that

$$
Y''_{i-1}(1) = Y''_i(0) \tag{5.9}
$$

or

$$
2c_{i-1} + 6d_{i-1} = 2c_i \tag{5.10}
$$

Substituting from equation 5.8, and simplifying gives,

$$D_{i-1} + 4D_i + D_{1+1} = 3(\theta_{i+1} - \theta_{i-1}) . \quad (5.11)$$

There are $n-2$ such equations provided by the $n-2$ internal knot points. There are however, n unknowns $(D_1 \ldots D_n)$. The two additional equations which are required come from the end conditions. The first choice is to simply specify the endpoint derivatives directly. A second common choice, the natural spline, requires that the second derivative be zero at the endpoints. This implies,

$$2D_1 + D_2 = 3(\theta_2 - \theta_1) \quad (5.12)$$

at the initial point and

$$D_{n-1} + 2D_n = 3(\theta_n - \theta_{n-1}) \quad (5.13)$$

at the final point. Note that the set of equations 5.11 along with the boundary equations such as 5.12 and 5.13 form a linear, tridiagonal set of equations. An algorithm used to solve these equations should make use of their special tridiagonal form. Since, for the present problem, the initial and terminal states of each joint are given in advance, the end data points and their associated derivatives are fixed and known.

Piecewise Quartic Spline Interpolation

It is possible to ensure C^2 continuity without giving up the freedom to choose the interpolation point derivatives by using piecewise quartic spline interpolation. Here the equation representing the trajectory between adjacent interpolation point is given by the quartic polynomial

$$Y_i(s) = a_i + b_i s + c_i s^2 + d_i s^3 + e_i s^4 \quad (5.14)$$

where, as before s is the parametric variable over the interval given by equation 5.2. The interpolation points and their derivatives require,

$$Y_i(0) = \theta_i = a_i \tag{5.15}$$

$$Y_i(1) = \theta_{i+1} = a_i + b_i + c_i + d_i + e_i \tag{5.16}$$

$$Y_i'(0) = \dot{\theta}_i = b_i \tag{5.17}$$

and

$$Y_i'(1) = \dot{\theta}_{i+1} = b_i + 2c_i + 3d_i + 4e_i. \tag{5.18}$$

Forcing second derivative continuity gives,

$$Y_{i-1}''(1) = Y_i''(0)$$

$$2c_{i-1} + 6d_{i-1} + 12e_{i-1} = 2c_i \tag{5.19}$$

Solving equations 5.15 through 5.19 yields the equations for the five coefficients in each interval.

$$\begin{aligned} a_i &= \theta_i \\ b_i &= \dot{\theta}_i \\ c_i &= c_{i-1} + 3d_{i-1} + 6e_{i-1} \\ d_i &= 4\theta_{i+1} - \dot{\theta}_{i+1} - 4a_i - 3b_i - 2c_i \\ e_i &= -3\theta_{i+1} + \dot{\theta}_{i+1} + 3a_i + 2b_i + c_i \end{aligned} \tag{5.20}$$

Note that starting values for the coefficients c_{i-1}, d_{i-1} and e_{i-1} for the first interpolation interval, $i = 1$ must be chosen. The fact that this choice exists indicates that only a cubic

polynomial is required in the first interval to meet the condition of second order continuity while maintaining the freedom to set the derivative at the interpolation points. This information can be used to select these coefficients to provide this cubic interpolation function. First, choose d_0 and e_0 equal to zero. Choose

$$c_0 = 3(\theta_2 - \theta_1) - 2\dot{\theta}_1 - \dot{\theta}_2. \tag{5.21}$$

Then, from equation 5.20, the coefficient of s^4, e_1, is zero. Therefore the interpolation function in the first interval is cubic.

Time Scaling

Unless the special case optimal path planning problem in which the optimal trajectory must be completed at time t_n is being considered, the time taken for the optimal trajectory is not known and a value for this terminal knot point can't be specified. The solution to this problem is to represent the data points in terms of a dummy variable,

$$\tau = \frac{1}{\omega} t. \tag{5.22}$$

This adds one additional parameter to the optimization, namely ω. If the terminal knot point τ_n is set equal to unity, ω becomes the trajectory time. For parameterization methods which use derivative information at the interpolation points, these derivatives must be adjusted for the time scaling factor. The appropriate derivatives to be used for the time scaled spline for each joint are,

$$D_i = \omega \dot{\theta}_j(\tau_i). \tag{5.23}$$

The optimization may then proceed by searching through the space spanned by the interior data points for each joint's trajectory and the time-scaling factor ω.

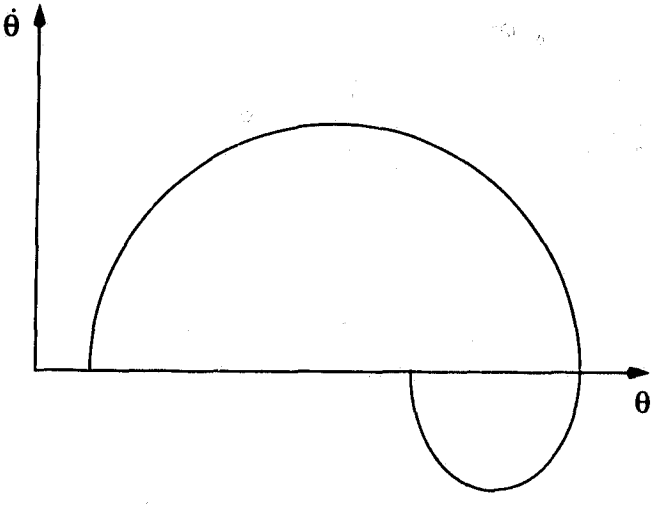


Figure 5.1 State Space Trajectory for a Single Joint

5.1.3 Phase Space

The phase space representation consists of joint velocity versus joint angle interpolation curves for each joint. While this representation completely describes the manipulator trajectory, it has some characteristics which must be taken into account when constructing the interpolation curves. First, it is possible that the curve representing the trajectory wraps back on itself as shown in Figure 5.1. When $\dot{\theta}$ is positive, the joint angle must be increasing. That is to say the trajectory must be moving from left to right. When $\dot{\theta}$ is negative, the joint angle must be decreasing or the trajectory must be moving from right to left. Where the trajectory intersects the $\dot{\theta} = 0$ axis, it must do so perpendicular to this axis. Since curves in this space may wrap back on themselves a joint's trajectory can not be represented by a function of the form

$$\dot{\theta} = f(\theta) \tag{5.24}$$

over its entire length. It may either be described in a piecewise form or in parametric form

$$\theta = \theta(s)$$
$$\dot{\theta} = \dot{\theta}(s) \qquad (5.25)$$

where s is a parametric variable.

Uniform Cubic B-spline Representation

A convenient way of representing a joint's phase space trajectory so that it meets the above criteria is to use uniform cubic B-splines. Uniform cubic B-splines use parametric cubic polynomials on a uniform knot sequence, specifically that of a sequence of successive integers, to construct a curve that approximates a set of data points. This representation allows local control in that shifting a single data point causes only a portion of the curve to change. It is important to note that this is an approximation technique. As such, the curve does not, in general, interpolate or pass through the data points. This posses no problem, since for a given joints trajectory it is not necessary that all of the data points be interpolated. Rather, it is only necessary that moving the data or control points effects the shape of the curve. The end points of the trajectory must, however, be interpolated and intersections with the $\dot{\theta} = 0$ axis must be perpendicular to this axis. The means by which these goals may be accomplished will be discussed after a short development of the basic uniform cubic B-spline representation.

Consider that a joint's trajectory is assembled from parametric cubic polynomials $\theta(s)$ and $\dot{\theta}(s)$ that have C^2 continuity. The parametric variable s assumes an integer value, s_i, of zero through m at each successive data point, where there are m+1 data points. This group of integer points is often referred to as the knot vector and each member of the group is called a knot point. Development is restricted to the single coordinate $\theta(s)$ since the development for $\dot{\theta}(s)$ is identical. C^2 continuity implies the following conditions on the θ coordinates at the joints between the curve segments.

$$\theta_{i-1}(s_i) = \theta_i(s_i)$$

$$\theta'_{i-1}(s_i) = \theta'_i(s_i)$$

$$\theta''_{i-1}(s_i) = \theta''_{i-1}(s_i) \tag{5.26}$$

where

$$\theta'(s) = \frac{d\theta}{ds} \qquad \theta''(s) = \frac{d^2\theta}{ds} \tag{5.27}$$

This continuity may be achieved if $\theta(s)$ and $\dot{\theta}(s)$ are both defined to be piecewise cubic polynomials with integer knot points s_i, since a linear combination of such basis functions will also be a C^2 continuous piecewise cubic polynomial. Local control is maintained if the majority of the polynomial segments defining the basis function are zero. The cubic basis functions are assumed to be nonzero over four consecutive parametric intervals each of unit length. For each interval the basis function is defined by the cubic polynomial,

$$(a_j + b_j\bar{s} + c_j\bar{s}^2 + d_j\bar{s}^3) \quad ; \quad (i-3) \le j \le 1, \tag{5.28}$$

where $\bar{s}$ is a local variable on each interval defined as,

$$\bar{s} = \frac{s - s_i}{s_{i+1} - s_i}. \tag{5.29}$$

The cubic basis function $B_i(s)$ therefore consists of four polynomial segments $b_{-0}(s)\ldots b_{-3}(s)$, each with four as yet undetermined coefficients. By definition $B_i(s)$ is zero outside of the parametric range s_i to s_{i+4}. As a result, its first and second derivatives are zero outside this range. The conditions for C^2 continuity require that,

$$
\begin{array}{lll}
0 = b_{-0}(0) & 0 = b'_{-0}(0) & 0 = b''_{-0}(0) \\
b_{-0}(1) = b_{-1}(0) & b'_{-0}(1) = b'_{-1}(0) & b''_{-0}(1) = b''_{-1}(0) \\
b_{-1}(1) = b_{-2}(0) & b'_{-1}(1) = b'_{-2}(0) & b''_{-1}(1) = b''_{-2}(0) \\
b_{-2}(1) = b_{-3}(0) & b'_{-2}(1) = b'_{-3}(0) & b''_{-2}(1) = b''_{-3}(0) \\
b_{-3}(1) = 0 & b'_{-3}(1) = 0 & b''_{-3}(1) = 0
\end{array} \tag{5.30}
$$

These equations constitute fifteen constraints for the sixteen unknown polynomial coefficients. One additional equation is required. It is convenient to require that,

$$b_{-0}(0) + b_{-1}(0) + b_{-2}(0) + b_{-3}(0) = 1 \tag{5.31}$$

Since equally spaced knots have been chosen, equations 5.30 and 5.31 amount to adding an unscaled sequence of basis functions B_i each of which is identical but shifted so that the parameter values for which it is nonzero (its support) begins at s_i. From equation 5.31, the three basis functions B_{j-3}, B_{j-2} and B_{j-1} that are nonzero at s_j sum to one. This normalizing condition uniquely defines the basis functions B_i. It turns out that the normalizing condition holds at all values of s, not just the knot points s_i. That is,

$$b_{-0}(\bar{s}) + b_{-1}(\bar{s}) + b_{-2}(\bar{s}) + b_{-3}(\bar{s}) = 1 \qquad \forall \ 0 \le \bar{s} \le 1 \tag{5.32}$$

Solving equations 5.30 and 5.31 for the sixteen unknown coefficients of the segments yields the following segment polynomial equations.

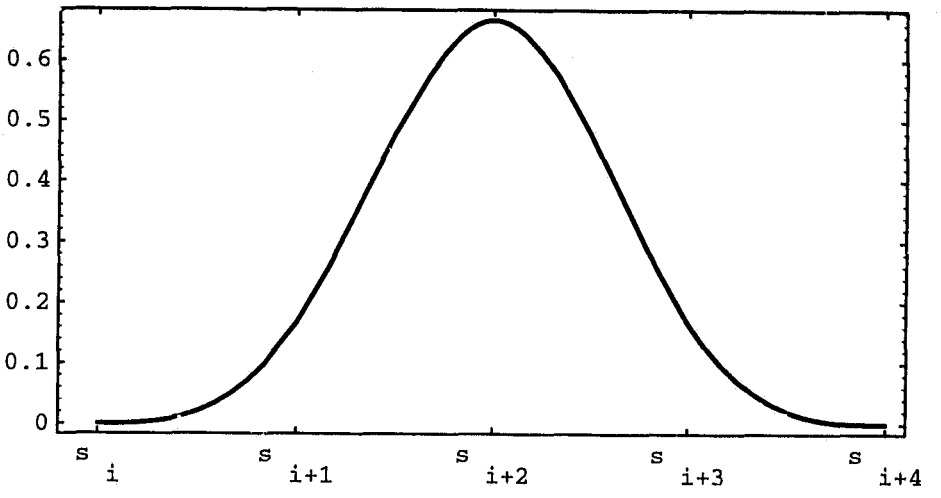


Figure 5.2 The Uniform Cubic B-spline Basis Function

$$
\begin{aligned}
b_{-0}(\bar{s}) &= \frac{1}{6}\bar{s}^3 \\
b_{-1}(\bar{s}) &= \frac{1}{6}(1+3\bar{s}+3\bar{s}^2-3\bar{s}^3) \\
b_{-2}(\bar{s}) &= \frac{1}{6}(4-6\bar{s}^2+3\bar{s}^3) \\
b_{-3}(\bar{s}) &= \frac{1}{6}(1-3\bar{s}+3\bar{s}^2-\bar{s}^3)
\end{aligned}
\tag{5.33}
$$

These four segments define the uniform cubic B-spline shown in Figure 5.2. Enough of these basis functions can represent any C^2 spline over a uniform knot sequence. The spline curve is determined by choosing the data points as control vertices V_i and defining the curve,

$$
Q(s) = \sum_i V_i B_i(s) = \sum_i (\theta_i B_i(s), \dot{\theta}_i B_i(s)) . \tag{5.34}
$$

Since the basis functions are nonzero on only four successive intervals, the curve over any one segment is described by the summation of only four basis functions;

$$Q_i(s) = \sum_{r=-3}^{0} V_{i+r} B_{i+r}(s). \tag{5.35}$$

Replacing each basis function $B_j(s)$ by that segment which pertains to the interval $[s_i, s_{i+1})$ yields,

$$Q_i(s) = V_{i-3} b_{-3}(\bar{s}) + V_{i-2} b_{-2}(\bar{s}) + V_{i-1} b_{-1}(\bar{s}) + V_{i-0} b_{-0}(\bar{s}). \tag{5.36}$$

Recall that the spline representation must interpolate the endpoints of the trajectory and intersect the $\dot{\theta} = 0$ axis perpendicular to the axis. To investigate endpoint interpolation, consider the end conditions for a uniform cubic B-spline curve with vertices V_0 to V_m. Curve segments may only be defined where there are four control vertices which have an effect over the interval in question. Therefore, the curve is first defined on the fourth interval $[s_3, s_4)$. Similarly, the last interval over which the curve is defined is $[s_m, s_{m+1})$. One choice is to define segments in the first three knot intervals by eliminating terms from equation 5.36 which do knot have a control vertex. A preferred approach, is to force the endpoints of the curve to have particular properties.

If no end conditions are specified, the start and end points, P_s and P_e, respectively, of the curve may be determined by evaluating equation 5.36 at $\bar{s} = 0$ for vertices V_0 to V_3 and at $\bar{s} = 1$ for vertices V_{m-3} to V_m. This gives,

$$\begin{aligned} P_s &= Q_3(0) = \frac{1}{6}(V_0 + 4V_1 + V_2) \\ P_e &= Q_m(1) = \frac{1}{6}(V_{m-2} + 4V_{m-1} + V_m) \end{aligned} \tag{5.37}$$

In order to control the positions, as well as the first derivative and curvature at the end points one of two techniques may be applied. First the vertex sequence may be extended

by repeating the end vertices V_0 and V_m some number of times. Alternatively, additional vertices, called phantom vertices V_{-1} and V_{m+1}, may be computed at either end so that P_s and P_e, or perhaps the first or second derivative vectors satisfy the specified condition at the end points. First consider the consequences of repeating the end vertices.

If the end vertices are doubled up, that is, the sequence of $m+3$ vertices $V_0, V_0, V_1, \ldots, V_{m-1}, V_m, V_m$ are used to compute an m-segment curve, the new segments added to the curve are given by,

$$Q_2(\bar{s}) = V_0[b_{-3}(\bar{s}) + b_{-2}(\bar{s})] + V_1 b_{-1}(\bar{s}) + V_2 b_{-0}(\bar{s})$$

$$Q_{m+1}(\bar{s}) = V_{m-2} b_{-3}(\bar{s}) + V_{m-1} b_{-2}(\bar{s}) + V_m[b_{-1}(\bar{s}) + b_{-0}(\bar{s})] \quad \textbf{(5.38)}$$

Evaluating the curves at $\bar{s} = 0$ and $\bar{s} = 1$ to respectively obtain the first and last points of the curve gives,

$$P_s = Q_2(0) = \frac{5}{6}V_0 + \frac{1}{6}V_1$$

$$P_e = Q_{m+1}(1) = \frac{1}{6}V_{m-1} + \frac{5}{6}V_m \quad \textbf{(5.39)}$$

Therefore, the curve begins and ends at a point one-sixth of the way from V_0 to V_1 and ends at a point one-sixth of the way from V_m to V_{m-1}. The derivative vectors at P_s and P_e are,

$$Q'_2(0) = \frac{1}{2}(V_1 - V_0)$$

$$Q'_{m+1}(1) = \frac{1}{2}(V_m - V_{m-1}) \quad \textbf{(5.40)}$$

Therefore the tangent of the curve at its endpoints is along a line joining the first and last two control vertices. Computing the second derivative at the endpoints shows that the curvature at these points is zero.

If the first and last vertices are tripled, so that the curve is computed on the $m+5$ vertices $V_0, V_0, V_0, V_1, \ldots, V_{m-1}, V_m, V_m, V_m$ four additional curve segments are added; two at the beginning and two at the end.

$$
\begin{aligned}
Q_1(\bar{s}) &= V_0\,[b_{-3}(\bar{s}) + b_{-2}(\bar{s}) + b_{-1}(\bar{s})] + V_1 b_{-0}(\bar{s}) \\
Q_2(\bar{s}) &= V_0\,[b_{-3}(\bar{s}) + b_{-2}(\bar{s})\,] + V_1 b_{-1}(\bar{s}) + V_2 b_{-0}(\bar{s}) \\
Q_{m+1}(\bar{s}) &= V_{m-2} b_{-3}(\bar{s}) + V_{m-1} b_{-2}(\bar{s}) + V_m\,[b_{-1}(\bar{s}) + b_{-0}(\bar{s})] \\
Q_{m+2}(\bar{s}) &= V_{m-1} b_{-3}(\bar{s}) + V_m\,[b_{-2}(\bar{s}) + b_{-1}(\bar{s}) + b_{-0}(\bar{s})]
\end{aligned}
\qquad \textbf{(5.41)}
$$

Evaluating the endpoints of the curve gives,

$$
\begin{aligned}
P_s &= Q_1(0) = V_0 \\
P_e &= Q_{m+2}(1) = V_m .
\end{aligned}
\qquad \textbf{(5.42)}
$$

Therefore the curve interpolates the first and last points. The equations for the first and last segments are given by,

$$
\begin{aligned}
Q_1(\bar{s}) &= (1 - \frac{\bar{s}^3}{6})\, V_0 + \frac{\bar{s}^3}{6} V_1 \\
Q_{m+2}(\bar{s}) &= (\frac{1-\bar{s}^3}{6})\, V_{m-1} + (1 - \frac{1-\bar{s}^3}{6})\, V_m .
\end{aligned}
\qquad \textbf{(5.43)}
$$

Note that these are straight lines. The second segment $Q_2(\bar{s})$ and penultimate segment $Q_{m+1}(\bar{s})$ begin and end respectively with double vertices. They therefore exhibit the behavior associated with double vertices.

From the above analysis it is clear that the endpoints of the trajectory may be interpolated by making them triple vertices. Double and triple vertices may also be used in the interior of a B-spline curve. The analysis applied above is directly applicable to these cases as well. Multiple vertices may be used to achieve the second goal of having the trajectory intersect the $\dot{\vartheta} = 0$ axis at right angles. The first situations to consider are those where the start or end point of the trajectory lies on the $\dot{\theta} = 0$ axis. Here triple vertices are used to ensure that the endpoints are interpolated. From equation 5.43 if an additional vertex and knot point were added with the same angular value θ, but different angular velocity $\dot{\theta}$, immediately after or preceding the initial or terminal points respectively, the initial and terminal curve segments would be vertical straight lines thereby forcing a perpendicular intersection. The length of the straight line segment would be one-sixth of the distance from the respective endpoint to the added vertex. A similar configuration, shown in Figure 5.3, could be used for a crossing point. Here the crossing point is interpolated with a triple vertex and an additional vertex is added to either side of the crossing point. This would create a straight line segment from a point one-sixth of the way between the crossing points and the vertex added before the crossing point and to a point one-sixth of the way between the crossing point and the vertex added after the crossing point. Theoretically, there is a problem with this approach. A trajectory, so represented, would be invalid in the straight line regions near the $\dot{\theta} = 0$ axis where $\dot{\theta}$ was not identically zero. The vertical straight line regions imply that it is possible to have no change in joint angle while maintaining a nonzero angular velocity. This is of course impossible. Practically speaking, the vertices added to force the straight line segments can be placed close enough to the

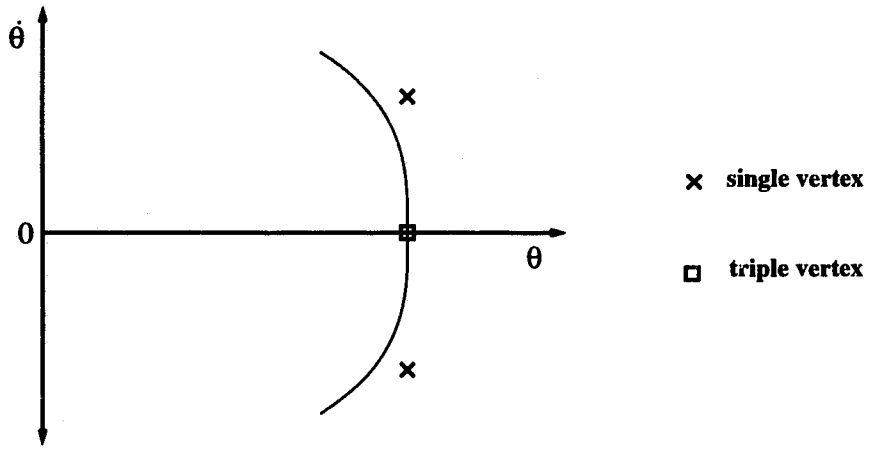


Figure 5.3 Repeated Vertices to Create a Perpendicular Crossing Point

$\dot{\theta} = 0$ axis to make the effect negligible in terms of computing the energy consumption for the prescribed trajectory.

The alternative technique of employing phantom vertices can be used to avoid this problem of an invalid trajectory representation near the $\dot{\theta} = 0$ axis. Suppose two phantom vertices, V_{-1} and V_{m+1}, are added to force the curve to begin and end at points P_s and P_e respectively. The equations for the end segments of the curve are,

$$Q_2(0) = P_s = \frac{1}{6}(V_{-1} + 4V_0 + V_1)$$

$$Q_{m+1}(1) = P_e = \frac{1}{6}(V_{m-1} + 4V_m + V_{m+1}) \qquad (5.44)$$

Solving for the positions of the phantom vertices gives,

$$V_{-1} = 6P_s - 4V_0 - V_1$$

$$V_{m+1} = 6P_e - 4V_m - V_{m-1} \qquad (5.45)$$

The curvature at P_s and P_e is determined by computing the first and second derivatives at these points:

$$Q'_2(0) = V_1 + 2V_0 - 3P_s$$

$$Q'_{m+1}(1) = 3P_e - 2V_m - V_{m-1} \tag{5.46}$$

$$Q''_2(0) = 6(P_s - V_0)$$

$$Q''_{m+1}(1) = 6(P_e - V_m) \tag{5.47}$$

For the special case where the curve is to interpolate the initial and terminal vertices, that is $P_s = V_0$ and $P_e = V_m$, equations 5.45 through 5.47 become,

$$V_{-1} = 2V_0 - V_1$$

$$V_{m+1} = 2V_m - V_{m-1} \tag{5.48}$$

$$Q'_2(0) = V_1 - V_0$$

$$Q'_{m+1}(1) = V_m - V_{m-1} \tag{5.49}$$

$$Q''_2(0) = 0$$

$$Q''_{m+1}(1) = 0 \tag{5.50}$$

From equation 5.45, the phantom vertices are colinear with the two adjacent vertices. From equation 5.49, the curve is tangent to a line connecting the first two or last two vertices at its endpoints. From equation 5.47, the endpoint curvature is zero.

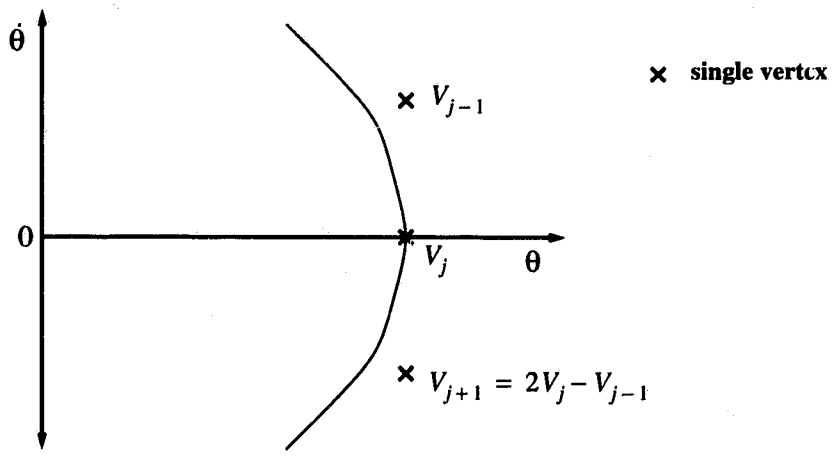


Figure 5.4 Vertex Positions for a Crossing Point

The same technique may be applied to crossing points. Consider the situation shown in Figure 5.4. Vertex V_j is the point at which the trajectory crosses the $\dot{\theta} = 0$ axis. Precede vertex V_j with a vertex V_{j-1} with the same angular value θ_j. Then add vertex V_{j+1} according to equation 5.45 with the subscript m replaced by j. This will ensure that the end of the curve segment Q_{j+1} interpolates vertex V_j and that the trajectory intersects the $\dot{\theta} = 0$ axis at right angles at this point. The curve segment corresponding to vertex V_{j+2} should begin at vertex V_j and have the same slope and curvature as at the end of the Q_{i+1} curve segment. This is verified as follows. The equation for the Q_{j+2} segment is,

$$Q_{j+2}(\bar{s}) = V_{j-1}b_{-3}(\bar{s}) + V_j b_{-2}(\bar{s}) + V_{j+1}b_{-1}(\bar{s}) + V_{j+2}b_{-0}(\bar{s}). \tag{5.51}$$

At $\bar{s} = 0$, this equation evaluates to,

$$Q_{j+2}(0) = \frac{1}{6}V_{j-1} + \frac{2}{3}V_j + \frac{1}{6}V_{j+1}. \tag{5.52}$$

Recall from the phantom vertex assignment according to equation 5.45 that

$$V_{j+1} = 2V_j - V_{j-1}. \tag{5.53}$$

Therefore,

$$Q_{j+2}(0) = V_j \tag{5.54}$$

which shows that the Q_{j+2} curve segment does begin at the crossing point V_j. In a similar manner it can be shown that,

$$Q'_{j+2}(0) = V_j - V_{j-1} \tag{5.55}$$

thereby confirming that the slope is vertical and matches that at the end of the previous curve segment and

$$Q''_{j+2}(0) = 0 \tag{5.56}$$

which also matches the end of the previous curve segment. Having established the mechanics of how to represent the trajectory in phase space, consider how trajectory information is extracted from the representation in order to compute the energy consumption of the trajectory

In phase space representation, angular position and velocity are represented directly. In the course of evaluating the energy consumed for a particular trajectory it is also necessary to extract both time and acceleration information from the joint trajectories. Consider a portion of the joint trajectory which may be represented as a function. Over this section the trajectory is described by

$$\frac{d\theta}{dt} = f(\theta). \tag{5.57}$$

Separating variables and integrating both sides of the equation yields the following expression for time,

$$t = \int \frac{d\theta}{f(\theta)}. \tag{5.58}$$

Representing the trajectory in the parametric form of equation 5.25 and denoting

$$\frac{d\theta}{ds} = \theta'(s) \tag{5.59}$$

the time differential is given by

$$dt = \frac{\theta'(s)ds}{\dot{\theta}(s)}. \tag{5.60}$$

Integrating both sides of equation 5.60 gives the following expression for time.

$$t = \int \frac{\theta'(s)ds}{\dot{\theta}(s)} \tag{5.61}$$

The joint's acceleration at any point along the trajectory is given by

$$\ddot{\theta} = \frac{d\dot{\theta}}{dt} = \frac{d\dot{\theta}}{ds}\frac{ds}{dt}. \tag{5.62}$$

Substituting for dt from equation 5.60 gives,

$$\ddot{\theta} = \frac{\dot{\theta}'(s)\dot{\theta}(s)}{\theta'(s)}. \tag{5.63}$$

A problem exists when calculating time and acceleration at a point where the trajectory crosses the $\dot{\theta} = 0$ axis. At a crossing point in phase space both $\dot{\theta}(s)$ and $\theta'(s)$ are zero

resulting in undefined 0/0 conditions in the equations for both time and acceleration; equations 5.61 and 5.63 respectively. This apparent difficulty can be resolved as follows.

The joint velocity and joint angle in each curve segment are described by cubic polynomials

$$\begin{aligned} \dot{\theta}(s) &= a_0 + a_1\bar{s} + a_2\bar{s}^2 + a_3\bar{s}^3 \\ \theta(s) &= b_0 + b_1\bar{s} + b_2\bar{s}^2 + b_3\bar{s}^3 \end{aligned} \tag{5.64}$$

where $\bar{s}$ is a local parameterization variable described in terms of parametric variable s by

$$\bar{s} = \frac{s - s_i}{s_{i+1} - s_i}. \tag{5.65}$$

From equation 5.65, $\bar{s}$ varies from zero to one on the interval $[s_i, s_{i+1}]$. Further, each crossing is an interpolation point. Therefore at a crossing point the local parameterization variable $\bar{s}$ will take on the value zero or one. Consider the ratio $\dot{\theta}(s)/\theta'(s)$ which occurs in the expression for acceleration, or inverted, in the expression for time. Consider an interval where the crossing point occurs at the beginning of the interval. Here evaluate this ratio as

$$\frac{\dot{\theta}(s)}{\theta'(s)} = \lim_{\bar{s}\to 0} \frac{a_0 + a_1\bar{s} + a_2\bar{s}^2 + a_3\bar{s}^3}{b_1 + 2b_2\bar{s} + 3b_3\bar{s}^2} \overset{\text{l'H}}{=} \lim_{\bar{s}\to 0} \frac{a_1 + 2a_2\bar{s} + 3a_3\bar{s}^2}{2b_2 + 6b_3\bar{s}} = \frac{a_1}{2b_2} \tag{5.66}$$

Note the application of L'Hôpital's rule. When the crossing occurs at the end of the interval the procedure is identical except that the limit is now taken as $\bar{s}$ approaches one.

The biggest difficulty with using phase space to represent the trajectory is that the algorithm user must know before hand how many crossing points exist in the optimal tra-

jectory so that an appropriate starting set of vertex points can be used. Barring this, the optimization algorithm must provide some means of adding or removing crossing points as the optimization proceeds. Consider, for example, planning the trajectory for a multiple link arm. Suppose it is required to move one joint of the arm from rest at one joint angle to rest at a different joint angle. The start and end states of the remaining joints are identical. First, for the joint required to move, the optimal path may require that the joint initially move away from its final destination before moving back toward it, or the optimal path may require that the joint move past its final destination before returning to it, or both. Each of the above situations would require the addition of a crossing point in the initial set of vertex points to enable the optimized trajectory to approach that of the true optimal trajectory. It is difficult to know before hand which crossing points will be necessary to achieve an optimal trajectory.

There is the additional complication that the joints which are not required to move should not necessarily remain fixed for an optimal trajectory to be realized. If this is the case, each of these joints which do move during the course of the optimal trajectory will trace some circuitous route in phase space. The user must know in advance which of these joints should move for an optimum trajectory to be achieved and assign initial vertex points to allow for this motion. Since it is difficult to know the general structure of the final optimal trajectory in phase space before hand it is also difficult to add heuristics to the algorithm to add or prune crossing points when required. One attempted approach is to assume a worst case number of crossing points, and then prune those that appear to be of little significance during the optimization. For example, for a joint which changes location, assume that the link initially moves away from and then overshoots its destination by adding two crossing points. Adding a crossing point creates a half loop above or below the $\dot{\theta} = 0$ axis. If this loop shrinks to a small size during the optimization, the associated

crossing point and any other vertex points associated with the loop would be removed. This procedure still won't resolve the problem of joints which move during the optimal trajectory but start and end at the same state. For these situations it is unclear whether these joints trace a clockwise or counter-clockwise path in phase space. Weighted against the additional difficulties of this form of representation is the possibility that it may be a superior representation of energetically optimal trajectories for robotic manipulators.

One possible solution to the above difficulties is to start the optimization in another space; for example, using the time domain representation. This should give an indication of the form, in phase space, which the optimal trajectory is taking. Based on this information an intelligent choice of starting interpolation points in phase space may be made. This idea is used in Section 5.2.7.

5.2 Optimization Strategies

5.2.1 Direction Set Methods

The predominant approaches to multidimensional optimization are the direction set methods. These algorithms start at some point P in n-dimensional space and then perform a series line minimizations along some vector directions $\vec{n}$ using a one-dimensional minimization routine such as golden section or Brent's method [60]. The algorithms differ in the method they use after each line minimization to select the next direction $\vec{n}$ to try. These algorithms may be classified depending on whether or not they involve computation of the function's gradient. In general, algorithms which make use of gradient information display superior convergence properties. For the present problem of minimizing the energy consumption for robotic manipulators, the gradient information is not generally available analytically. It could be calculated numerically, but this task is in itself computa-

tionally expensive and of suspect accuracy. Therefore, methods which do not require knowledge of the gradient are employed.

The prototypical direction set method, which does not require gradient evaluations, is Powell's method [60]. The purpose of the method is to generate a set of "non-interfering" directions, such that the gain obtained by performing a line minimization along one direction is not lost by subsequent line minimizations along other directions. These "non-interfering directions" are more precisely termed conjugate directions. To define what constitutes conjugate directions, consider the approximation of a multivariable function f by the first two terms in its Taylor series expansion about a particular point $\vec{P}$.

$$f(\hat{x}) = f(\vec{P}) + \nabla f(\vec{P}) \cdot \hat{x} + \frac{1}{2}\hat{x} \cdot \nabla^2 f(\vec{P}) \cdot \hat{x} + \ldots \tag{5.67}$$

From equation 5.67, the gradient of f is

$$\nabla f = \nabla f(\vec{P}) + \nabla^2 f(\vec{P}) \cdot \hat{x} \tag{5.68}$$

From equation 5.68, moving along some direction $\delta\hat{x}$, the gradient changes as follows,

$$\delta(\nabla f) = \nabla^2 f(\vec{P}) \cdot (\delta\hat{x}). \tag{5.69}$$

If a line minimization along some direction $\hat{u}$ has been performed and a new direction along which to minimize, $\hat{v}$, is sought, the condition that the minimization along $\hat{v}$ will not interfere with the minimization along $\hat{u}$ is that the gradient stay perpendicular to $\hat{u}$. From equation 5.69, this condition is satisfied if,

$$0 = \hat{u} \cdot \delta(\nabla f) = \hat{u} \cdot \nabla^2 f(\vec{P}) \cdot \hat{v}. \tag{5.70}$$

When equation 5.70 is satisfied for two vectors $\hat{u}$ and $\hat{v}$, they are said to be conjugate. If this relation holds pairwise for all members of a set of vectors then the vectors are said to form a conjugate set.

The goal for a direction set method in n dimensions is to come up with a set of n linearly independent mutually conjugate directions. If this goal is achieved then one pass of n line minimizations will find the exact minimum of a quadratic form. Such an algorithm is said to exhibit quadratic convergence. Powell's method, given below, achieves this goal.

1. Initialize the set of line minimization directions to the basis vectors $\hat{u}_i = \hat{e}_i \qquad i = 1, \ldots, n$
2. Set the starting position to $\vec{P}_0$.
3. For $i = 1, \ldots, n$, move $\vec{P}_{i-1}$ to the minimum along $\hat{u}_i$. Call this point $\vec{P}_i$.
4. For $i = 1, \ldots, n-1$, set $\hat{u}_i = \hat{u}_{i+1}$.
5. Set $\hat{u}_n = \vec{P}_n - \vec{P}_0$.
6. Move $\vec{P}_n$ to the minimum along direction $\hat{u}_n$. Call this point $\vec{P}_0$.
7. Repeat from step 3 until convergence to a specified tolerance.
8. Restart the algorithm from step 1 (just once) to prevent it from being stopped by an anomalous step which failed to make any improvement.

Powell's algorithm, in the form given above, has a flaw which makes it unusable. The procedure of eliminating at each stage the line minimization direction $\hat{u}_1$ in favor of $\vec{P}_n - \vec{P}_0$, often creates a set of directions that become linearly dependent. As a result the algorithm tends to find the minimum of the function only over a subspace of the original n dimensional problem. Three methods to fix the this problem of linear dependence are;

1. Reinitialize the set of directions to the basis vectors after every n iterations of the basic procedure.

2. Reinitialize the set of directions to the principle directions of the Hessian matrix $\nabla^2 f(\vec{P})$. This procedure, due to Brent, uses the fact that the columns of any orthogonal matrix can be used to reset the search directions. Rather than neglect the information built up in the existing conjugate directions, this method is an attempt to preserve some of that information. This method of course assumes that the Hessian matrix is readily calculated.
3. Discard the direction from the previous set with the largest decrease in favor of $\vec{P}_n - \vec{P}_0$. This heuristic technique, due to Powell, attempts to avoid the build up of linear dependence by eliminating the direction from the previous direction set which is most likely to be the largest component of the new direction $\vec{P}_n - \vec{P}_0$. This method gives up the property of quadratic convergence. A complete implementation is found in [60].

To determine if Powell's algorithm can be successfully applied to the optimal trajectory planning problem, it can be tested in a situation where the form of the optimal solution is known. Such a situation is provided by considering moving a single link arm, with a Type A1 or Type B2 idealized actuator, between two joint angles in a zero gravity environment. If the actuator is of Type A1 then energy is consumed by the actuator both in the positive and negative work regimes. In this case energy is being consumed at all times. The optimal motion policy in this case is to complete the given motion as quickly as possible and the performance index can be expressed as

$$E = \int_0^{t_f} dt. \tag{5.71}$$

For a single link arm this minimum time motion is achieved by driving the actuator to its limit for the first half of the motion and then switching the drive to its opposite limit to bring the link to rest at the desired point. For future reference this test will be referred to as

the "single-link-minimum-time" (SLMT) test. For an ideal Type B2 actuator, energy is consumed only when positive work is being done on the system. In this situation the energy consumption is given by,

$$E = \frac{1}{2}\int_0^{t_f} [\,(1 + \mathrm{sgn}\,(\tau\dot{\theta}))\,\tau\dot{\theta}]\,dt$$
$$E = \frac{1}{2}\int_0^{t_f} [\,(1 + \mathrm{sgn}\,(\tau\theta))\,\tau\theta]\,dt \qquad (5.72)$$

In this situation the optimal control policy is to provide an impulse of maximum torque to start the motion after which the link coasts toward its terminal angle. An impulse of maximum torque in the opposite direction stops the motion. This test will henceforth be referred to as the "single-link-minimum-energy" (SLME) test.

Consider a single link arm of $1m$ length and $1kg$ mass moving in a zero gravity environment. Assume that the arm's mass is concentrated at the distal end of the link. The magnitude of the joint torque required to move the link is numerically equal to the joint's acceleration. Assume the joint actuator is of type A1 and has a maximum absolute torque output of $30Nm$. The goal is to move the arm from zero radians to $1.57rad$, with zero initial and terminal joint velocities, while minimizing the performance index given by equation 5.71. Since this is a constrained optimization problem the performance index must be modified to discourage the descent algorithm from converging to a result which violates one or more of the constraints.

Two approaches are the so called penalty and barrier methods. Penalty methods add a penalty to the performance index for points in the space which violate a constraint. The size of the penalty is usually related to the degree by which the constraint has been violated. As such penalty methods do allow the algorithm to reach points on the constraint boundary. The drawback is that they also enable the algorithm to reach points which violate the constraints. A good choice of a penalty function in conjunction with multiple

passes of the algorithm can ensure that the constraints are violated by an arbitrarily small amount ε; with increasing computational load as ε becomes smaller. Barrier methods construct a barrier by adding a penalty to points inside but close to the constraint boundaries. Often the barrier function is chosen to reach a very large value at the constraint boundary. While these methods prevent the algorithm from reaching solutions on the constraint boundary they prevent solutions which violate the constraints. The choice of an appropriate barrier function in conjunction with multiple passes of the algorithm allows solutions to be obtained arbitrarily close to the constraint boundaries with the computational burden increasing the closer you attempt to get to the constraint boundaries. For the current problem maximum allowable joint torques are provided. Since these constraints can't be violated a barrier method is selected. The barrier used is given by,

$$E_1 = \frac{1}{C_1(\tau_{max} - \tau_m)} \tag{5.73}$$

where τ_{max} is the maximum absolute torque output ($30Nm$), τ_m is the maximum joint torque obtained during the trajectory simulation to compute the motion time and C_1 is a factor used control the steepness of the barrier at the constraint boundaries. If the problem is ill-conditioned it may be necessary to run the algorithm several times while steadily increasing C_1 to ensure convergence to an optimum near the constraint boundary. For the SLMT and SLME problems this was not required. The value of C_1 was set to 1.0×10^6. For the SLMT problem an additional barrier is required. The algorithm must be prevented from attempting to find solutions with negative time. An appropriate barrier function is given by

$$E_2 = \frac{1}{C_2 \omega} \tag{5.74}$$

where ω is the time-scaling factor. The value of C_2 was set to 1.0×10^{10}. With the addition of barrier functions, equation 5.73 and equation 5.74, the overall performance index becomes;

$$J = \int_{t_0}^{t_f} dt + E_1 + E_2 \tag{5.75}$$

For the SLME problem the terminal time of the optimal solution if left unchecked tends towards infinity. This is of course impractical. To make the problem more realistic a penalty can be added if the trajectories terminal time exceeds a specified value t_p. This is accomplished with the penalty function,

$$E_3 = \begin{cases} 0 & t_f \leq t_p \\ K_3 (t_f - t_p) & t_f > t_p \end{cases} \tag{5.76}$$

giving an overall performance index of

$$E = \frac{1}{2}\int_0^{t_f} [\,(1 + \mathrm{sgn}\,(\tau\dot{\theta}))\,\tau\dot{\theta}]\,dt + E_1 + E_3 \tag{5.77}$$

$$E = \frac{1}{2}\int_0^{t_f} [\,(1 + \mathrm{sgn}\,(\tau\dot{\theta}))\,\tau\dot{\theta}]\,dt + E_1 + E_3$$

An alternative to this approach comes from the realization that the solution to this constrained problem will always lie on this constraint boundary. Therefore the time scaling factor, ω, may be eliminated as a variable by simply fixing it to the selected terminal time t_p.

A final consideration, before applying Powell's algorithm, is the number of interpolation points to use in the search for an optimal path. The initial and final states are known and provide the initial and terminal interpolation points and derivatives. It is the

space of interior interpolation points through which the algorithm searches. Selecting only one interpolation point keeps the dimension of the search space small since with n intermediate interpolation points, the number of variables in the optimization is given by $2n+1$ for C1 cubic and quartic interpolation, and $n+1$ for C2 cubic interpolation. The drawback however is that a spline with only one interpolation point can only approximate a relatively small class of functions. As a result one can expect that it is more likely to obtain a better minimum with more interpolation points. The price for this richer function space is a higher computational burden. As the dimensionality of the search space is increased one runs into the law of diminishing returns. It takes an ever increasing effort to achieve more and more marginal improvements. One solution to this function space richness versus computational burden trade-off, is to start the algorithm with one interior interpolation point. After the algorithm has converged to a specified tolerance, ε, increase the number of interior knots by halving each interpolation interval. This should be done without affecting the interpolation curve. Restart the algorithm with this increased number interpolation points. Continue this procedure until the return from increasing the number of knots is less than a specified tolerance, δ. The tolerances selected are as follows.

$$\varepsilon = \delta = 0.01\ J \tag{5.78}$$

A summary of the results for the SLMT trajectory optimization are shown in Table 5. The best results are achieved with a C1 piecewise cubic interpolation function after 179 function evaluations. Adding additional interpolation points by bisecting the interpolation intervals fails to yield an improvement. The reason for this is that a single interpolation point midway in time between the initial and terminal points is capable of exactly representing the true optimal trajectory for this simple manipulator. The optimal trajectory and corresponding joint torque is shown is Figure 5.5. Note that the joint torque is at the positive limit for half of the joint trajectory and at the negative torque limit for the

other half of the trajectory. Since C2 cubic interpolation and the quartic interpolation have C2 continuity, its is impossible for the optimal in these function spaces to be the true optimum in this example. With C2 cubic interpolation an optimized trajectory time of 0.5606 seconds is achieved. Bisecting the intervals yields no further improvement. The richer function space of the quartic interpolation, owing to the freedom to set the derivative at the interpolation points, gives an improved result of 0.4755 seconds after 1707 function evaluations. The optimized trajectories and corresponding joint torques for the C2 cubic and quartic interpolation are shown in Figures 5.6 and 5.7 respectively.

A summary of the results for the SLME trajectory optimization are shown in Table 6. The best results are achieved with piecewise C2 cubic spline interpolation after 782 function evaluations. The torque curves of Figures 5.8 through 5.10 show the expected torque characteristics of a torque pulse to the joint at the start of the trajectory, followed by "cruise period" where the torque is near zero, and finally a negative torque pulse to bring the link to rest.

Powell's method performs adequately for the simple one link manipulator test cases. However, for an optimization approach to be useful it must be able to be scaled to

Table 5 Single Link Minimum Time Trajectory Optimization

interpolation curve	**# of variables**	t_f	**# function evaluations**
C1 cubic	3	0.4577	179
	7	0.4577	407
C2 cubic	2	0.5606	123
	4	0.5606	277
quartic	3	0.5209	189
	7	0.4766	1174
	15	0.4755	1707

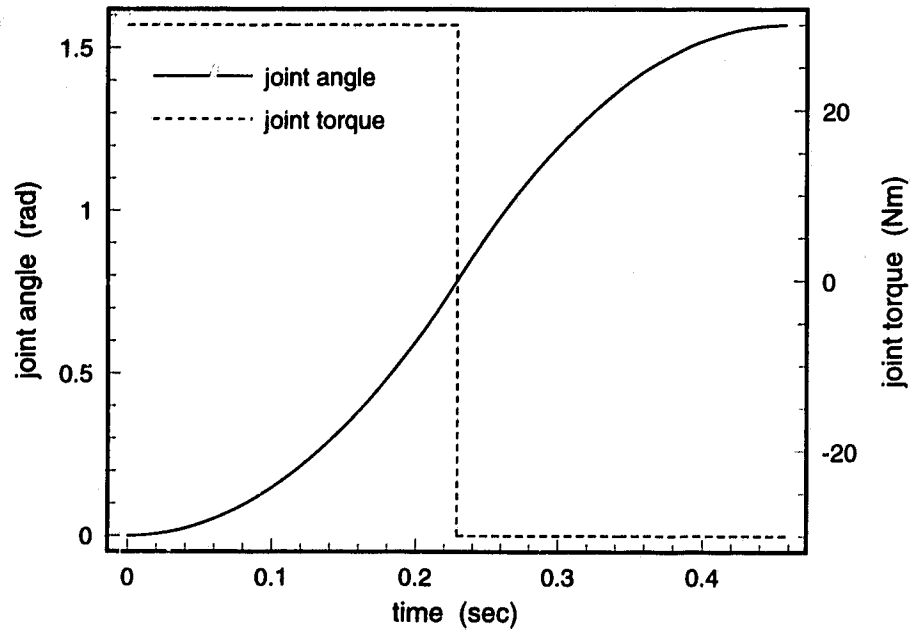


Figure 5.5 Single Link Minimum Time Optimal Trajectory - C1 Cubic Interpolation

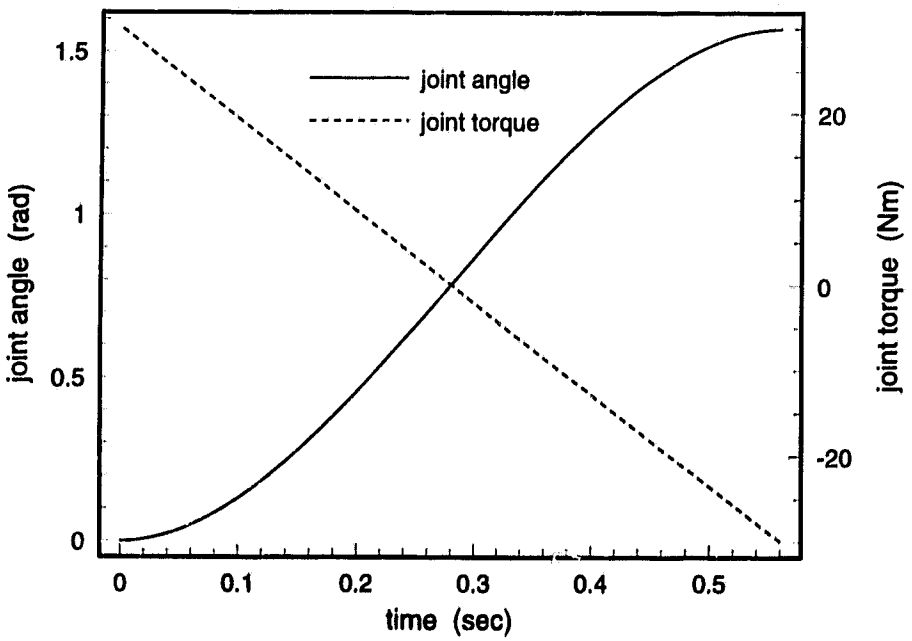


Figure 5.6 Single Link Minimum Time Optimized Trajectory - C2 Cubic Interpolation

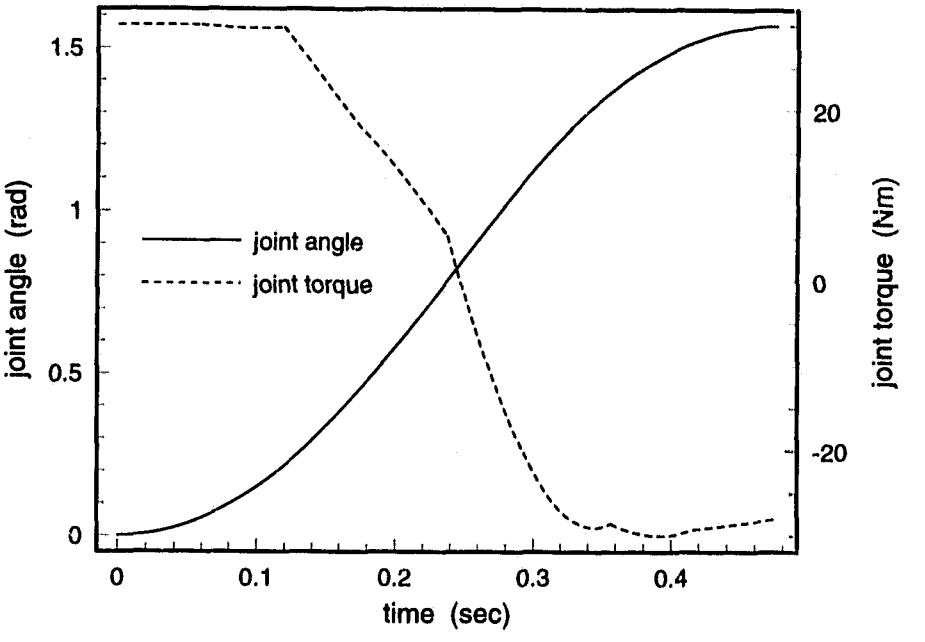


Figure 5.7 Single Link Minimum Time Optimized Trajectory - Quartic Interpolation

Table 6 Single Link Minimum Energy Trajectory Optimization

interpolation curve	# of variables	energy consumed (J.)	# function evaluations
C1 cubic	3	0.006393	168
	7	0.005375	1065
	15	0.005372	1527
C2 cubic	2	0.006393	120
	4	0.004869	494
	8	0.004864	782
quartic	3	0.005585	250
	7	0.005147	920
	15	0.005143	1439

problems of higher dimensionality. To be of practical use in determining minimum energy trajectories a method must be capable of at least optimizing the trajectories for the gross motion joints which consume the majority of the energy. For many manipulator systems

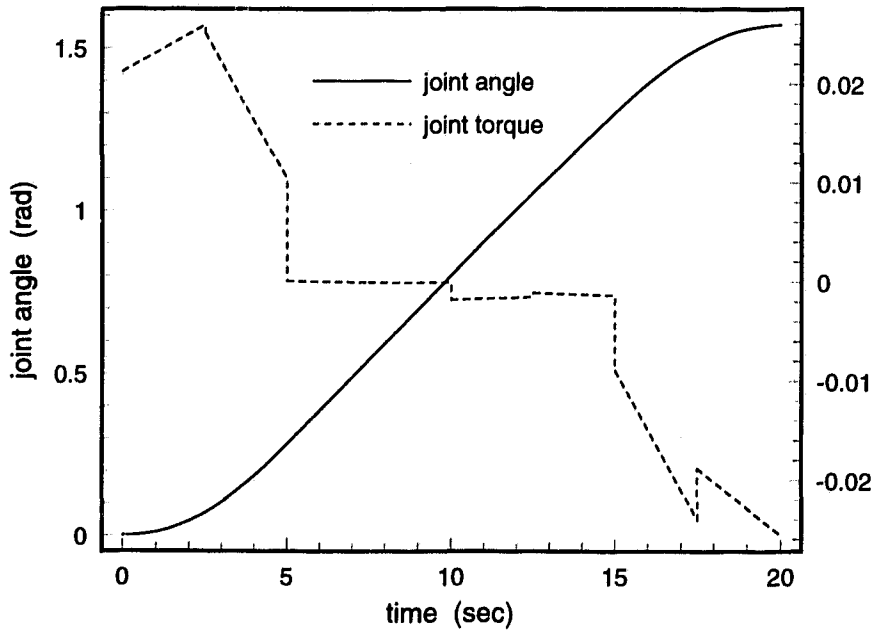


Figure 5.8 Single Link Minimum Energy Optimized Trajectory - C1 Cubic Interpolation

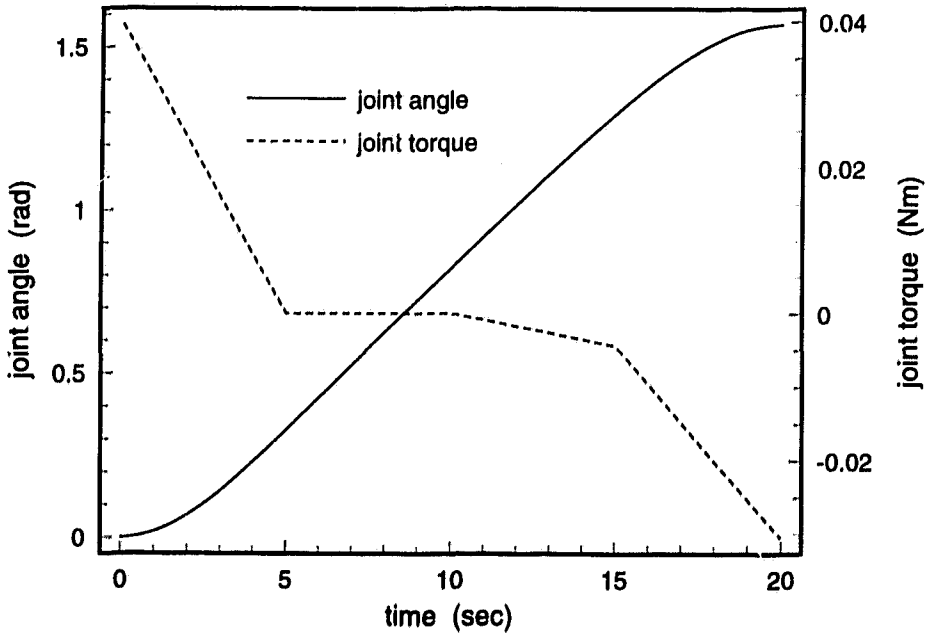


Figure 5.9 Single Link Minimum Energy Optimized Trajectory - C2 Cubic Interpolation

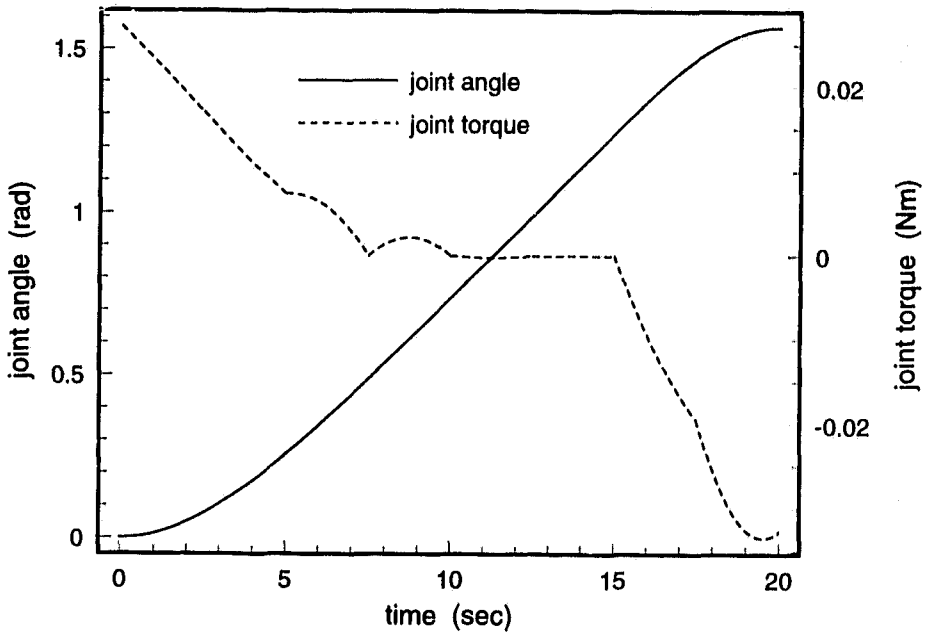


Figure 5.10 Single Link Minimum Energy Optimized Trajectory - Quartic Interpolation

this requires an optimization approach to comfortably handle problems with a minimum of three links. The test system here will be the three link arm model developed in Section 4.1.4, driven by the D.C. actuators modelled in Section 4.2.2. The increased dimensionality of the problem will illustrate how various methods scale in terms of their computational burden. In addition, the more complex solution space is likely to include multiple local minima which will trap some optimization procedures.

For the purposes of numerical simulation, the following numerical values are ascribed to the mechanism model parameters of Section 4.1.4. The link masses are, $m_2 = 100\ kg$ and $m_3 = 50\ kg$. The link and mass center offsets are, $r^2_{3_x} = 1.5\ m$, $r^2_{2G_x} = 0.5\ m$ and $r^3_{3G_x} = 0.5\ m$. The relevant rotational inertia terms are, $J^{1G}_{1z} = 0.5\ kg \cdot m^2$, $J^{2G}_{2x} = 34\ kg \cdot m^2$, $J^{2G}_{2y} = 34\ kg \cdot m^2$, $J^{2G}_{2z} = 1.0\ kg \cdot m^2$, $J^{3G}_{3x} = 4.4\ kg \cdot m^2$, $J^{3G}_{3y} = 4.4\ kg \cdot m^2$ and $J^{3G}_{3z} = 0.5\ kg \cdot m^2$. Each joint is assumed to

Table 7 Three Link Minimum Energy Trajectory Optimization

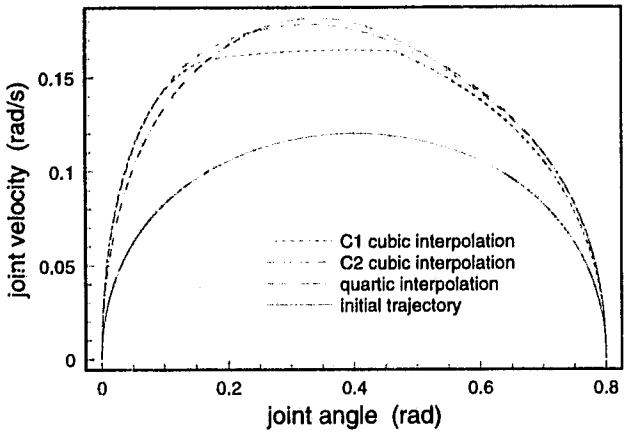
interpolation curve	# of variables	energy consumed (J.)	# function evaluations
C1 Cubic	7	160.0	593
	19	158.6	1102
C2 Cubic	4	160.2	296
	10	156.5	877
Quartic	7	160.6	557
	19	159.8	1162

driven with the same type of actuator. The parameters for the D.C. motor actuator model developed in Section 4.2.2 are selected as follows. The Coulomb and viscous friction coefficients are $\tau_\mu = 0.1\ Nm$ and $K_f = 0.0\ Nms$ respectively. The motor constant is $K_m = 0.4\ Nm/A$. The armature inertia is $J_a = 0.0002\ kg \cdot m^2$. The supply voltage is $E = 24\ V$. The armature resistance is $R_a = 2.0\ \Omega$ and the gear ratio is $N = 0.01$. The maximum absolute motor torque is 5.0 Nm. Finally, the arm is assumed to be operating under no gravitational load; so $g = 0.0\ m/s^2$.

The trajectory to be planned is a minimum energy consumption trajectory from (0.0, 0.0, 0.0) radians to (0.8, 0.8, -0.8) radians with initial and terminal velocities all zero. With an initial time scaling factor of ten seconds the energy consumption of a cubic spline trajectory between the initial and terminal states is 210.5 J. The same barrier and penalty functions used for the single link case (equations 5.73, 5.74 and 5.76) are used. The convergence tolerances for Powell's algorithm and the outer loop of the algorithm are, as before, $\varepsilon = \delta = 0.01\ J$. The results for each type of interpolation function are summarized in Table 7. As indicated by Figure 5.11, each type of interpolation curve produces a similar result but the C2 cubic interpolation achieves the result with fewer function evaluations due to the lower dimensionality of the space which must be searched for a given

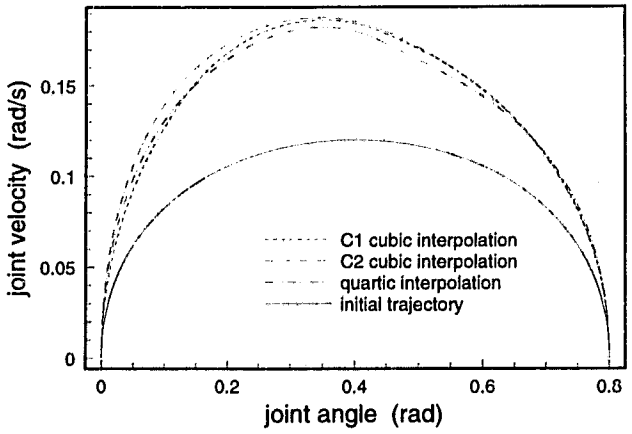
number of intermediate interpolation points. This test suggests that the piecewise C2 cubic spline provides as good a representation of the optimal trajectory as the methods which include derivative information in the search space. Therefore, one would be advised to at least start with the assumption of a piecewise C2 cubic trajectory for the minimum energy trajectory planning for robotic manipulators.

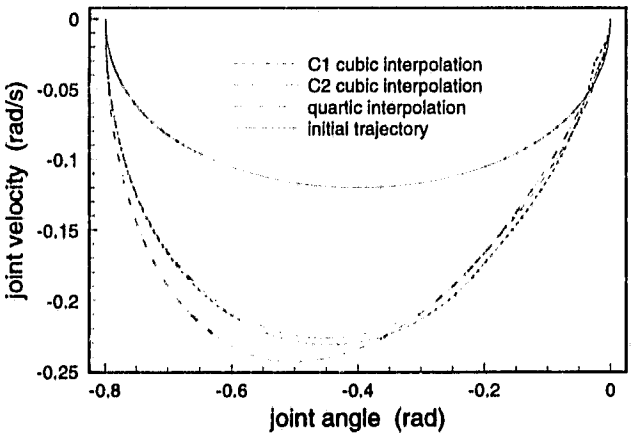
A drawback to direction set methods is that they are local optimization procedures. As such they may get trapped in a local minimum with an energy consumption significantly higher than the global optimum. This is confirmed by performing the previous trajectory optimization starting with each of one hundred randomly selected initial trajectories. This test was performed using piecewise C2 cubic interpolation with three intermediate interpolation points for each joint, or a ten dimensional search space. Each initial trajectory is determined by randomly selecting a value from within a specified range for each of the three intermediate interpolation points for each joint and for the time scaling factor. The range of valid values used for each interior interpolation point and the time scaling factor are shown in Table 8. If a randomly selected initial trajectory was invalid, in that the particular trajectory required violation of the motor torque constraints, it was removed and another random initial trajectory was generated as a replacement. From the one hundred different starting trajectories one hundred local minima, none of which are identical, are obtained. Energy consumption values for these trajectories range from a low of 147 Joules to a high of 208 Joules. Clearly, the quality of the optimal trajectory depends a great deal on the initial trajectory selected. There are several approaches in the literature which have been used to avoid the problem of getting trapped in local minima. Those that were considered in the course of the investigation and the extent to which they were pursued are described briefly below.



joint 1

joint 2





joint 3

Figure 5.11 Minimum Energy Trajectory for a Three Link Arm Using Powell's Method

Table 8 Ranges for Interpolation Points for Random Initial Trajectory Generation

interpolation point	joint	minimum	maximum
1	1	-0.8	0.8
	2	-0.8	0.8
	3	-0.8	0.8
2	1	-0.1	0.9
	2	-0.1	0.9
	3	-0.9	0.2
3	1	0.2	1.2
	2	0.2	-1.2
	3	-1.2	-0.2
time scaling factor	NA	2.0	20.0

5.2.2 Interval Methods

Most "global optimization" methods are not able to guarantee that the global optimum points have been found to a given tolerance. Examples include techniques which use local methods started from many different points, genetic algorithms and simulated annealing. This shortcoming may be avoided using methods based on the techniques of interval analysis. These techniques rely on ones ability to represent and perform arithmetic on numerical values as intervals. Consider that modern computers represent and perform arithmetic on numbers in what is called floating point form. In this form, the set of real numbers are represented by a subset of real numbers which can be represented by the computers finite word length for a floating point number. This subset is called the machine representable numbers. This means that real valued data are only approximated by machine numbers. Interval arithmetic provides the ability to estimate and control these errors automatically. Instead of representing a number by a real value x, it is represented by a closed interval $X = [a, b]$ having machine number upper and lower boundaries, a

and b respectively, such that the interval X contains the value x. The width of the interval may be used as a measure of the quality of the approximation and computations are carried out using intervals instead of real numbers. The four basic interval arithmetic operations for addition, subtraction, multiplication and division are,

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d] \\
[a,b] - [c,d] &= [a-d, b-c] \\
[a,b] \cdot [c,d] &= [min(ac, ad, bc, bd), max(ac, ad, bc, bd)] \\
[a,b] / [c,d] &= ([a,b] \cdot [1/d, 1/c]) \text{ if } 0 \notin [c,d]
\end{aligned} \tag{5.79}$$

The operation of division can be extended to intervals which include zero, provided the arithmetic operations are extended to unbounded intervals. This extension and methods for other mathematical operations can be found in references [1][63][64]. In order to guarantee that the intervals calculated on a floating point machine are always inclusive, the direction of rounding in the computer must be controlled. For example, when computing the lower bound of an interval, the computer must be forced to round down. If the computer were to round up, the lower end of the interval would be lost. Fortunately most modern computer processors provide an instruction which allows the direction of rounding to be controlled. There are a number of software packages in which machine interval arithmetic are implemented. A library of C++ routines which implements interval routines for a large number of arithmetic, set, transcendental, vector and matrix functions was developed during the course of this thesis work.

The concept of inclusion can be extended from numbers and mathematical operations to more general functions. In [64], Ratschek and Rokne show that any function $f(x)$, composed of the variable x, or its components, $x_1, x_2, \ldots, x_m$, real numbers such as con-

stants or coefficients, the four basic arithmetic operators, a set of predeclared functions for which interval extensions exist (ex. sin, cos, log), and auxiliary symbols such as brackets and parenthesis, can be represented in an interval form by replacing each element of the function by its interval extension. Such a representation is called the function's natural interval extension. The natural interval extension of $f(x)$ is denoted $F(Y)$ where Y is an element of the interval numbers. Here,

$$x \in Y \quad \text{implies} \quad f(x) \in F(Y). \tag{5.80}$$

It turns out that different expressions for the same function lead to different natural interval extensions of that function. As such one form of the natural interval extension for a function may form a better representation than the other. For example consider the functions $f_1(x) = x - x^2$ and $f_2(x) = x(1-x)$. These are different as expressions, but equal as functions. The natural interval extensions which result from these expressions are also different. Consider the interval, $Y = [0, 1]$. Then, $F_1(Y) = Y - Y^2 = [-1, 1]$ and $F_2(Y) = Y(1-Y) = [0, 1]$. For comparison, the true interval value for the function over Y is $\square f(Y) = [0, 1/4]$. It is very important to find an interval extension that approximates the true interval of the function as closely as possible.

A measure of the quality of the inclusion function is called the excess width,

$$w(F(Y)) - w(\square f(Y)) \tag{5.81}$$

where $w([a, b]) = b - a$ is the width of interval $[a, b]$. A measure of the asymptotic decrease in the excess width as $w(Y)$ decreases is called the convergence order. If there exists a constant $c \geq 0$ such that

$$w(F(Y)) - w(\square f(Y)) \leq c w(Y)^{\alpha} \tag{5.82}$$

the inclusion function $F(Y)$ is said to have convergence order α. For the optimization algorithms to be computationally quick, it is important to find an inclusion function with as high an order as possible. Natural interval extensions are not the only choice for the generation of inclusion functions. For many functions, particularly polynomials and rational functions, superior inclusion functions can be constructed by the use of centered forms. Several types of centered forms are derived in [63].

Given that an inclusion function F can be created for the function f, for which the global minimum f^* over the m-dimensional interval hyperbox $X \in I^m$ is sought, the following algorithm, attributed to Moore[53] and Skelboe[79], determines f^* by splitting the domain X into subboxes and then searching the subboxes using a branch and bound principle.

1. Set interval hyperbox Y to be the initial domain. $Y = X$
2. Calculate $F(Y)$.
3. Set $y = min(F(Y))$.
4. Initialize a list $L = ((Y, y))$.
5. Choose the coordinate direction k along which the hyperbox Y has its edge of maximum length.
6. Bisect Y normal to direction k obtaining boxes V_1 and V_2 such that $Y = V_1 \cup V_2$.
7. Calculate $F(V_1)$ and $F(V_2)$.
8 Set $v_i = min(F(V_i))$ for $i = 1, 2$.
9. Remove (Y, y) from the list L.
10. Enter the pairs (V_1, v_1) and (V_2, v_2) into the list such that the second members of all pairs of the list do not decrease.
11. Denote the first pair of the list by (Y, y).

12. If termination criteria hold, then go to 14. For most cases the termination criteria, "If $w(Y) < \varepsilon$ then terminate.", will be sufficient. Due to machine interval arithmetic there are cases where this termination criteria will fail. Ratchek and Rokne[64] suggest alternative termination criteria for such cases.
13. Go to 5.
14. End.

An enhancement of the above algorithm by Hansen[27][28], allows one to obtain an error estimate for the approach and to obtain all minima, both local and global. In addition a number of acceleration devices, designed primarily to weed out portions of the domain in which no minima exist, are available. These devices are covered in Ratschek and Rokne[64].

For the minimum energy trajectory problems being considered in this work, it is simple to describe the solution domain in terms of an interval hyperbox. The possible range for each spline interpolation point simply forms one edge of the hyperbox. It is equally as simple to create an inclusion function for the energy consumption using natural interval extensions. The difficulty is that, such an inclusion function is a poor representation of the bounds on the energy consumption. Even for modest domain sizes the natural interval extension inclusion functions yield the infinite range $[0, \infty]$. A much better inclusion function is required if the interval approach is to be successful in solving these types of problems. Due to the complexity of the energy consumption function, a good inclusion function has yet to be developed.

5.2.3 Genetic Algorithms

Genetic algorithms are an attempt to make use of the process of natural selection which determines the genetic makeup in nature, or perhaps more accurately, the selective

breeding practices used by man since the first formation of an agrarian society. The structure of most genetic algorithms may be described as follows. More detail may be found in [19].

The possible solution space of the problem is discretized. For the problem of optimal trajectory planning, the interior interpolation points for each joint are assumed able to take on a fixed number of values within some range. Each possible solution must be representable by a bit string. For example if each interpolation point of each joint were able to take on one of 256 different values, then each interpolation point could be represented by an eight bit binary number. An entire trajectory for say a three link arm with three interior interpolation points for each link could then be represented by a set of nine of these eight bit numbers or a single 72 bit number. Even with this coarse discretization the number of possible trajectories which can be generated of the size of the genospace is 2^{72} or 4.722×10^{21} members. If it took, on average, one second to evaluate the performance of a single member of the population it would take in excess of 10^{14} years to search the entire genospace member by member for a global optimum.

A starting population is generated and a measure of fitness "fitness function" is evaluated for each member of the population. The way in which the starting population is generated depends on the problem. The starting population is often randomly generated or may constructed by perturbations from a local optimization or human solution. The size of the initial population is up to the programmer or program user. A larger initial population provides more initial genetic diversity which may lead to finding a better solution in fewer generations but at the cost of more function evaluations at each generation.

The first stage of generating a new population from the preceding one is called "crossover". It is analogous to a selective breeding process. The program selects pairs of

the existing population based on the value of each members fitness function using some set of heuristic rules. For each selected pairing, each member of the pair is cut at the same randomly selected point along the bit string. The tail end of each string is then swapped to generate two new members of the population which maintain some of the characteristics of their predecessors. This is the stage at which genetic algorithms vary widely. The process of pairing members of the population for crossover can vary between two extremes. At one extreme, the pairings are completely random. This is not advisable since it results in a completely random walk through the genospace. The other extreme is that the new population is generated by repeated pairings of the two most fit individuals in the population. This extreme would result in the algorithm being confined to a not necessarily globally optimal or even good region of the genospace. In order to provide for an increasingly fit population it is necessary that the more fit members of the population, on average, produce more offspring.

The second stage of generating the new population is "mutation". In this stage, each bit of each member of the population which resulted from crossover is flipped (changed from zero to one or one to zero) with some probability μ between zero and one. Mutation prevents the population from getting stuck in a few regions of the genospace by allowing jumps to new untried regions. The size of μ is another parameter left to the programmer or program user. If it is too large the algorithm will randomly walk through the solution space, if it is too small a monoculture may develop. Typically, μ is set to value in the neighborhood of three percent.

The fitness function for each member of the new population is evaluated. A record is kept of the best few bit strings yet encountered and the corresponding value of the fitness function. The population is then ready to create the next generation starting with the

crossover stage. With each successive generation the overall population should become more fit thereby increasing the chances of generating an improved solution.

Given enough time and a nonzero mutation factor μ, a genetic algorithm will find a nearly globally optimal solution from within the discrete genospace. For continuous optimization problems such as trajectory planning one is limited to the best trajectory which can be represented given the granularity of the discretization. The biggest drawback to genetic algorithms is the huge number of function evaluations, relative to most local optimization procedures, which must be performed before one begins to achieve near optimal solutions in the population. The benefit of this extra effort of course is that one is able to reach a near global optimum without an exhaustive search of the solution space. The increased computational effort of the extra function evaluations can be partially mitigated by the ability to evaluate the fitness function for population members in parallel. Genetic algorithms can be powerful tools for difficult optimization problems but they are best suited to situations where a solution to a single problem is sought; for example, the optimal layout of a pipeline to deliver reagents to various points in a chemical plant. For the trajectory planning problem where a new optimization must be performed each time the initial and terminal states are changed, genetic algorithms are too time consuming.

5.2.4 Simulated Annealing

Like genetic algorithms, simulating annealing is a technique developed for the global solution of combinatorial optimization problems. Simulated annealing, like genetic algorithms, has its basis in a natural process; in this case the thermodynamic process by which materials achieve lower energy states when cooled. The technique is based on the Botlzmann probability distribution,

$$Prob(E) \sim e^{-E/(kT)} \tag{5.83}$$

which says that at a system in thermal equilibrium at temperature T has its energy probabilistically distributed among all different energy states E. The quantity k. called Bolzmann's constant, relates temperature to energy.[1] Equation 5.83, indicates that even at low temperatures there is a probability of a high energy state. Therefore if a system following this distribution changes from one energy state to another at a given temperature there is a possibility that the energy state it moves to will be higher. The lower the temperature, the less likely this "uphill" movement is to occur. Allowing this uphill movement allows simulated annealing algorithms to avoid getting trapped in local minima.

This approach was first incorporated into a numerical optimization by Metropolis and his co-investigators in 1953.[52] Given a process by which a successive number system states with calculable energy levels (performance index, fitness function) could be generated, a simulated thermodynamic system was assumed to accept a proposed change in state based on the energy change from existing energy E_1 to the energy of the proposed state E_2 with probability,

$$p = e^{-(E_2 - E_1)/T}. \tag{5.84}$$

Note that if $E_2 < E_1$, the calculated probability is greater than one. In such cases the probability is assumed one and the new state is always accepted. Therefore the system always accepts a downhill change while sometimes accepting an uphill change.

---

1. Boltzmann's constant: $k = 1.3085 \times 10^{-23} \frac{J}{^{\circ}K}$

Based on the above discussion the essential elements for creating a simulated annealing algorithm for a particular problem are as follows. First one must have a description of the possible system configurations. For the optimal trajectory planning problem this is just the location of the interior interpolation points for each joint. Second a generator must be provided to create random changes in the system configuration. This is easily provided by selecting random locations for each interior interpolation point for each joint from within a specified range. Third, the objective function E, is identical to that used for the descent algorithm. That is, the energy consumption of the trajectory created by the set of interpolation points. Finally, there must be a control parameter T, analogous to the system temperature and an associated annealing schedule which determines after how many changes in configuration the temperature is lowered and by how much it is lowered.

A simulated annealing algorithm was applied to the same three link arm minimum energy trajectory planning problem used to test Powell's method in Section 5.2.1. A pseudo-random number generator created random changes in trajectory by selecting random locations for each of the interior interpolation points from within the ranges specified in Table 8. The initial "temperature" of the system was set to $T = 500$, which means initially the algorithm would accept an uphill movement of 500 Joules with a probability of 36.8 percent. After 5,000 accepted state transitions, or a maximum of 10,000 attempted state transitions the temperature was lowered by a factor of 0.9. Following this annealing schedule, the algorithm achieved a trajectory with an energy consumption of 223 Joules after 335,499 function evaluations. This is much worse than the performance of Powell's method. Not only was the minimum achieved higher than the worse case encountered for Powell's algorithm, the number of function evaluation is over 100 times more than a typical run of Powell's algorithm on the same problem. The problem with this algorithm is clear. It is taking random guesses at an optimum in a space with a continuum of possible

trajectories. It is possible that a close to globally optimal trajectory could be reached if the number of accepted state transitions at each temperature level were dramatically increased. This of course would result in a corresponding increase in the number of function evaluations necessary. This algorithm is much like the approach of genetic algorithms except for the fact that genetic algorithms do not create totally random changes in state but rather have changes which occur in more "preferred areas". The large number of function evaluations required by this approach, like genetic algorithms, indicates that it is best suited to problems where a single global solution is sought and computational time is not an important factor.

5.2.5 Dynamic Programming

Dynamic programming is a valid approach to determine a globally optimal trajectory between two states. The approach is identical to that described for the optimal control problem described in Section 2.2.6 on page 29, except that it is no longer necessary to quantize a set of control variables and interpolate between points on the tessellated grid. For the trajectory planning problem the control decision is the decision to which state on the next stage of the grid to move. For clarity, the dynamic programming procedure as it applies to the optimal trajectory planning problem in joint angle / time space is as follows.

1. Generate a discrete grid of possible states between the given initial and final states. First divide the scaled time ($0 \leq \tau \leq 1$) into N stages numbered $i = 1 \ldots N$. At the terminal stage $i = N$, quantize the range of possible time scaling factors, ω, into M values enumerated, $j = 1 \ldots M$. At intermediate stages, $i = 2 \ldots N-1$, for each of the P joints, tessellate the possible ranges joint angles into Q values and the possible ranges of angular velocities into R values. This will result in $S = Q^P R^P$ points, enumerated $k = 1 \ldots S$, at each intermediate stage. The initial stage, $i = 1$, contains only the given initial point.

2. Determine the energy or cost associated with each of the M points on the terminal stage and store the value with each point. This is where a penalty function associated with the terminal time would be evaluated. If there is no time penalty function to apply the stored energy would simply be zero.
3. From each of the S points on stage $i = N-1$, compute the energy cost of moving to each of the M points on stage $i = N$ plus the energy stored at the destination point. Between any two points on adjacent stages the trajectory segment follows a quartic spline trajectory. A quartic spline is selected to ensure a continuous acceleration curve. The five coefficients of the quartic spline are calculated from states of trajectory segment's endpoints and the known accceleration at the terminal point of the trajectory segment. At this final interval of the grid the acceleration is assumed to be zero at the end of the trajectory segment. At all other trajectory intervals, the terminal acceleration is given by the initial acceleration of previously calculated next trajectory segment. The time for the move is given by the time scaling factor of the point being moved to divided by $N-1$. At each of the S points store the number j of the destination point associated with the lowest energy, the energy value, the acceleration at the start of the interval and the time scale factor associated with the destination point.
4. The procedure continues moving backwards through the stages until the initial stage. At each point on each stage, the point on the next stage which provides a minimum energy value evaluated as the cost of the transition between the points plus the energy value stored at the destination point is recorded. The resulting energy is also saved as is the acceleration at the start of the interval and the time scaling factor associated with optimum destination point of the next stage.
5. The optimal path sequence is recovered by moving forward through the stages following the list of pointers to the next stage of the grid which was constructed during the backward pass of the algorithm.

This procedure has several advantages. First, it finds a global optimum. In fact not only does it finds the globally optimal trajectory between the initial and final states but

between any of the states on the grid and the final state. Second, many function evaluations may be done in parallel. For example, between two intermediate stages with S points each, S^2 function evaluations may be done simultaneously if one has access to a suitable computer architecture. The biggest disadvantage to this approach is that it doesn't scale well. To provide a good approximation for a continuous problem by using a discrete grid may require a tight grid spacing and therefore large values for M, Q and R. The total number of function evaluations for the algorithm is given by,

$$\text{total function evaluations} = Q^P R^P \{ (N-3) Q^P R^P + M + 1 \}. \tag{5.85}$$

From equation 5.85, the number of function evaluations increases by the power of 2P with Q and R and linearly with M. This "curse of dimensionality" is the bane of dynamic programming approaches.

As an example, consider the same three link problem discussed in Section 5.2.1. With Q and R set to 10, M set to 100 and N set to 5, the dynamic programming algorithm will require more than 2×10^{12} function evaluations to reach an optimum trajectory over the prescribed grid. Despite the large number of function evaluations the grid is still very coarse. To adequately cover the possible solution space with the number of points prescribed above the spacings in joint angle and angular velocity and time scaling factor were 0.1 radians, radians per second, and seconds respectively. The computing power required to solve this problem, even with this coarse grid, using dynamic programming is not available. Even though between two intermediate stages, 1.0×10^{12} functions may be evaluated simultaneously, this is a practical impossibility.

Like genetic algorithms or simulated annealing, dynamic programming offers the possibility of a near globally optimal trajectory. The computational cost of achieving this trajectory is high or impractical with current computing capabilities. In the sections which

follow methods of reducing the number of function evaluations to reach an improved trajectory by modifying the dynamic programming algorithm are investigated. In performing these modifications, the guarantee of finding a global optimum is given up in the hopes of drastically reducing the number of function evaluations while still achieving a "near" optimal trajectory.

5.2.6 Modified Dynamic Programming

To reduce the number of function evaluations required by the dynamic programming algorithm, the algorithm is modified on two fronts; reduce the dimension of the problem and/or increase the coarseness of the grid. The largest gains can be made by reducing the dimension of the problem. The dimension of the dynamic programming algorithm described in the previous section, at the key intermediate stages, is $2P$, since space is tessellated in one dimension for both joint angle and joint velocity for each of P joints. Suppose the need to tessellate for the joint velocity is removed. This would cut the problem's dimension in half to P. The difficulty is how to maintain the C2 trajectory without velocity information at intermediate points in the trajectory. This can be accomplished by changing the class of functions in which optimum is sought. Rather than considering the trajectory to made of a sequence of subtrajectories between the state points on adjacent stages, consider a trajectory associated with each grid point to be a trajectory through the zeroth points on all previous stages, the point of interest on the current stage, and the linked list of points constructed by the algorithm as it moves backwards from the terminal stage. Using this representation, a C2 piecewise cubic spline can be used to define the trajectory as outlined in Section 5.1.2, since the specified initial and terminal velocities are all that is required to complete the set of equations 5.11. With this change in trajectory representation the dynamic programming algorithm is as follows.

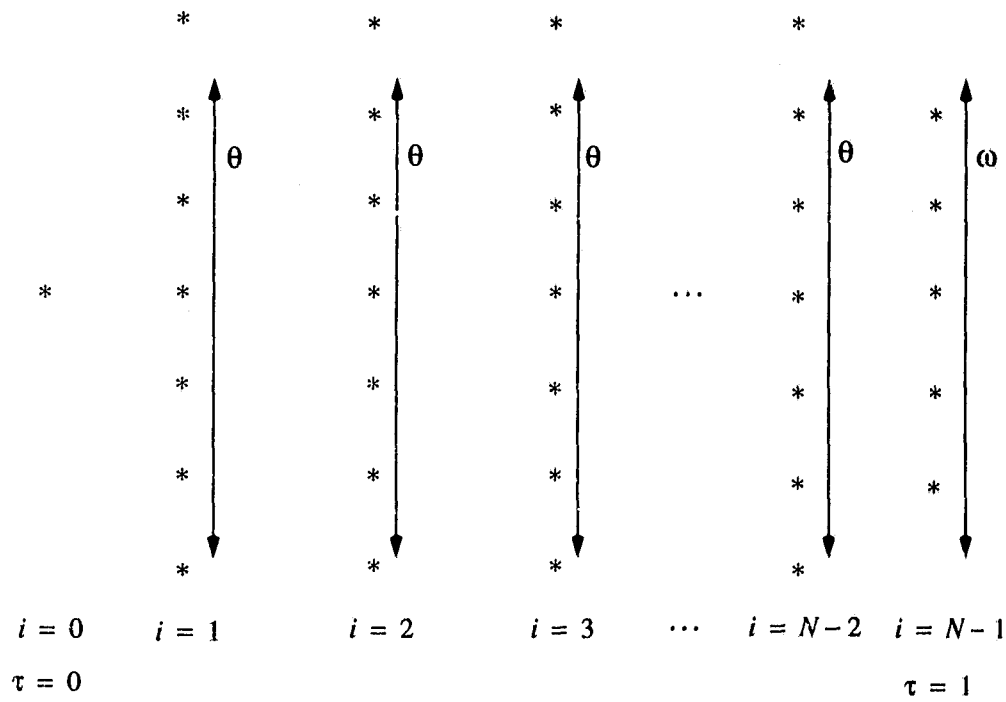


Figure 5.12 Grid for a Single Link Arm

1. Consider the scaled time for the manipulator motion $\tau_f = 1.0$ to be divided into N stages, numbered zero through $N-1$. At each stage a discrete grid of Q possible joint angles for each joint is generated. Number these points on the grid zero through $Q^P - 1$ where P is the number of joints. The initial stage has one point corresponding to the known initial joint angles. The terminal stage has one set of joint angles but is tessellated into M points numbered zero to $M-1$, along a dimension corresponding to the time scaling factor ω. When constructing the grid of points, the zeroth point of each stage is set to the initial trajectory specified. The remaining points at each stage of the grid are determined by successively adding offsets of $\pm\delta$ to the zeroth point recursively for each dimension, where δ is the grid spacing. A similar procedure is used to tessellate the time scaling factor along the terminal stage. Figure 5.12 shows a grid pattern for a single link arm. For multiple link arms the intermediate stages have one dimension of tessellation for each joint.

2. Starting at the terminal stage ($i = N-1$), calculate the energy consumption for a trajectory corresponding to each point of the stage. A point's trajectory is constructed from the zeroth points on all previous stages and the point being considered on the current stage. Store the energy consumption associated with each point in a structure representing the point.
3. Take the minimum energy consumption value from the terminal stage and store it in the structure associated with the zeroth point of stage $N-2$. Create a pointer from the zeroth point on stage $N-2$ to the point on the terminal stage corresponding to this minimum energy consumption trajectory.
4. For each point, one through $Q^P - 1$, on stage n-2 determine the minimum energy trajectory from a set of trajectories made up of the zeroth points on all previous stages, the current point and each point on the next (terminal) stage. Store this minimum energy consumption in the structure for the current point and set a pointer to the corresponding point from the next (terminal) stage. Take the minimum energy value from the current stage and store it in the zeroth point of stage $N-3$. Create a pointer from the zeroth point of stage $N-3$ to the point on stage $N-2$ corresponding to this minimum energy trajectory.
5. Continue this procedure, moving back by stages until the initial stage is reached. Remember, for each point on each stage the trajectory is made up of the zeroth point on all previous stages, the current point, the target point on the next stage and the points obtained by tracing forward through the linked list of points, found at the target point, to the terminal stage.
6. The optimum trajectory is recovered by tracing through the generated linked list of points in the grid starting from the initial point.

Like the unmodified algorithm, this algorithm will achieve a global optimum over the tessellated grid of points, but is searching over a reduced dimension space. The cost of the algorithm in terms of function evaluations is now given by,

$$\text{total function evaluations} = Q^P \{ (N-3)\ (Q^P - 1) + M \} . \tag{5.86}$$

Care should be taken when comparing the number of function evaluations of the unmodified dynamic programming algorithm with the current algorithm. In the unmodified algorithm a function evaluation consists of the calculation of the energy consumption over a single quartic segment of trajectory. In the modified algorithm a single function evaluation calculates the energy consumption over an entire trajectory of $N-1$ cubic segments. Even if one were to estimate that a function evaluation for the modified algorithm takes on average $N-1$ times longer, the reduction is computational effort remains significant.

Even though the dimension of the algorithm has been cut in half, the computational cost in terms of the number of function evaluations remains high. For example, with the same coarse grid spacing used in the dynamic programming example, which results from setting $P = 3$, $Q = 10$, $M = 100$, and $N = 5$, the required number of function evaluations is 2,098,000. This a large number compared to the approximately 1000 function evaluations it takes to achieve a local optimum using Powell's method. To complete the optimization in a similar time to that achieved by Powell's method would require that 2000 functions be evaluated simultaneously. While theoretically possible, since, for this problem and grid spacing, the reduced dimension dynamic programming algorithm allows for the simultaneous evaluation of nearly one million functions between intermediate stages, it is not a practical approach. Furthermore, the global optimum achieved on this coarse grid spacing is likely to be a worse result than the local optima achieved by Powell's method where then search is not restricted to a discrete grid of points.

If one is willing to give up the ability to achieve an absolute global minimum the computational burden can be further reduced. Instead of tessellating the entire solution space, construct a coarse grid over a portion of it. Apply the modified dynamic program-

ming algorithm to find the optimum over this reduced grid. If the optimum appears on the boundary of the grid, shift the grid to force the optimum through the zeroth point of each stage and repeat the algorithm. If the optimum goes through the zeroth point of each stage, construct a grid with a tighter spacing around the optimum and repeat the algorithm. Repeat this procedure until a desired tolerance is achieved. This method is a local optimization technique so gives up the possibility of finding an absolute global optimum. Note however that it is not local in the sense of a single point as with the descent methods. Instead it searches over local regions. As such, the method has the ability to avoid some local minima in favor of deeper minima.

This approach was tried on the three link problem described in Section 5.2.1. The same starting point was used for the optimization, with three points on each intermediate stage spaced 0.2 radians apart and nine points at the terminal stage spaced 1.0 seconds apart. The criteria for stopping was the similar to that used for Powell's method; namely terminate after successive steps of the algorithm fail to yield a relative improvement better than one percent. When the optimal trajectory reached after a pass of the algorithm went through the zeroth points on each stage of the grid, the grid spacing was halved. After eight passes of the modified dynamic programming, or a total of 13,176 function evaluations, a trajectory with an energy consumption of 156.5 Joules was achieved. This is the same value as that achieved using Powell's method with C2 cubic spline interpolation. The optimized trajectory, shown in phase space form in Figure 5.13, is different than the trajectory of Figure 5.11 achieved by Powell's algorithm, however.

Even though the modified algorithm took approximately 15 times as many function evaluations to achieve a similar result to Powell's algorithm it has some advantages which make it worth consideration. First, many of the function evaluations may be done in parallel. For the present example $Q^P \times (Q^P - 1)$ or 702 function evaluations may be done

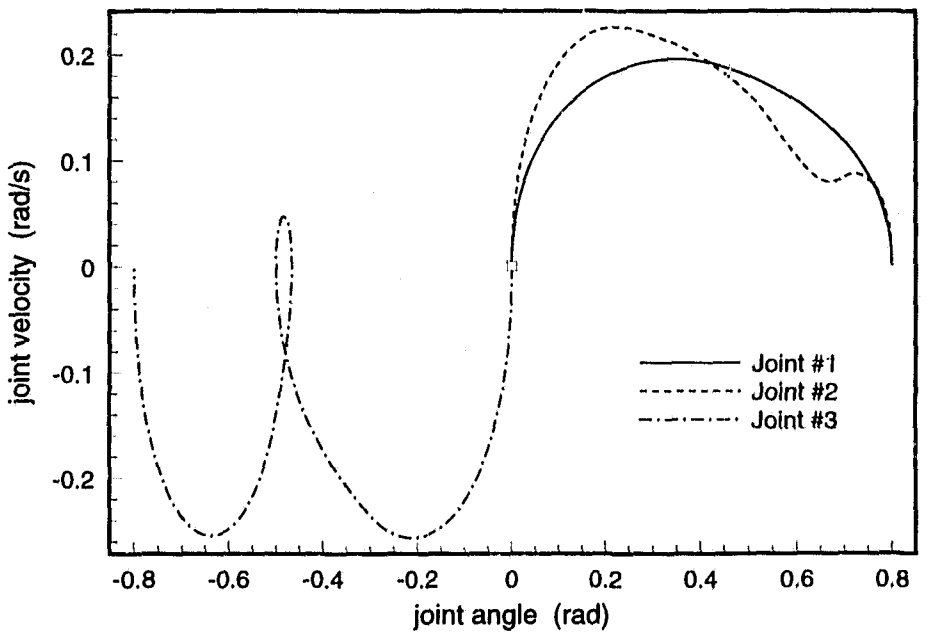


Figure 5.13 Minimum Energy Trajectory for A Three Link Arm Using Modified Dynamic Programming

in parallel between intermediate stages, and $(Q^P - 1) \times M$ or 234 function evaluations may be done in parallel between the last intermediate and terminal stage. If a computer with more than fifteen processors is available, the speed achievable by Powell's algorithm can be surpassed. Such a computer architecture is certainly feasible. Second, and perhaps more importantly, the modified dynamic programming approach is not as sensitive to the user provided initial trajectory. Since it searches for optima over a sequence of local regions of the solution space it should be able to avoid some of the poorer local minima that could trap Powell's method. How well it avoids these local minima depends of course on the size and spacing of the local grid at each pass. To support this hypothesis, the modified dynamic programming algorithm was started at the same 100 randomly generated starting trajectories used in Section 5.2.1 to test Powell's method. A coarse grid of only three points spaced 0.2 radians apart at the intermediate stages and nine points spaced one

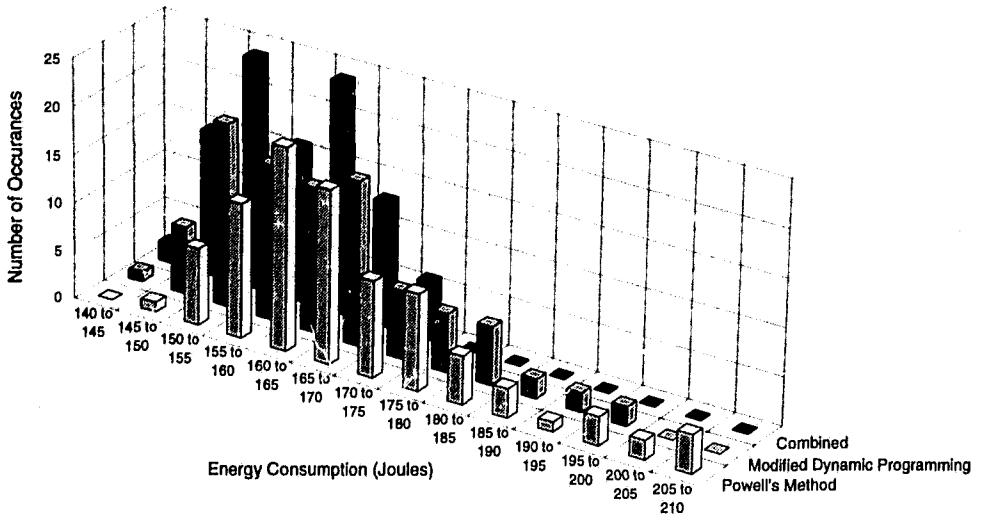


Figure 5.14 Sensitivity of Powell's Method and Modified Dynamic Programming to Initial Trajectc

second apart at the terminal stage was used. A frequency histogram is shown in Figure 5.14 for both Powell's method and the modified dynamic programming approach. Note how the optima reached by the modified dynamic programming approach are more clustered toward the lower energy portions of the histogram. This result is even more obvious if a tighter grid with more points on the intermediate stage is used. The number of function evaluations required to perform the optimization, of course, increases. An even tighter packing of the results at the lower energy levels is achieved if Powell's algorithm is run after the modified dynamic programming algorithm using the trajectory reached by the modified dynamic programming algorithm as its starting point. This is shown on the histogram labelled "Combined" in Figure 5.14. On average the additional cost of performing this operation was 1000 function evaluations.

Table 9 Initial Phase Space Control Vertices for Three Link Arm Example

Joint	Initial Control Vertices $(\theta, \dot{\theta})$ (rad, rad$/s$)
1	(0, 0), (0.1, 0.2), (0.3, 0.3), (0.5, 0.3), (0.7, 0.2), (0.8, 0.0)
2	(0, 0), (0.1, 0.2), (0.3, 0.3), (0.5, 0.3), (0.7, 0.2), (0.8, 0.0)
3	(0, 0), (-0.1, -0.2), (-0.3, -0.3), (-0.5, -0.3), (-0.7, -0.2), (-0.8, 0.0)

5.2.7 Modified Dynamic Programming in Phase Space

In Section 5.1.3, phase space was suggested as a means to represent a manipulator's trajectory. While a number of practical difficulties were mentioned, it is possible that phase space representation can provide a better approximation to the manipulator's true optimal trajectory. As discussed at the end of Section 5.1.3, the largest difficulty with the phase space representation is the selection set of initial control vertices to avoid the book keeping difficulties associated with the addition or elimination of crossing points as the optimization proceeds. A proposed solution was to start the optimization using the time history of the joint angles for each joint. After a trajectory pattern has developed, a good set of phase space control vertices may be chosen. The optimization would then proceed using a phase space representation.

Again consider the three link example problem of Section 5.2.1. The phase space trajectories for each joint achieved by running Powell's algorithm over a set of variables consisting of a number of spline interpolation points in time domain is shown in Figure 5.11. From this result it is reasonable to conclude that a set of starting control vertices in phase space would require intersection with the $\dot{\theta} = 0$ axis only at the endpoints of each joints trajectory. Phantom vertices are used to ensure endpoint interpolation and perpendicular intersection with the $\dot{\theta} = 0$ axis. The starting control vertices are given in Table 9. For this problem a representation has been chosen which uses the same number of

vertices to describe each joint's trajectory. This need not be the case. For initial trajectories where some of the joints require one or more crossing points, these joints will likely require more control vertices to form a joint trajectory. This characteristic of the phase space representation highlights an adjustment which must be made to the modified dynamic programming algorithm as applied to splined trajectories in time domain. In time domain, the joint trajectories are divided into definite time stages, in terms of the scaled time τ, at which interpolation points were placed. Since in phase space, time is contained implicitly in the individual joint trajectories, this is not possible. Instead, each joint is divided into stages with one stage for each control vertex. Since joints don't necessarily have the same number of stages, the stages of one joint's trajectory are not correlated to the stages of another joint's trajectory. As a result, at each stage, each joint cannot be tessellated in one dimension, rather each joint must be tessellated in some manner at each of its own stages. This means that during each pass of the optimization, the optimal motion for each joint over its own grid is determined in turn while the remaining joints are held fixed. While this significantly reduces the number of function evaluations required for one pass of the algorithm to,

$$\sum_{i=1}^{P} [Q(N_i - 3)(Q-1) + (Q-1)] \tag{5.87}$$

where P is the number of joints, N_i is the number of stages for joint i and Q is the number of points on each stage, from that given by equation 5.86, the optimization proceeds by perturbing the trajectories of each joint in turn.

While it has been determined that at each pass of the algorithm, each joint's trajectory will be optimized in turn over its own grid of control vertices, the manner in which the tessellation is performed at each stage of a joint's trajectory must still be determined.

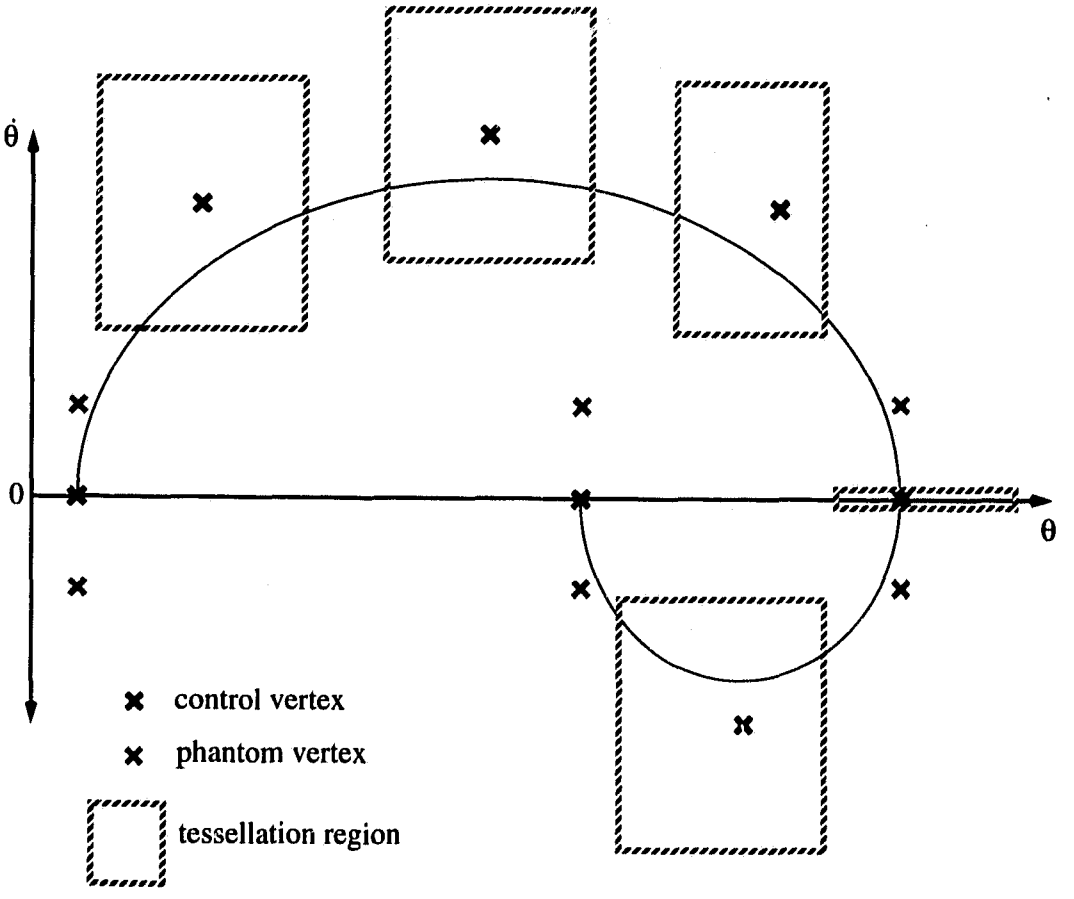


Figure 5.15 Tessellating a Phase Space Trajectory in Two Dimensions

One possibility is to tessellate each of the given initial control vertices in two dimensions corresponding to the joint angle θ and the joint velocity $\dot{\theta}$, with some grid spacing δ. Consider the situation shown in Figure 5.15. Here, the initial trajectory described by a set of control vertices and a set of computed phantom vertices which are necessary to ensure proper intersections with the $\dot{\theta} = 0$ axis. There are seven stages represented by the seven control vertices. The initial and terminal stages are fixed and therefore contain only one point. Each intermediate stage is tessellated into some grid of prescribed spacing in the regions shown. The initial control vertex is labelled the zeroth point of the grid at each

stage. Three points about these tessellation regions should be noted. First adjacent regions do not overlap along the θ coordinate. This prevents the possibility of generating a trajectory which moves negatively along θ when $\dot{\theta}$ is positive or moves positively along θ when $\dot{\theta}$ is negative. Such trajectories are physically impossible. Second the tessellation regions do not cross the $\dot{\theta} = 0$ axis. This prevents the trajectory from crossing this axis without the necessary phantom vertices to ensure the correct conditions. Finally, intermediate stages which lie on the $\dot{\theta} = 0$ axis must remain on this axis to ensure the proper crossing conditions. Therefore the grid at these stages is generated by tessellating in only one dimension; namely along the θ coordinate.

An alternative technique, which may reduce the number of points at each stage involves tessellation along only one dimension. Rather that just tessellate along the θ coordinate for crossing points and along the $\dot{\theta}$ coordinate for the other intermediate stages however, tessellate along a line through a stage's initial control vertex and perpendicular to the line segment jointing the zeroth control vertices on either side of the stage. For stages which are crossing points tessellate only along the θ coordinate. This allows for somewhat radial expansion or contraction in localized regions of the phase space trajectory. This type of movement was noted in observing the progression of some of optimization algorithms in the time domain when intermediate trajectories were viewed in phase space. For example, in Figure 5.11 the initial and optimized trajectories found by Powell's method in time domain show this characteristic for the example three link arm problem. This one dimensional approach to generating the grid is shown pictorially in Figure 5.16. The segments of the lines along which the tessellation occurs is restricted by the same three constraints that were required for the two dimensional tessellation regions previously described.

With the means of grid creation and grid constraints in place a single pass of the modified dynamic programming algorithm in phase space may be summarized as follows.

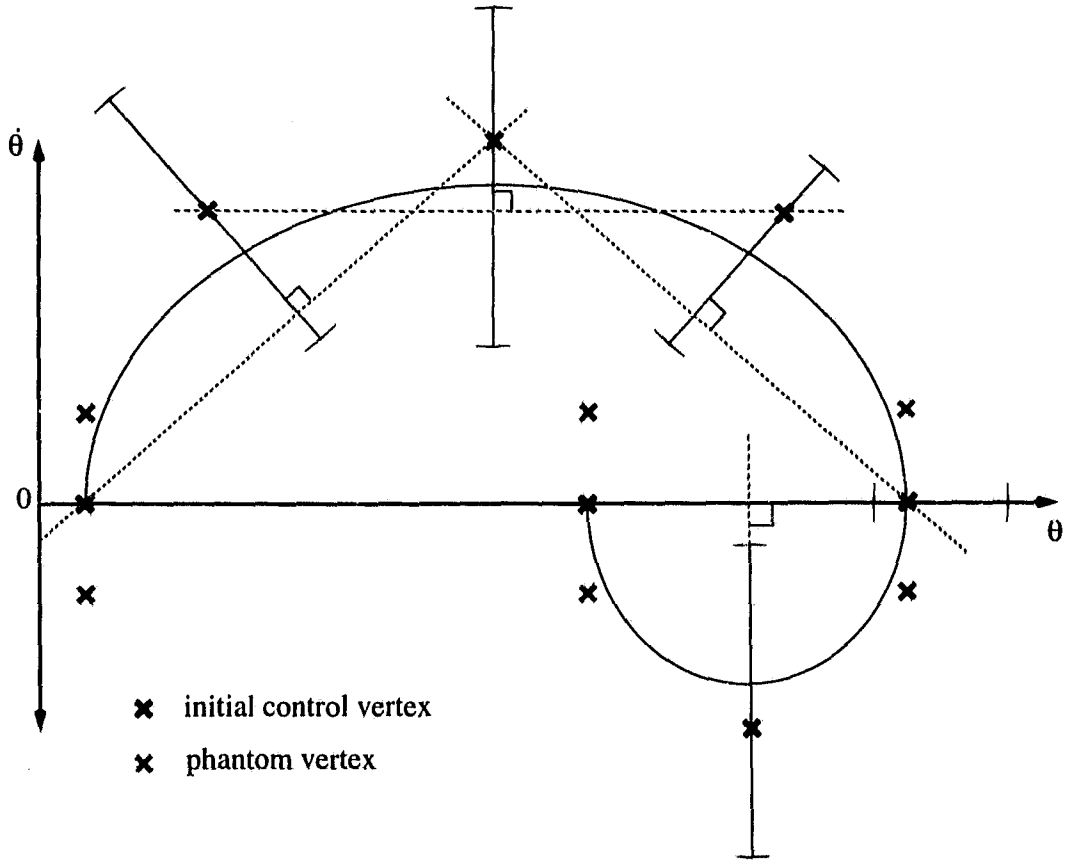


Figure 5.16 Tessellating a Phase Space Trajectory in One Dimension

1. Calculate the energy consumption of the initial trajectory. Save this value as the "trajectory energy". Each joint's initial trajectory is constructed by using the set of control vertices specified by the user or some starter algorithm. Phantom vertices are computed and added as required to provide the proper intersection conditions with the $\dot{\theta} = 0$ axis.
2. Set the joint number $i = 1$.

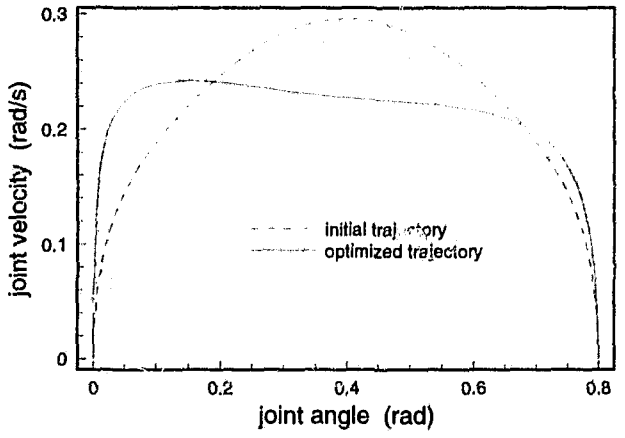
3. For joint i divide the joint trajectory into N_i stages, numbered zero through $N_i - 1$. At each stage a discrete grid of Q possible states is generated. Number these points on the grid zero through $Q - 1$. The initial and terminal stages have one point each corresponding respectively to the known initial and terminal states. When constructing the grid of points, the zeroth point of each stage is set to the initial control vertex specified by the user or the starter algorithm. The remaining points at each stage of the grid are determined by successively adding offsets of $\pm\delta$ to the zeroth point along a line through the zeroth point and perpendicular to the line segment connecting the zeroth points of the stages on either side of the current stage, where δ is the grid spacing. If the current stage is a crossing point, the offsets are added along the $\dot{\theta} = 0$ axis. Added points which would violate the constraints on the tessellation region are clipped.
4. Store the "trajectory energy" in the structure associated with the zeroth points of the terminal stage $(N - 1)$ and the penultimate stage $(N - 2)$. Initialize a set of linked lists by setting pointers from each point on stage on $N - 2$ to the point (zeroth point) on stage $N - 1$. Mark the end on the linked lists by setting the pointer associated with the single point on the terminal stage to the null pointer.
5. Set the stage number to $j = N_i - 2$.
6. For each point, one through $Q - 1$, on stage j determine the minimum energy trajectory from a set of trajectories constructed from the zeroth points on all previous stages, the current point, each point on the next stage and points determined by tracing the established linked list to the terminal stage. Store this minimum energy consumption in the structure for the current point and set a pointer to the corresponding point from the next $(j + 1)$ stage. Take the smallest of these computed minimum energy values from the current stage and store it in the zeroth point of stage $j - 1$. Create a pointer from the zeroth point of stage $j - 1$ to the point on stage j corresponding to this smallest minimum energy trajectory.
7. Decrement j by one.
8. If $j > 0$, go to step 6.

9. The optimized trajectory for joint i is recovered by tracing through the generated linked list of points in the grid starting from the initial point.
10. Assign the points recovered from the link list to be the zeroth points of each stage for joint i.
11. Assign the energy consumption of this recovered trajectory to the "trajectory energy".
12. If $i < P$, where P is the number of joints, go to step 3.
13. The optimized trajectory at this point, after one pass of the basic algorithm, is contained in the zeroth points of the stages for each joint. The reduced energy consumption is stored as the "trajectory energy".

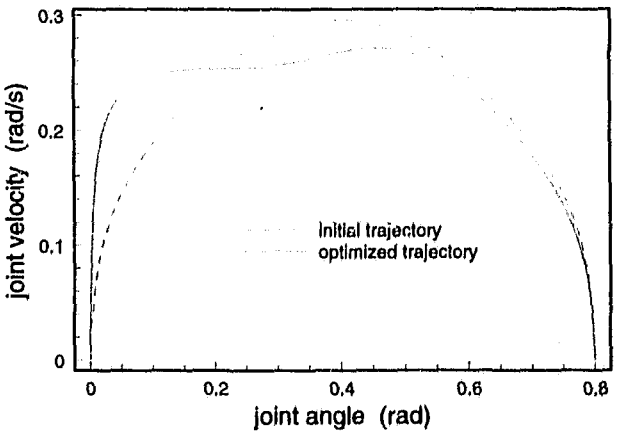
As with the modified dynamic programming algorithm in time domain this basic algorithm is applied in multiple passes over an initially coarse but modifiable grid. If the optimum appears on the boundary of any of the joints' grid, shift the grid to force the optimum through the zeroth point of each stage and repeat the algorithm. If the optimum goes through the zeroth point of each stage, construct a grid with a tighter spacing (reduce δ) around the optimum and repeat the algorithm. Repeat this procedure until a desired tolerance is achieved.

This procedure was applied to the sample three link problem. The initial trajectory was specified by the initial control vertices of Table 9. The grid consisted of seven stages for each joint with three points at each intermediate stage. The grid spacing was $\delta = 0.1$. When the optimized trajectory on a given pass went through the zeroth point on each stage the grid spacing was reduced by a factor of five. After nine passes of the basic algorithm, or 702 function evaluations, the energy consumption of the trajectory was reduced from an initial value of 237 Joules to 135 Joules. This is a thirteen percent improvement over the result achieved by both Powell's method and modified dynamic programming in the time domain. Furthermore this improvement was realized with fewer function evaluations than either of the other methods. Figure 5.17 shows the initial and optimized trajectories for

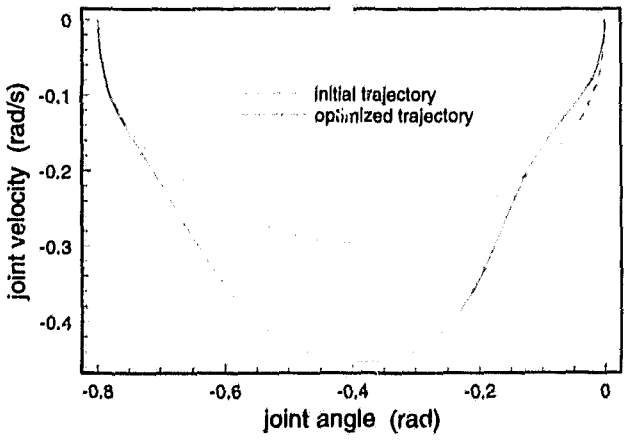
each joint achieved using this method. For this problem, phase space does indeed provide a superior representation. The use of the modified dynamic programming approach in this space maintains the ability to do some of the function evaluations in parallel. In phase space, $Q(Q-1)$ functions may be evaluated in parallel when working between intermediate stages.



joint 1



joint 2



joint 3

Figure 5.17 Minimum Energy Trajectory for a Three Link Arm Using Modified Dynamic Programming in Phase Space

Chapter 6
Experimental Verification

In Chapter 4 a mathematical model was developed for the Reis V15 industrial manipulator so that the trajectory optimization could be tested on a real manipulator. For both the development of the model and the verification of the optimization methods, low level control of the manipulator and access to signals such as motor currents and joint velocities is required. As supplied by the manufacturer, the robot provides none of this utility.

6.1 Reis V15 Experimental Setup

In order to allow the low level control and signal access necessary to experimentally verify the simulated results, a control and interface system was constructed around the Reis V15 manipulator. This is system is called the Telerobotic Experimental STation (TEST). The Reis V15 manipulator is controlled by a multiprocessor VME bus computer running the VxWorks multiprocessor, multitasking operating system. The control computer is attached to the campus ethernet, thereby enabling user interaction via a Sun workstation running X-windows. See Figure 6.1.

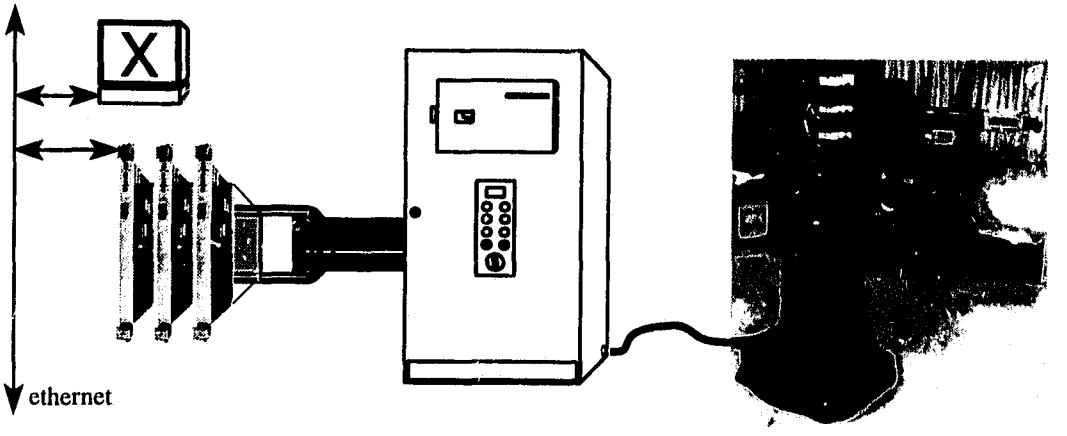


Figure 6.1 TEST Hardware

6.2 Drive Configuration

The Reis V15 manipulator, illustrated in Figure 6.2, is a 6R electrically actuated industrial manipulator. Each axis of the robot is driven by a permanent magnet DC motor through a harmonic drive gear reducer with a ratio of 100:1. Mounted on the shaft of each motor are a tachometer for velocity feedback and a four channel encoder for position feedback.

The drive system for each motor is shown in Figure 6.3. A digital drive signal from the VME bus control computer is sent to the Reis I/O card responsible for driving the joint of interest. There are two Reis I/O cards, each capable of driving four joints. For the V15 manipulator the first card drives the first four joints of the manipulator. The second card drives joints five and six. The digital signal is converted to an analog voltage and passed to a pulse width modulated (PWM) servo amplifier card which drives the motor. Each motor has its own amplifier card. The servo amplifiers use analog PID velocity controllers to drive the motors at a speed proportional to the input voltage. The Reis V15 servo amplifier cards are described in more detail in Section 4.3.2, where the actuator model is developed.

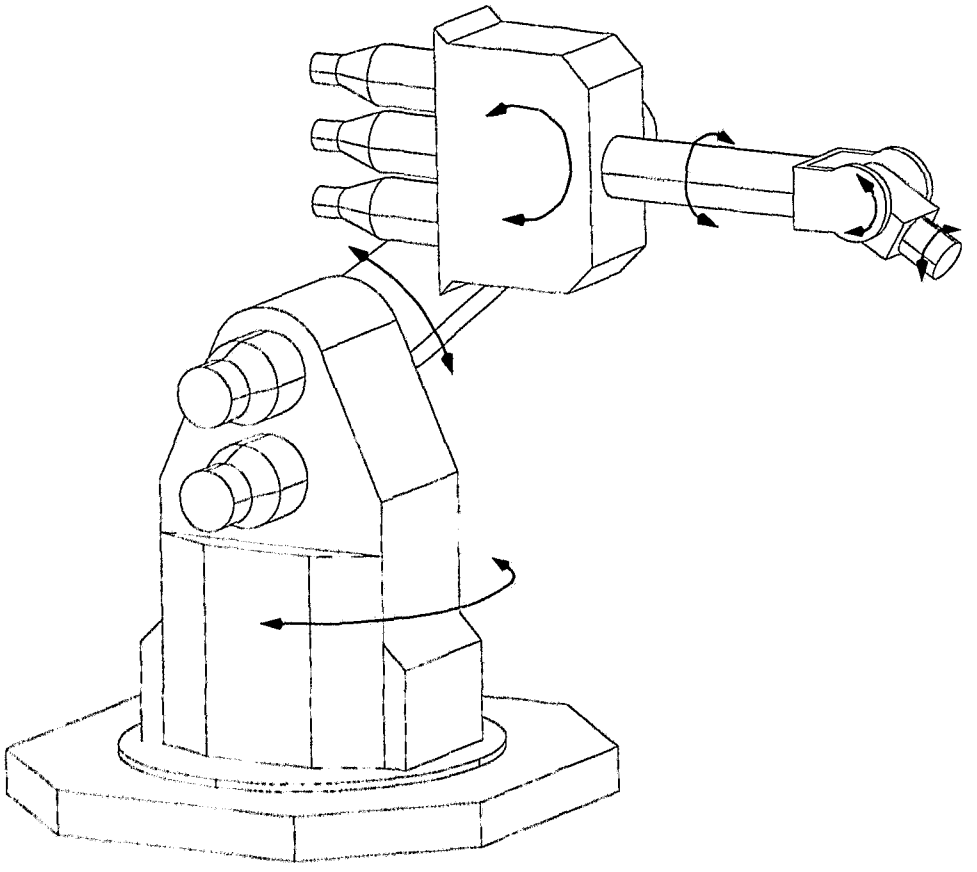


Figure 6.2 Reis V15 Industrial Manipulator

The trains of pulses from the four channel optical encoders drive counters on the Reis I/O cards to keep track of position. The control computer reads these counters to close the digital position control loop which updates each motor's commanded velocity.

As shipped by the manufacturer, the robot is controlled by a high level proprietary command language called RobotStar II, which is entered through a teach pendant. This language provides no low level control or signal access capabilities. Fortunately, the structure of the Reis system allows the manufacturer's control system to be replaced by one of my own design with a minimum of hardware modifications. To create this system, the two

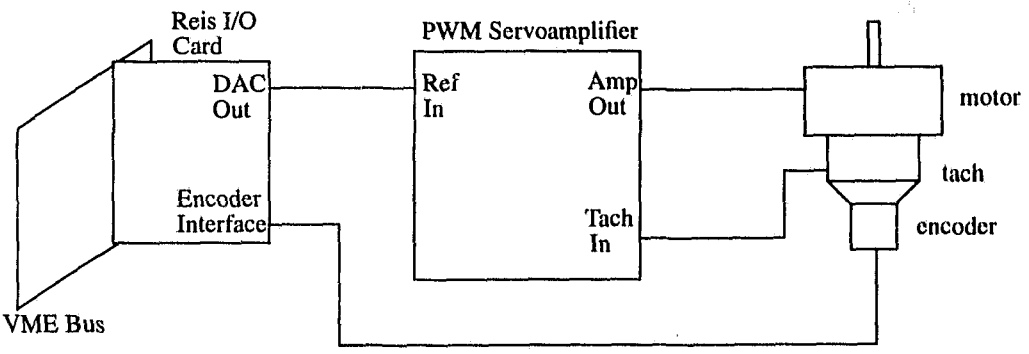


Figure 6.3 Reis V15 Actuator Configuration

Reis I/O cards were moved from the VME backplane with the Reis digital controller into a multiprocessor VME computer. To control the robot, the computer writes the low level digital signals to the I/O cards and reads back the robot status and sensor information. To do this requires a memory map of the Reis I/O cards. This memory map, given in Appendix C, was obtained by observing the signals on the VME bus with a logic analyzer when the robot was under the control of the original Reis controller.

6.3 Mechanical Configuration

When writing the drive signal to the manipulator it is important to realize that the DC motors are being commanded, not the manipulator's joints directly. The configuration of the transmission systems from the motors to the joints means that for some axes a positive rotation of the motor yields a negative rotation of the joint axes. The nature of the transmissions results in some kinematic coupling between some of the joints. This effect must be taken into account if accurate control is to result.

The transmission for the first axis is the most straightforward. The drive motor is located inside the base of the robot's first link and drives directly into the input of the har-

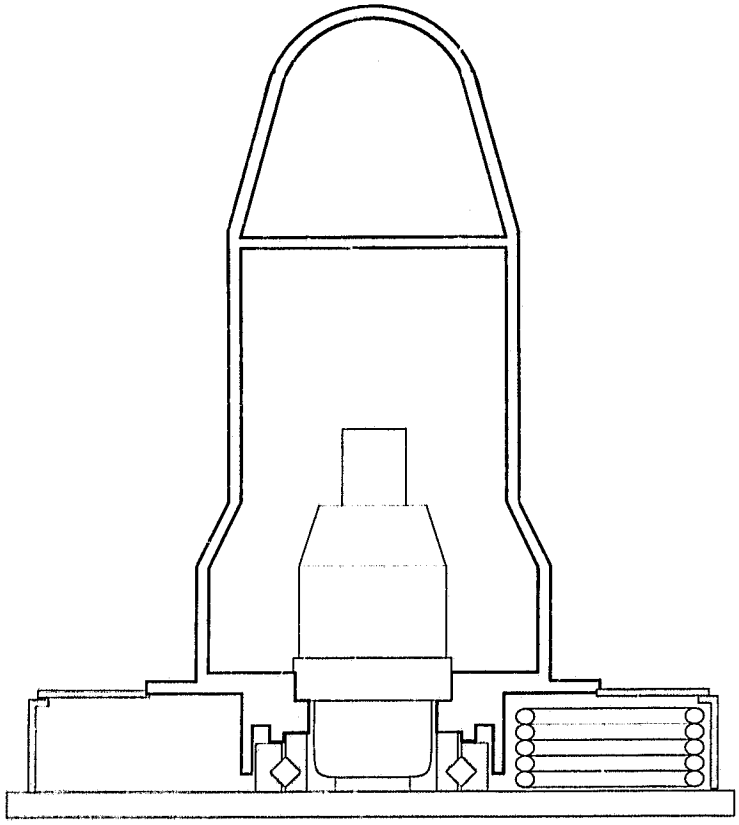


Figure 6.4 Mechanical Configuration of the First Axis

monic drive gear reducer as shown in Figure 6.4. The motor casing is connected to the link and the output of the harmonic drive gear reducer is connected to the robot's fixed base. Since the output of a harmonic drive rotates in the opposite sense to its input wave generator, the first link rotates in the same direction as that of the drive motor at 1/100'th of the motor speed.

The configuration of the transmission for the second axis is illustrated in Figure 6.5. The drive motor for the joint is mounted on the side of the first link just below the drive motor for the third joint which is concentric with the axis of rotation for the second joint. A timing belt transmits the output of the motor to the wave generator of the har-

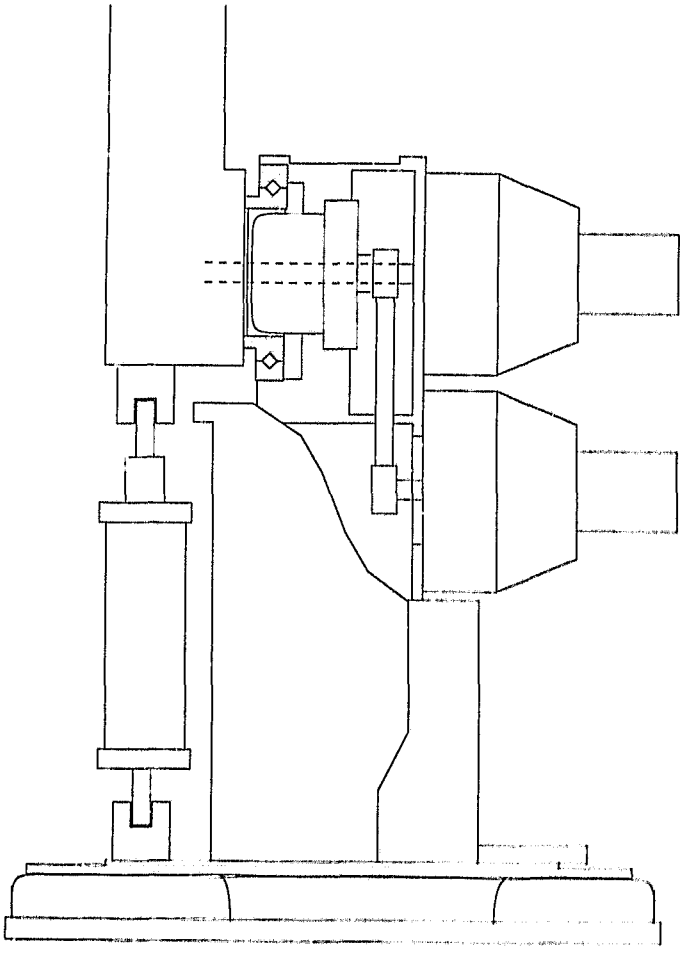


Figure 6.5 Mechanical Configuration of the Second Axis

monic drive located on the joint's axis. The rigid spline of the harmonic drive unit is fixed to the first link and its output flex spline drives the second joint. With this arrangement, the second link rotates in the opposite sense to its drive motor at 1/100'th of the speed.

The third joint of the manipulator is driven by the upper motor mounted on the first link as shown in Figure 6.6. The output shaft of the motor passes through the center of the second joint and its harmonic drive. A timing belt in the second link of the arm transmits the power to the third joint's harmonic drive, which is located on the axis of rotation of the

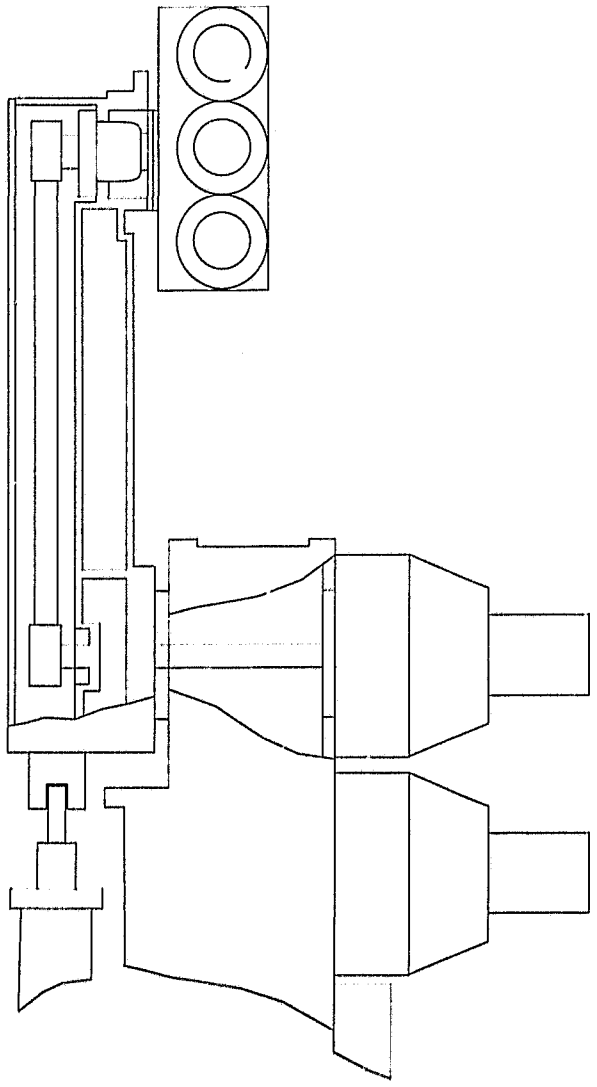


Figure 6.6 Mechanical Configuration of the Third Axis

third joint. Since this drive motor is attached to the first link rather than the second link, motion of the second link causes motion of the third link even when its drive motor is held stationary. This kinematic coupling may be described as follows. If the second joint is held stationary, the output rotation at the third link will be 1/100'th of the third drive motor's rotation, but in the opposite sense. If the second joint is moved and the third drive motor is held stationary, then the output rotation of the third joint will be 1/100'th of the rotation of

the second joint in the same direction. The sum of these two effects yields the motion of the third joint and is described by the following equation.

$$\omega_{j,3} = \frac{1}{100}(\omega_{m,3} + \omega_{j,2}) = \frac{1}{100}(\omega_{m,3} - \frac{\omega_{m,2}}{100}) \quad (6.1)$$

where $\omega_{j,i}$ is the rotation rate of joint i and $\omega_{m,i}$ is the rotation rate of motor i.

Joints four through six form a spherical wrist configuration. The arrangement of the transmission system is shown in Figure 6.7. The output of the fourth motor is connected via a timing belt to the input wave generator of a harmonic drive on the fourth joint axis. The output of the fifth motor is coupled, with a timing belt, to a hollow drive shaft which passes through the centre of the fourth joints harmonic drive. This hollow shaft drives a harmonic drive mounted on the fifth joint axis via a bevel gear and a timing belt. The output shaft of the sixth motor passes through the hollow drive shaft of the fourth joint. A bevel gear followed by a timing belt and a second bevel gear transmits this shaft's power to the harmonic drive mounted on the sixth joint. Since the drive motors for joints five and six are not mounted on links four and five respectively, kinematic coupling between the respective joint motions again exists. The following equations describe the relations between the motor and joint velocities for the wrist joints.

$$\omega_{j,4} = \frac{\omega_{m,4}}{100} \quad (6.2)$$

$$\omega_{j,5} = \frac{1}{100}(\omega_{m,5} - \omega_{j,4}) = \frac{1}{100}(\omega_{m,5} - \frac{\omega_{m,4}}{100}) \quad (6.3)$$

$$\omega_{j,6} = \frac{1}{100}(\omega_{m,6} - \omega_{j,5} - \omega_{j,4}) = \frac{1}{100}(\omega_{m,6} - \frac{1}{100}(\omega_{m,5} + 0.99\omega_{m,4})) \quad (6.4)$$

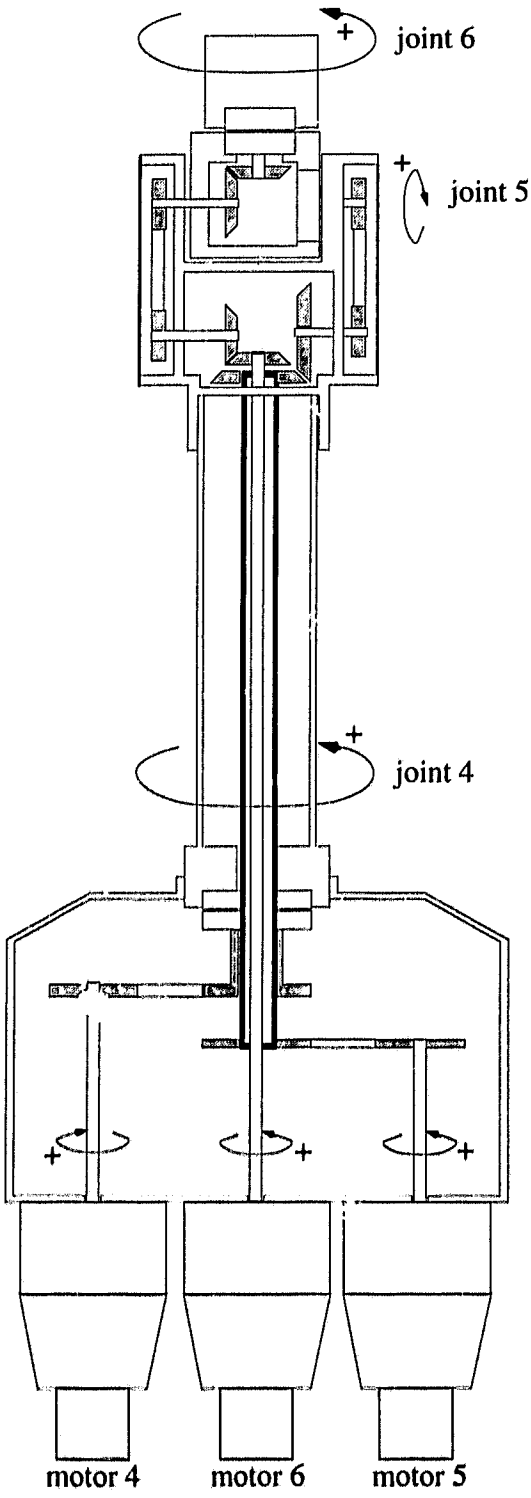


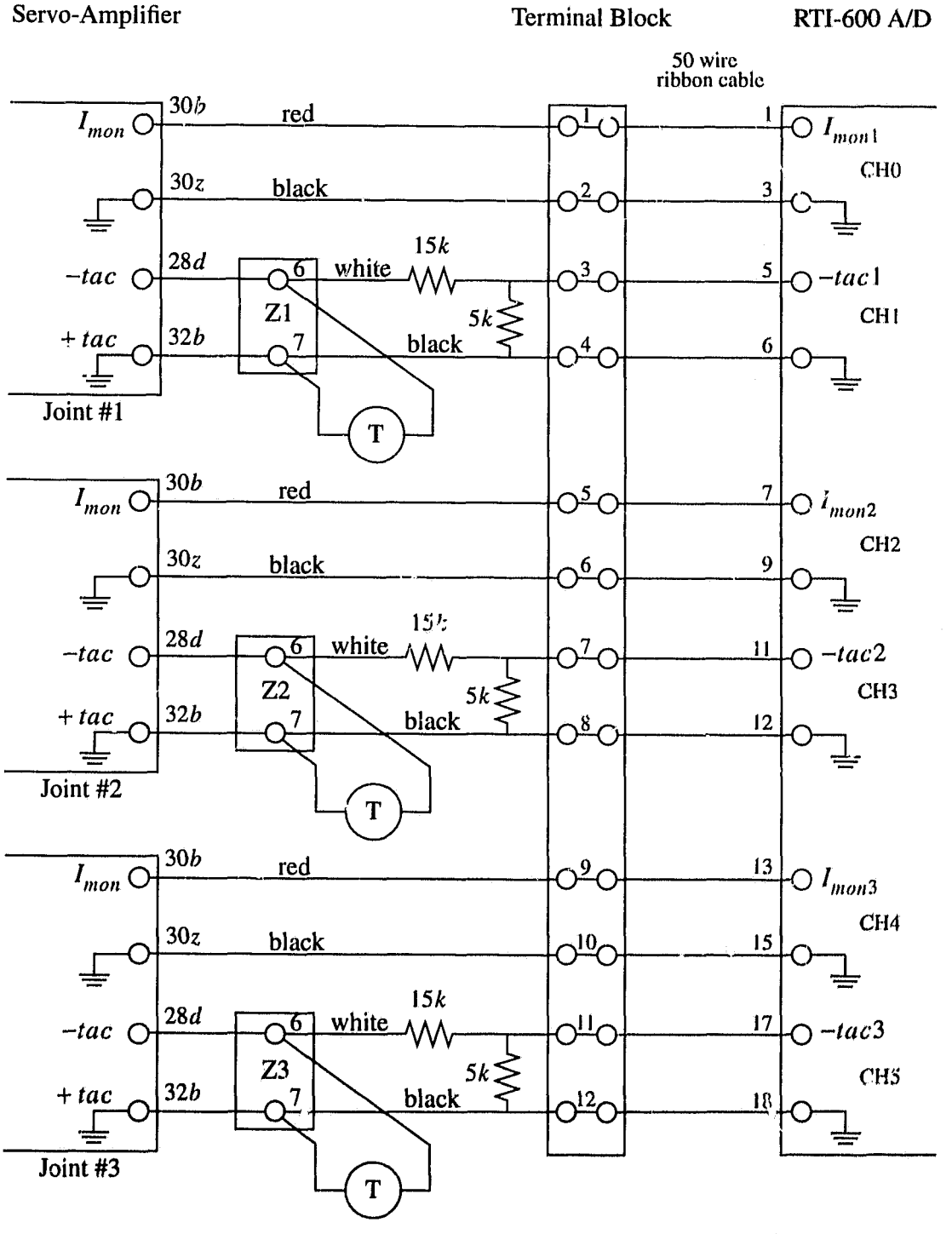
Figure 6.7 Mechanical Configuration of the Reis V15 Wrist

6.4 Access to Motor Velocity and Current Signals

In order to determine the reduction in energy consumption obtained by planning optimized trajectories the energy consumed in the actuators of the real manipulator for a particular trajectory must be estimated. To do this requires access to motor velocity and motor current information. This armature current information is available by reading the voltage across pins 30b and 30z on each servo amplifier card. The instantaneous current in Amperes in given by multiplying this current monitor voltage (I_{mon}) by four. The motor velocity information is available by reading the tachometer voltages for joints one, two, and three, across pins six and seven of block connectors Z1, Z2, and Z3, respectively. These block connectors are located beneath the Reis servo amplifier cards inside the Reis controller box. The tachogenerator constant is 10 V/1000 rpm. Since the motors may turn at as much as 4000 rpm, a voltage divider is used to reduce the voltage by one quarter so that it has a suitable range for analog to digital conversion. These voltages are pulled off onto a connector block as shown in Figure 6.8. The terminals on the connector block are then connected to the indicated channels on a VME bus RTI-600 analog to digital converter.

The RTI-600 is a 32 channel 12 bit analog to digital converter. It occupies 256 bytes in the VME short I/O space. In this application, the address selection jumpers assign a base address of FFFFB400. The complete set of user configurable parameters is provided in Appendix D. The used portion of the memory map and its function are shown in Figure 6.9. A channel is selected by writing an unsigned short (16 bit) integer corresponding to the desired channel (0 to 31 decimal) to the base address. This initiates the A/D conversion. The busy bit (D15), is used to indicate when the analog to digital conversion for the selected channel is complete. When the busy bit goes low, the 12 bits of data is read into an unsigned short integer variable. Since the A/D conversion is 12 bit, this variable

Figure 6.8 Armature Current and Tachometer Measurement Connections



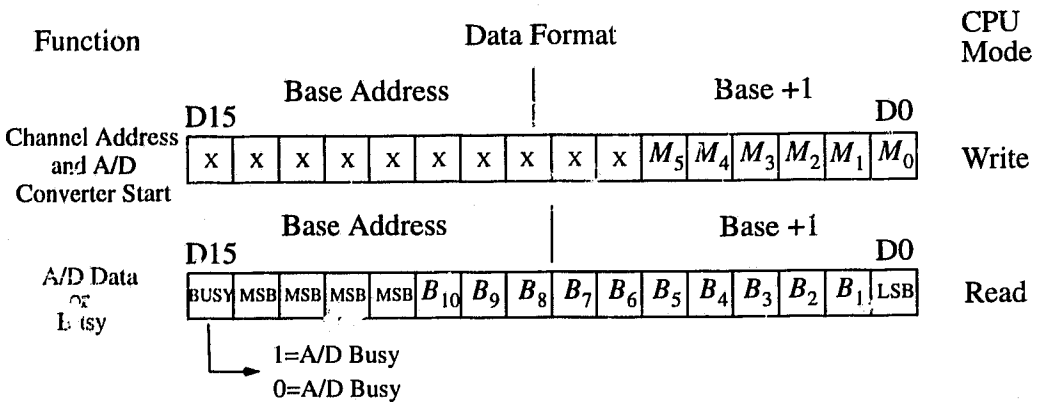


Figure 6.9 RTI-600 A/D Converter Memory Map

will take on a value between 0 and 4095. The reading can be converted to voltage by noting that this integer range linearly covers the ± 10 V input range selected for the A/D card.

6.5 Force Sensor

Also available to the user is a six axis force/torque sensor. The force sensor may be used as input to a model reference impedance controller which was developed during the course of the thesis work. This controller allows for the planned path to be modified in the event an object, which was not known during the trajectory planning phase, is encountered during the trajectory execution. Extensive modifications were made to the force sensor to make it useful for real-time robotic control. These modifications are described in Appendix E. The implementation of model reference impedance control is described in Appendix F.

6.6 Software Organization

The manipulator is monitored and controlled by a system of software running under Version 4.1 of the VxWorks multiprocessor, multitasking real-time operating system. The system runs on three MVME147-1 single board microprocessor cards. With the VxWorks operating system, the control software is set up as a series of tasks running on each of the processors. Each processor runs one task at a time according to a priority scheduler. Tasks may exchange information with each other by use of a number of message passing strategies. If the tasks exchanging information are on the same processor, information is exchanged using semaphores or message queues.

Semaphores provide the fastest intertask communication mechanism in VxWorks. A semaphore is not used to exchange data between tasks but rather as the primary means of mutual exclusion and task synchronization. A semaphore may be viewed as a cell in memory that can be full or empty. When a task takes a semaphore, using a call to *semTake()*, the outcome depends on whether the semaphore was full or empty at the time of the call. If the semaphore was full, then the semaphore is made empty and the task execution continues immediately. If the semaphore was empty, then the task is put on a queue of blocked tasks and enters a state of pending on the availability of the semaphore. The length of time the task will wait for the semaphore to be filled depends on a parameter passed to the *semTake()* task which specifies the time in system ticks. If the task succeeds in taking the semaphore in the allotted time, *semTake()* returns *OK*; if it times out before successfully taking the semaphore, it returns *ERROR*. A time-out value of *WAIT_FOREVER* means wait indefinitely; a value of *NO_WAIT* means do not wait at all. For mutual exclusion, semaphores interlock access to shared resources. In this technique, a semaphore is created to guard the resource. Initially the semaphore is full. When a task wants to access the resource, it must first take that semaphore. As long as the task keeps the sema-

phore, all other tasks seeking access to the resource will be blocked from execution. When the task is finished with the resource, it gives back the semaphore allowing another task to use the resource. When used for task synchronization, a semaphore may represent a condition or an event that a task is waiting for. Initially the semaphore is empty. A task or interrupt service routine signals the occurrence of an event by giving the semaphore. Another task waits for the semaphore by calling *semTake()*. The waiting task will be blocked waiting until the event occurs and the semaphore is given.

Message queues are the primary means by which different tasks on the same CPU exchange information. Message queues allow a variable number of messages, each of variable length, to queued in a first in first out order. Any task or interrupt service routine can send messages to a message queue. Any task can receive messages from a message queue. Full-duplex communication between two tasks generally requires two message queues, one for each direction. A task or interrupt service routine sends a message to a message queue with *msgQSend()*. If no tasks are waiting for messages on that queue, the message is simply added to the queue's buffer of messages. If any tasks are already waiting for a message from that message queue, the message is immediately delivered to the first waiting task. A message is received from a message queue with *msgQReceive()*. If messages are already available in the message queue's buffer, the first message is immediately dequeued and returned to the caller. If no messages are available, then the calling task will block and be added to a queue of tasks waiting for messages. Both *msgQSend()* and *msgQReceive()* take time-out parameters. When sending a message, if no buffer space is available to queue the message, the time-out specifies how many ticks to wait for space to become available. When receiving a message, if no message is immediately available, the time-out specifies how many ticks to wait for a message to become available. As with semaphores,

the value of the time-out parameter may have special values of *NO_WAIT* meaning to return immediately or *WAIT_FOREVER* meaning to never time out of the function.

Message passing between tasks on two separate processors or between one of the tasks and a program running on the host X-Windows workstation across an ethernet connection is implemented using stream sockets. A task creates a stream socket and binds it to a particular port number. Another process, on any other CPU on the VME backplane or host machine connected to the ethernet, can create another stream socket and specify that it be connected to the first socket by specifying its Internet address and port number. Once the sockets have been connected, there is a virtual circuit set up between them allowing error-free socket-to-socket communications using the *send()* and *recv()* commands.

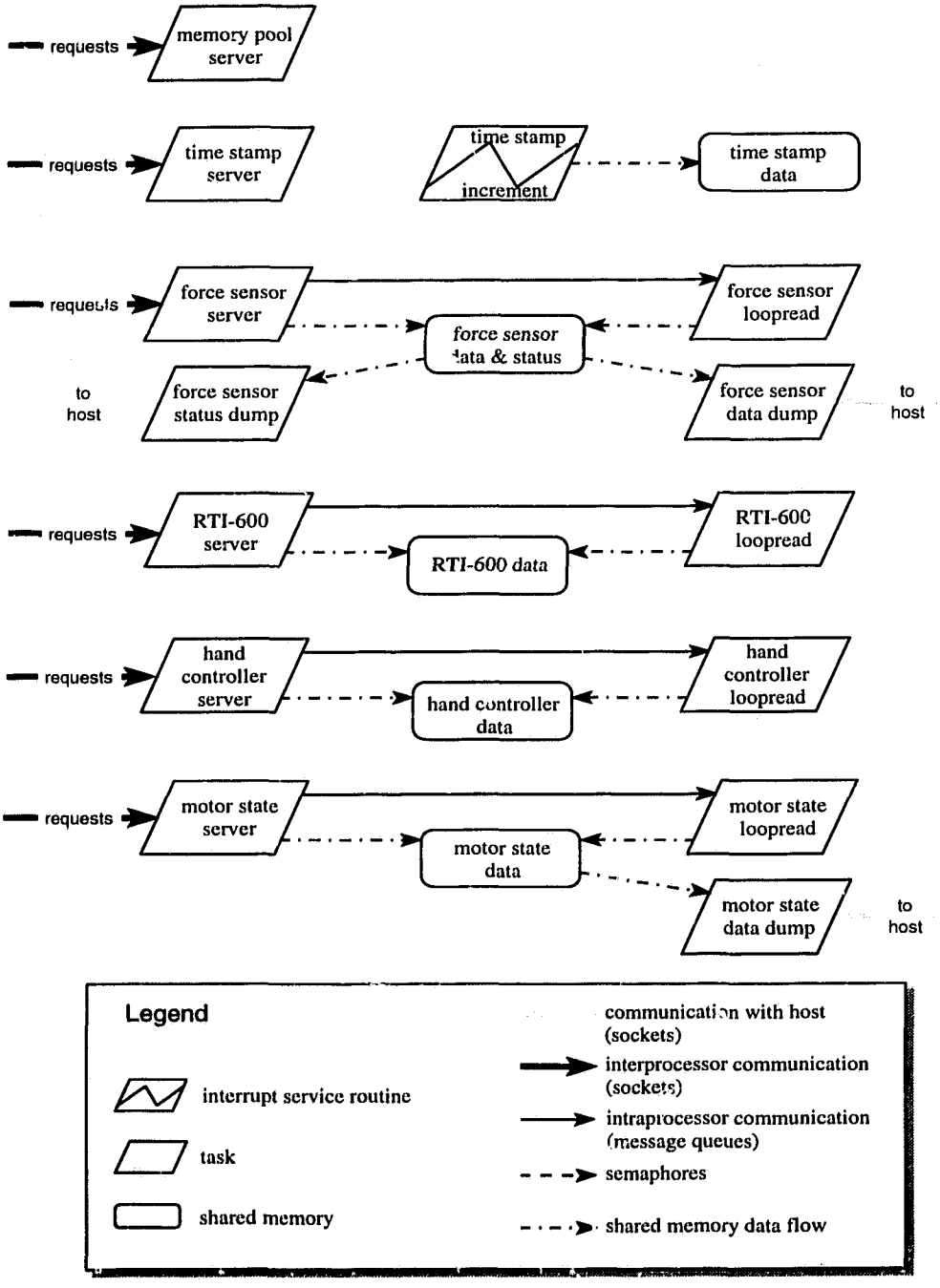
More detailed information on the use of semaphores, message queues and sockets can be found in the VxWorks Programmer's Guide[88] and the VxWorks Reference Manual.[89] Additional information on sockets can be found in Unix Programer's Manual.[83]

6.6.1 CPU_0

CPU_0 is used primarily to control and interface to peripheral devices. It also provides facilities to allocate and control global memory pools which may be shared between tasks on different processors and to provide a global time stamp for the collection of data. The tasks, communication and data flow on CPU_0 are shown in Figure 6.10.

The memory pool server allocates and frees global shared memory space as requested by other tasks. The shared memory allocated is accessible from any processor on the VME backplane. Writing this server was necessary to increase the speed of interprocessor communication. Exchanging time critical data between processors using the socket protocol provided by the VxWorks operating system proved too slow due to the

Figure 6.10 Tasks and Interprocess Communication on CPU_0



excessive overhead associated with the socket protocol. For example, providing force data read using CPU_0 to an impedance controller running on CPU_2 is done using shared memory. It should be noted that in later releases of the VxWorks operating system, (starting with V5.1 beta) additional utilities which used shared memory objects to provide higher speed interprocessor communication are provided as a cost added option.

The time stamp server uses the global memory pool server to allocate a piece of global shared memory in which to keep an integer count. After zeroing the count, it initializes an on board hardware timer and starts it. The timer generates an interrupt every millisecond. The time stamp increment interrupt service routine responds to the interrupt and increments the counter by one. The time stamp server awaits requests from other tasks for the address of the time stamp counter. Tasks use the counter data to time stamp any data that they collect.

Peripheral devices controlled by CPU_0 include the wrist mounted force sensor and the RTI-600 A/D card through which access to a hand controller and motor velocity and current information are obtained. The software structure for each device is similar as can be seen from Figure 6.10. Each device has a server task whose job it is to respond to and service requests from other tasks on the system or client tasks.

For the force sensor, the server initializes and blocks waiting for a client task to connect to its socket and send a request. When the request is received, the server jumps to one of a number of subroutines to service the request depending on its coded value. Valid request for the force sensor are as follows.

OPEN:

The address of the global time stamp data is obtained from the time stamp server. The time stamp will be used to mark each piece of collected data. The shared memory pool in which the force sensor data and status will be stored is allocated by sending a memory allocation request to the shared memory pool server. Each of five possible tool transformations is set to the identity matrix. The zeroth tool is selected. All bias values are set to zero. Analog to digital conversion is started by reading from *AD_Start* on the A/D converter controller card. The device is marked as open in the force sensor status area of the shared memory pool.

CLOSE:

The device is marked as closed in the force sensor status area of the shared memory pool. The analog to digital conversion is stopped by reading from AD_Stop on the A/D controller board. The shared memory pool is freed using the shared memory pool server.

FTSREAD:

After checking that the force sensor device is open, exclusive access is obtained to the force sensor data on the A/D controller card by reading from RD_Start. The eight strain gauge voltages are copied into local memory. Exclusive access is released by reading from RD_Stop. The raw data is premultiplied by the calibration matrix and converted into units on Newtons and Newton meters. The force sensor bias and the tool transformation are applied. A semaphore is posted to indicate that new force data is available.

LOOPREAD:

This request either enables or disables the force sensor loopread task. When enabled, this task reads the strain gauge data continuously and updates the force sensor

data. This provides much faster updating of the force data than is possible by forcing a task to make an FTSREAD request each time new force data is required. When a task opens the force sensor it is provided with the shared memory address of the force sensor data. After enabling loopread, it need only read the data from the address as required.

BIAS:

This request is used to bias or unbias the force sensor. To bias the force sensor, fifty readings are taken. The average reading is saved in the bias vector and is subtracted from all subsequent force sensor readings. To unbias the force sensor the bias vector is set to zero.

TOOL:

This request includes a tool number from zero to four and an associated 4x4 tool transformation matrix. The tool transformation is stored in a tool transformation database associated with the provide tool number.

SELECT:

This request selects a tool from the tool transformation database. The tool transformation of the selected tool will be applied on subsequent read operations

default:

In the case that the request matches none of the valid request, a flag indicating an invalid request is returned to the client task.

The force sensor is provided with two additional tasks to collect and send force sensor status and data to the host workstation. Since the ethernet connection between the host machine and the VME backplane is relatively slow, the host machine collects data by sending a request for an integer number of data points. The force sensor data dump task

collects the requested number of data points, storing the data in on board memory. After the requested number of data points are collected, the data is relayed, record by record, back to the host machine.

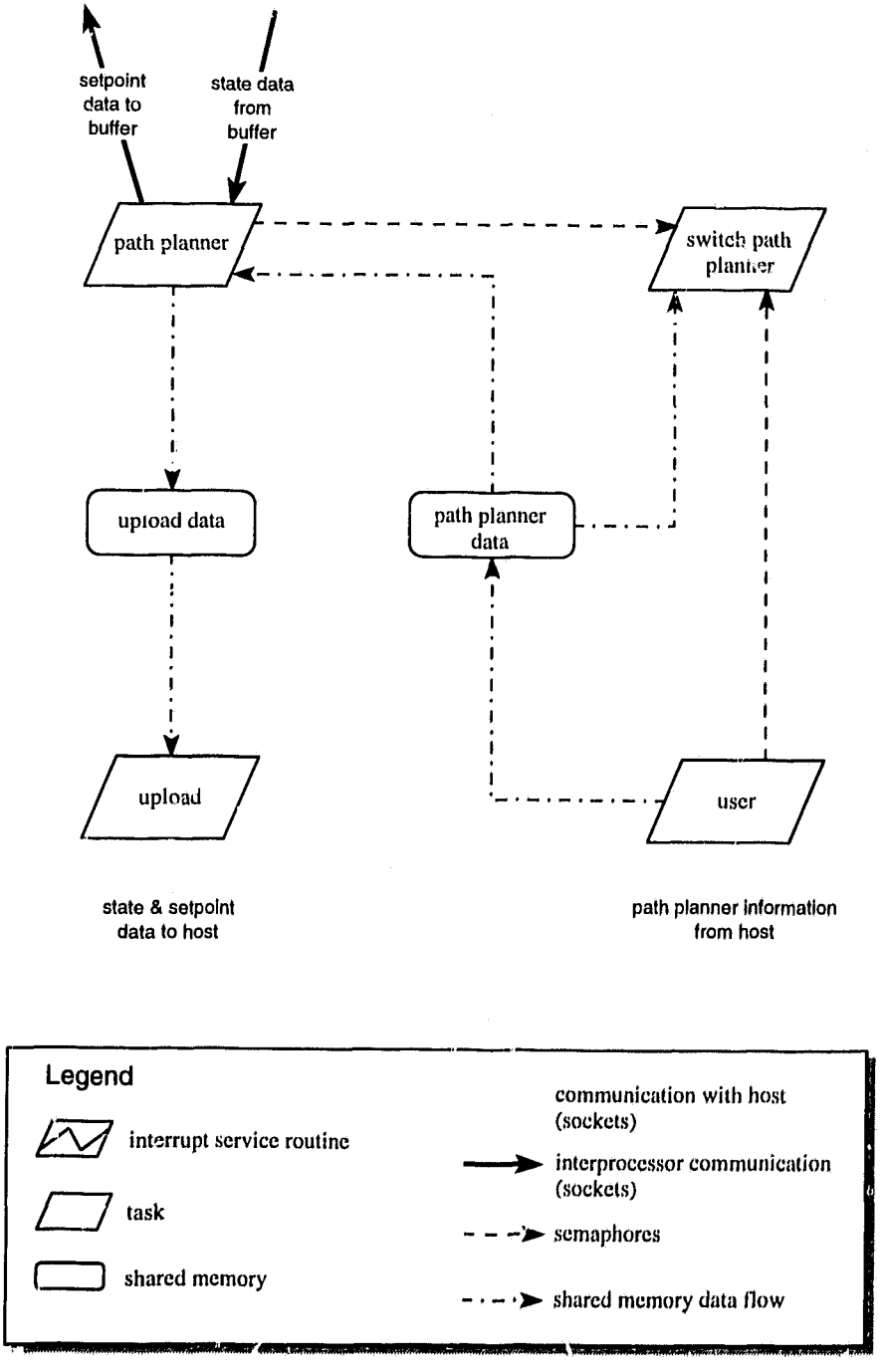
The task groups which provide the interfaces to RTI-600 A/D card, the hand controller, and the motor's state data have a similar structure to that of the force sensor. For these servers valid requests are; OPEN, CLOSE, READ and LOOPREAD. The RTI-600 A/D card and the hand controller have no tasks to pass their data along to the host machine. The data from these devices is used internally. Any task running on one of the three CPUs on the VME backplane can be given access to the shared memory areas containing the data for these devices. A single task is provided to collect and send motor data to the host workstation in the same fashion as is done for the force sensor.

6.6.2 CPU_1

CPU_1 is used as a path planner and as a means of uploading robot state and setpoint data to the X-Windows host computer. The tasks found on CPU_1 are shown in Figure 6.11.

The path planner task sends setpoints to the controller on CPU_2 at fixed time intervals. The specific time interval is stored in the path planner data and may be set from the X-Windows interface via the user task. The setpoint data may be generated by any of a number of path planners. The path planner in use is indicated by the path planner data and is also selectable from the X-Windows interface via the user task. Current path planners are sinewave, hand controller, file and hold. State information returned by the controller and setpoint data are time stamped and stored in the upload data shared memory area. The sinewave path planner was constructed as a means to test that low level control of the manipulator had been achieved. The sinewave planner allows the user to specify a separate

Figure 6.11 Tasks and Interprocess Communication on CPU_1



sinewave input on each joint with a specified centre point, amplitude, frequency and phase angle. Each of these parameters is given in the path planner data and is setable from sliders on the X-Windows interface via the user task. The hand controller path planner reads data from the hand controller global memory pool on CPU_0 to generate Cartesian setpoints for the manipulator. The arm's inverse kinematics are used to translate these into joint setpoints for the manipulator. The file path planner allows a preplanned trajectory to be read from the host machine. At each time increment of the path planner the next line of the file is passed to CPU_2 as the setpoint. The hold path planner is the default. It keeps the setpoint at a fixed value.

The switch path planner task is used to control the switch between two different path planners. When the path planner data indicates that a new path planner has been chosen this task temporarily switches the path planner to hold, closes any devices used by the old path planner, opens any devices used by the new path planner and the switches the path planner to the new requested one.

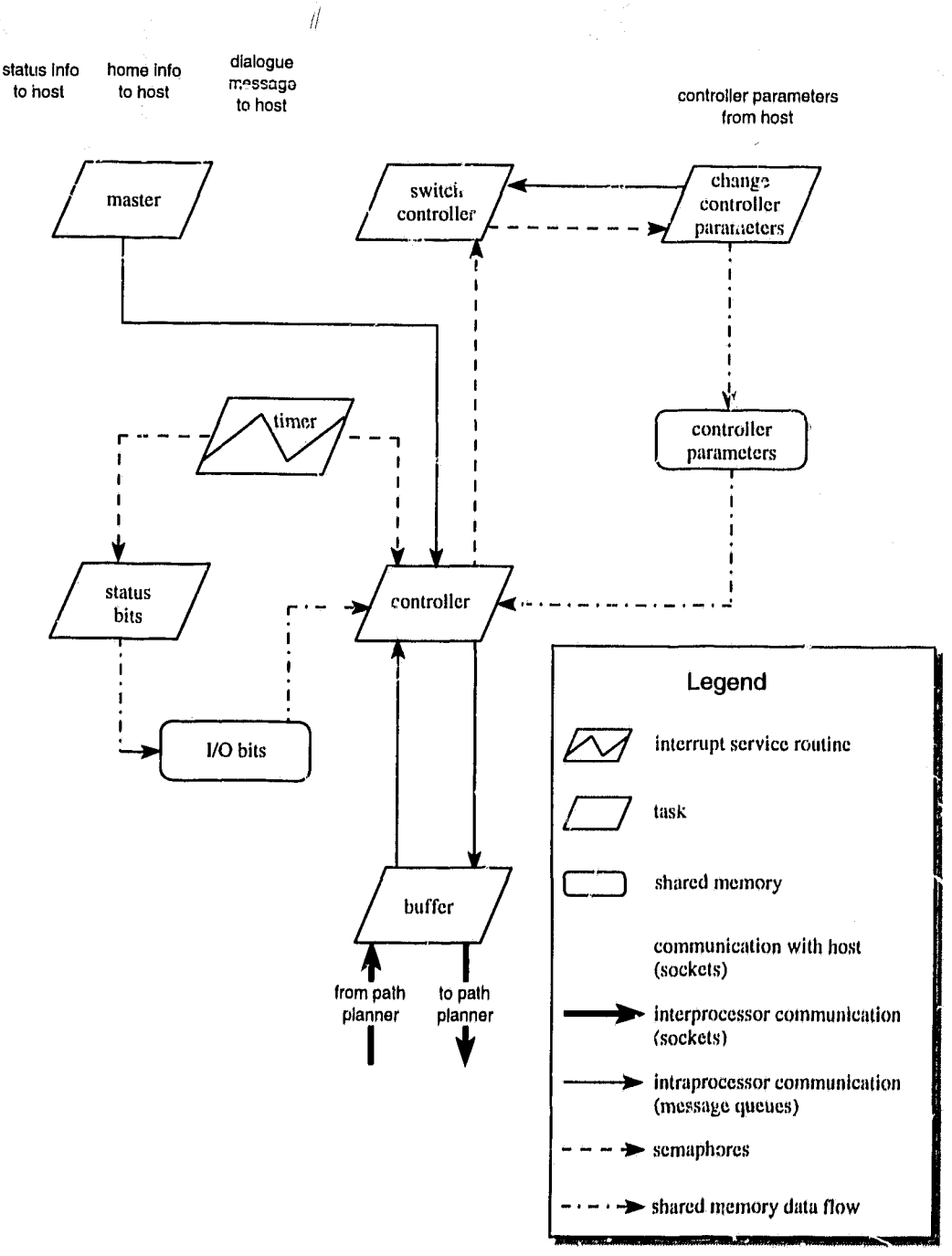
The user task receives data packets from the user interface program running on the host X-Windows workstation. Data received is parsed and placed in the path planner data memory space where it effects path planner operation.

The upload task receives data collection requests from the host workstation. It collects the requested data from the upload data memory space and returns it to the host.

6.6.3 CPU_2

CPU_2 executes the low level control program for the manipulator and monitors the health of the system's hardware. The tasks running on CPU_2 are shown in Figure 6.12.

Figure 6.12 Tasks and Interprocess Communication on CPU_2



The buffer task accepts setpoint data from the path planner task on CPU_1 and passes it on to the controller task. State data received from the controller task is passed back to the path planner task. The buffer task, as the name suggests, acts as a buffer between the path planner and the controller. It is used to ensure that the controller task never waits for setpoint data from the path planner. If new setpoint data is not available, the controller program notes this but continues unblocked. This function is critical since blocking the controller can have severe repercussions on the manipulator performance including damage to the equipment.

The controller task performs the low level control operations for the manipulator. At each controller period it is awakened by the timer interrupt service routine to perform one control cycle. The controller period is selectable by the user through the change controller parameters task. A typical controller period is ten milliseconds. The controller task checks the I/O bits data area to verify the status of the system hardware and note the status of the various binary input bits upon which it must act during this controller cycle. If the I/O bits indicate no problem with the hardware, the controller indicated in the controller parameters data area uses setpoint data passed from the buffer task, and newly read state data to compute new control outputs for each joint. These control outputs are written to the Reis I/O card registers appropriate for each joint. The controller may also write various binary output bits to the hardware to control binary output devices such as the pneumatic gripper. There are two controllers available. A simple PI joint control law on joint position and an impedance controller. The PI control law is described in more detail in Section 4.3.3. The impedance controller is described in Appendix F.

The status bits task is responsible for reading the status of the system hardware and storing its condition in the I/O bits data area so that the information can be used by the

controller task. Like the controller task, it is awakened at set time intervals by the timer interrupt service routine to perform a single cycle of operation.

Similar to the switch path planner task on CPU_1, the switch controller task is used to manage the switch between two different controllers. During a controller switch it temporarily switches the controller to hold the manipulator at its current set of joint angles, closes any devices used by the old controller, opens any devices used by the new controller and then switches the controller to the new requested one.

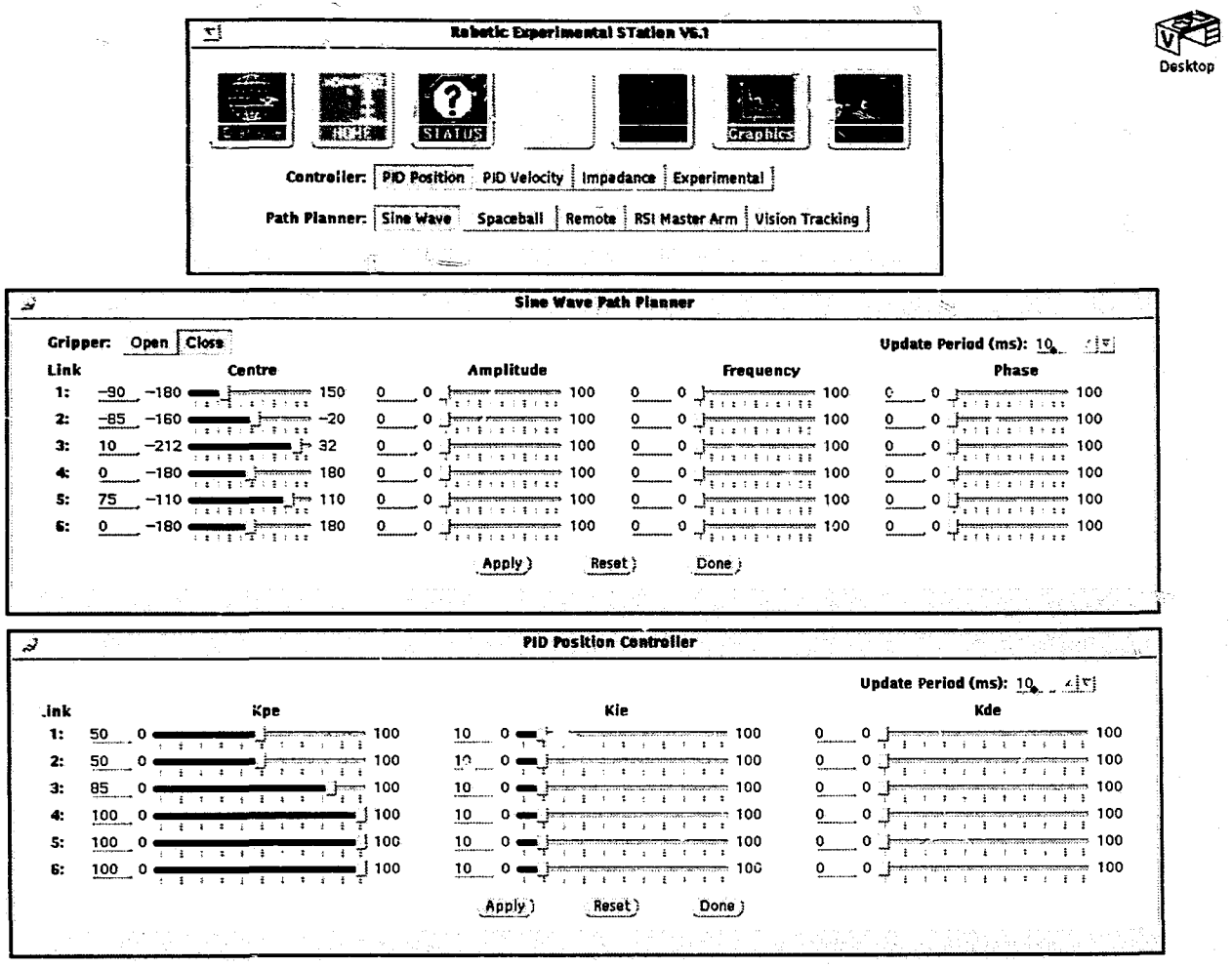
The change controller parameters communicates over a socket connection to the user interface program running on the host X-Windows workstation. Updated controller parameters from the user are received through this connection, parsed and stored in the controller parameters data area where they may effect the operation of the controller on the next cycle of the control task.

The master task is responsible for synchronizing the start up of the system. It initializes all of the hardware, spawns the other tasks on CPU_2 and ensures that all necessary task synchronization is complete. The master task also provides three socket connections to the user interface program running on the host X-Windows workstation through which hardware status information, homing status information and dialogue messages are communicated. Dialogue messages are used to queue the operator to perform certain operations either in the start up of the system or when necessary in the correction of a system error.

6.7 X-Windows User Interface

A graphical user interface program running on a Sun workstation and displayed on any X-Windows terminal allows the system user to monitor and control the operations of

Figure 6.13 X-Windows User Interface



the Reis V15 manipulator. Information is exchanged with the VME bus control computer over ethernet using stream sockets. The connection points are indicated on Figures 6.10 through 6.12 where the tasks running on each of the three VME bus CPUs are shown. The main interface panel consists of seven iconic buttons, a row of "Controller" buttons and a row of "Path Planner" buttons.

The "Dialogue" button opens a communications panel which provides operating instructions to the user. The "Home" button brings up a panel showing each joints home status. The "Status" button opens a panel showing the status of the manipulator hardware. The "F/T Sensor" button provides a display of the force sensor and provides facilities for force sensor data collection. The "Spaceball" button shows the status of the Spaceball input device. The Spaceball is an off the shelf six degree of freedom joystick device. This device was the first external device incorporated into the system. It was on temporary loan from an industrial laboratory. It has since been replaced by an RSI six axis hand controller. The "Graphics" button brings up a panel which provides an interface to state and setpoint data acquisition facilities. The sound button was created as an interface to allow prescribed sounds to be played as a joint neared its joint limit. This was intended as an auditory feedback mechanism to the operator.

The row of controller buttons allows the user to select a controller. Controllers may be changed on the fly. The panel brought up by the "PID Position" controller button is shown at the bottom of Figure 6.13. Here the user may adjust the proportional, integral and derivative gains for each of the manipulator's joints. The controller period, whose default is ten milliseconds, can be adjusted by changing the number in the upper right corner of the panel.

A selection of path planners are also available through the row of "Path Planner" buttons. Again the path planners may be changed on the fly. The panel brought up by the "Sinewave" path planner button is also shown in Figure 6.13. The parameters defining the sinewave path applied to each joint are adjusted via this panel. The path planner period is also adjusted in the upper right corner of the panel.

6.8 Trajectory Optimization

6.8.1 A Test Problem

The test problem is to find a trajectory which moves the Reis V15 manipulator from a position of $(0.7853, -0.7853, -0.524)$ radians to $(-2.356, -0.7853, -0.524)$ radians with the minimum energy consumption. The joint velocities at the trajectory endpoints are all zero. In its initial position, the manipulator is extended out towards the periphery of its workspace with tip of the manipulator approximately one meter off of the floor. The trajectory can be completed by simply turning the first joint through π radians. Further, the trajectory is to be completed in twenty seconds. This type of time constrained motion commonly arises when a motion must be completed to coordinate with an external event such as the arrival of a part in the manipulator workspace. The starting trajectory is depicted in phase space in Figure 6.14. The starting point for each joint's trajectory is as indicated in the figure.

6.8.2 Optimization Using Time History of Joint Angles

Optimization using a C2 cubic spline representation of the time history of the manipulator's joint angles proceeds using the Modified Dynamic Programming algorithm described in Section 5.2.6. The optimization begins with three stages. The stage contain-

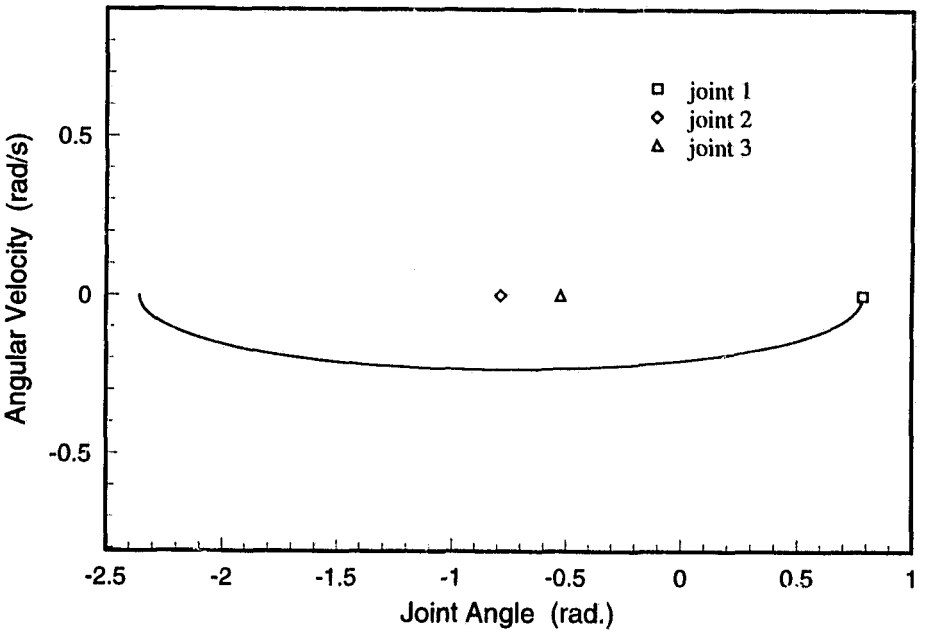


Figure 6.14 Test Problem - Initial Trajectory

ing the initial point, the stage containing the terminal point and a single intermediate stage. Usually the terminal stage in tessellated along a single dimension corresponding to variations in the time scaling factor or overall trajectory time. However, since the trajectory is to be completed in a fixed and specified time, the terminal stage contains only a single point with a value equal to the prescribed trajectory time of twenty seconds. The initial spline interpolation points are shown in Table 10. The intermediate stage is tessellated into a 125 point grid generated by five points for each joint at an initial spacing of 0.2 radians. The model estimation of the energy consumption for the initial trajectory is 3302 Joules. After eight passes of the algorithm or 2007 function evaluations the trajectory is as shown in Figure 6.15. The trajectory for the first joint is virtually unchanged. The second joint is lifted up before being returned to its terminal position. The primary effect of this motion is to reduce the energy which the second joint's motor must expend to support the weight of the link against gravity during a portion of the trajectory. The third joint moves very little.

Table 10 Initial Spline Interpolation Points - Three Stage Grid

scaled time	Joint Angle 1	2	3	time scaling factor
0.0	0.7853	-0.7853	-0.524	-
0.5	-0.7853	-0.7853	-0.524	-
1.0	-2.356	-0.7853	-0.524	20

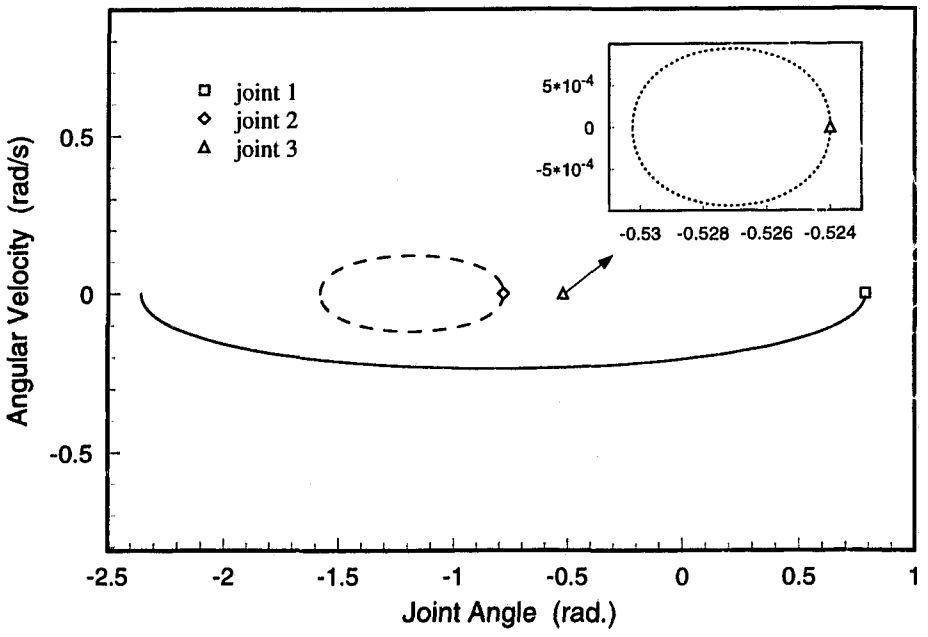


Figure 6.15 Test Problem - Optimized Trajectory with Three Stages

Its motion is shown magnified in the inset of the figure. The model estimated energy consumption for this trajectory is 1734 Joules.

Two intermediate stages were added such that their zeroth points left the best trajectory achieved by the three stage optimization unchanged. The initial interpolation points for the five stage optimization are shown in Table 11. The intermediate stages were tessellated into 27 point grids generated by three points along each joint dimension at a

spacing of 0.2 radians. After four passes of the algorithm or an additional 6051 function

Table 11 Initial Spline Interpolation Points - Five Stage Grid

scaled time	Joint Angle 1	2	3	time scaling factor
0.0	0.7853	-0.7853	-0.524	-
0.25	0.304563	-1.183737	-0.527125	-
0.5	-0.764987	-1.582175	-0.530250	-
0.75	-1.854990	-1.183738	0.527125	-
1	-2.356	-0.7853	-0.524	20

evaluations the trajectory is as shown in Figure 6.16. The motion of the first joint is slowed in the middle portion of the trajectory to allow the second link to be an elevated position for a greater portion of the trajectory duration. The second joint is lifted more quickly. The loop in the phase space trajectory is an attempt by the algorithm to create a point in the trajectory where the second joint is held fixed in a vertical position (approximately $-\pi/2$ radians). As before, there is little motion of the third joint. This is shown in the inset of the figure. The model estimated energy consumption for this trajectory is 1289 Joules. Despite the fact that there is a significant reduction in the estimated energy consumption by using five stages in the optimization algorithm, the fact that the algorithm generated a trajectory with a loop in the phase plane is of some concern. It is unlikely that a truly energetically optimal trajectory would exhibit this behavior. What the algorithm is attempting to achieve is a trajectory where the second joint is lifted quickly to its vertical position and then held there at zero velocity before being returned to its prescribed terminal position. The splined time history of joint angles representation is capable of generating this behavior but may require more stages to do so. In addition, the distribution of stages in the representation is as important a factor as the number of stages. In the algorithm used here, the stages are assumed distributed evenly along the time axis. This is likely not the best approach.

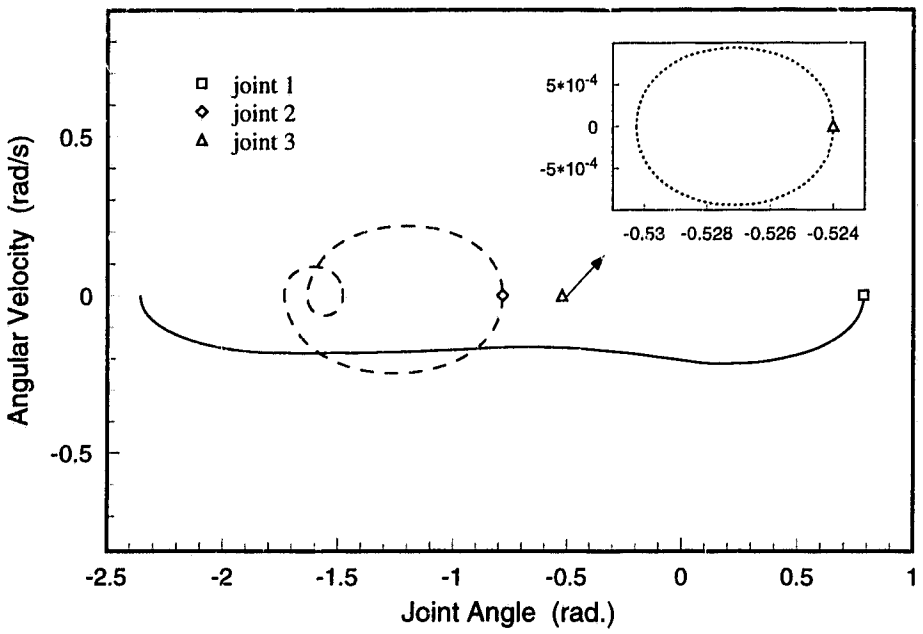


Figure 6.16 Test Problem - Optimized Trajectory with Five Stages

Unfortunately it is not clear before hand what the best stage distribution should be. A heuristic could be developed to strategically shift or add stages. Another approach is to change the representation completely.

6.8.3 Optimization Using a Phase Space Representation

Recall that in Chapter 5, optimization of a trajectory for a simulated three link arm was achieved with fewer function evaluations and better results when the Modified Dynamic Programming algorithm was applied in phase space as outlined in Section 5.2.7. For the problem considered here, there are a couple difficulties which arise when computing the energy consumption for a given trajectory. First since the Reis V15 model contains a model of a discrete PI controller which expects joint inputs at fixed 0.01 second intervals, it is not possible to integrate the system equations directly in terms of the B spline

representation's parametric variable s. Second, the trajectory time has been specified to be twenty seconds. In the phase space representation, time information is contained implicitly in the trajectory. Both problems can be addressed by a time scaling approach.

Perform the numerical integration

$$t_i = \int \frac{\theta'_i(s)}{\dot{\theta}_i(s)} ds \tag{6.5}$$

for each joint i. At intervals along the integration save the time, position and velocity information. After the trajectory time for each joint has been determined, create a time scaling factor for each joint by dividing the prescribed trajectory time of twenty seconds by the calculated trajectory time of the corresponding joint t_i. The time data collected during the integration are adjusted by multiplying by this time scaling factor. The corresponding velocity data must be adjusted by dividing by this time scaling factor. A set of position and velocity setpoints which act as input to the Reis V15 model are generated by linearly interpolating in this time scaled data.

With this modification to the method of computing the energy consumption for the Reis trajectory, the Modified Dynamic Programming algorithm was applied in phase space to the current problem. Since the phase space approach has difficulties, described in Section 5.1.3, with the selection of an initial trajectory when the initial and terminal states of one or more of the joints are the same, the result obtained by the three stage optimization using the time history of the joint angles representation is used to create an initial trajectory. The B spline control vertices for the initial trajectory are shown in Table 12. Three points were used at each intermediate stage for each joint. The grid spacing was 0.1. After two passes of the algorithm or 137 function evaluations the trajectory is as shown in Figure 6.17. The B spline control vertices for this trajectory are given in Table 13. The tra-

Table 12 Initial B Spline Control Vertices - Phase Space Representation

Joint	Position (rad)	Velocity (rad/s)	Description
1	0.7853	0.0	initial point
	0	-0.2	
	-0.7853	-0.2	
	-1.5708	-0.2	
	-2.356	0.0	terminal point
2	-0.7853	0.0	initial point
	-0.9817	-0.1	
	-1.178	-0.1	
	-1.3744	-0.1	
	-1.5708	0.0	crossing point
	-1.3744	0.1	
	-1.178	0.1	
	-0.9817	0.1	
	-0.7853	0.0	terminal point
3	-0.524	0.0	initial point
	-0.527	-0.001	
	-0.530	0.0	crossing point
	-0.527	0.001	
	-0.524	0.0	terminal point

jectory for the first joint is similar to that in Figure 6.16. Again the third joint moves very little. Details of its motion are shown in an inset of the figure. The largest difference is in the trajectory of the second joint. With the phase space representation the second link is moved to a vertical posture more quickly and is held near zero velocity without creating a loop in the phase space trajectory. An inset in the figure magnifies the crossing point in this near zero velocity region. As the inset shows the trajectory does indeed cross the axis at a right angle as is physically mandated. The model estimated energy consumption for this trajectory is 844 Joules.

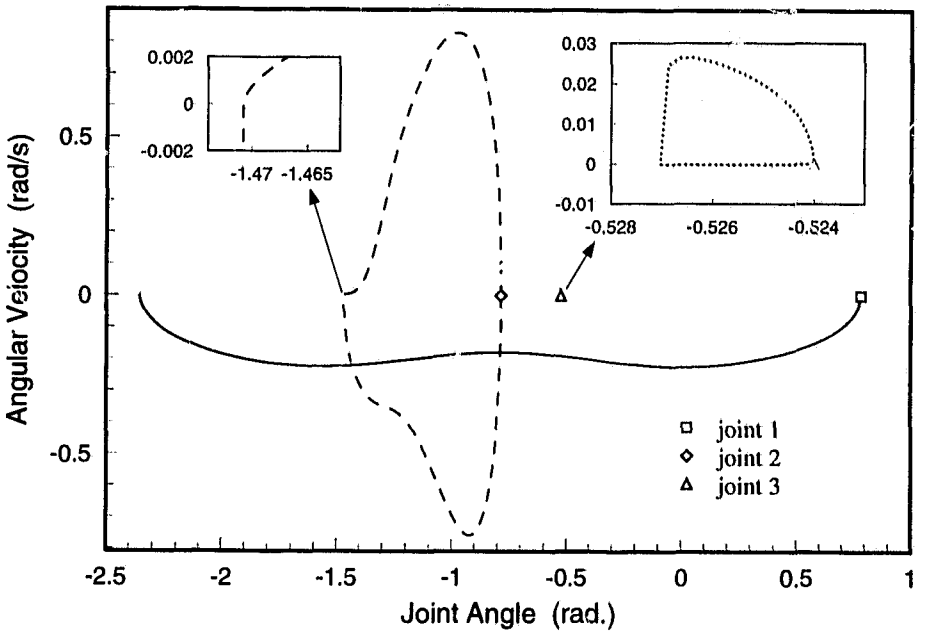


Figure 6.17 Test Problem - Optimized Trajectory Using Phase Space Representation

Each of the trajectories in Figure 6.15, 6.16 and 6.17 were executed on the Reis V15 manipulator. Motor current and motor velocity data were collected during each trajectory. From the bond graph model of the Reis V15 actuators in Figure 4.18 on page 87, the power consumption for a given trajectory is the sum over each joint of the power dissipated in resistive elements representing the armature and amplifier resistances,

$$P_R = i_a^2 (R_a + R_{Tr}), \tag{6.6}$$

and that delivered to the manipulator mechanism by the motors,

$$P_M = \tau_m \omega_m = K_m i_a \omega_m. \tag{6.7}$$

In the event that the motor torque and angular velocity have the same sign the power flow is positive and the motor is delivering energy to the mechanism. If the motor torque and

Table 13 Optimized B Spline Control Vertices - Phase Space Representation

Joint	Position (rad)	Velocity (rad/s)	Description
1	0.7853	0.0	initial point
	0	-0.2	
	-0.7853	-0.1	
	-1.5708	-0.2	
	-2.356	0.0	terminal point
2	-0.7853	0.0	initial point
	-0.9116	-0.2860	
	-1.1780	-0.1	
	-1.3427	-0.1143	
	-1.4708	0.0	crossing point
	-1.3497	0.0031	
	-1.2223	0.1897	
	-0.9323	0.2938	
	-0.7853	0.0	terminal point
3	-0.524	0.0	initial point
	-0.527	-0.001	
	-0.52703	0.0	crossing point
	-0.527	0.101	
	-0.524	0.0	terminal point

angular velocity have opposite signs then energy is being returned from the mechanism to the actuator. Since the Reis V15 has no way to capture and reuse this energy, it is lost. In the model it is, for the most part, sunk into the flow source. Therefore when computing the energy consumption for a given trajectory, power delivered to the mechanism is considered as a positive consumption, power returned from the mechanism is co sidered zero consumption rather than negative consumption. The power consumption for a particular actuator is therefore given by,

$$P = (1 + \text{sgn}(K_m i_a \omega_m)) K_m i_a \omega_m + i_a^2 (R_a + R_{Tr}) . \quad (6.8)$$

The values of the constants K_m, R_a and R_{Tr}, are provided in Table 3 on page 82. Equation 6.8 is integrated with respect to time with an Euler integrator using the motor current and velocity data collected during the execution of the planned trajectories. The resulting energy consumption estimates for each of the planned trajectories are shown along with a summary of the model estimates in Table 14.

Table 14 Energy Consumption of Planned Trajectories

trajectory	model energy estimate (J)	% reduction	total function evaluations	robot energy (J)	% reduction
initial	3302	0	0	4451	0
3 stage	1734	48	2007	3056	31
5 stage	1289	61	8058	1987	55
phase representation	844	74	137	1022	77

The Modified Dynamic Programming algorithm is successful in planning trajectories with significant reductions in energy consumption for the test move. Due to limitations in the model, particularly mass and inertia estimates, the values of the energy consumption estimates provided by the model are as much as 43% off the experimentally determined energy consumption. As supported by the data in Table 14, the model does however capture the sufficient characteristics of the actual system to enable optimal trajectory planning. As with the simulated example in Chapter 5, the phase space representation again provides a superior representation. A 77% reduction in the energy consumption over the initial trajectory was achieved with only 137 function evaluations. One qualifier to this is that for this trajectory, it was necessary to use information gained from a previous optimization using the time history of the joint angles to generate a suitable set of starting control vertices.

Chapter 7
Conclusions

The goal of this research has been to develop a practical approach to the planning of minimum energy consumption trajectories for robotic manipulators. That is, a trajectory between given initial and final states such that the energy consumption is minimized is sought. As stated, this is a functional optimization problem. If the manipulator equations of motion are unencumbered by ill-behaved nonlinearites such as friction and saturation in the actuators, or a discrete time control law, the minimum energy trajectory planning problem is easily formulated using the variational or so called classical approach. The difficulty is that the two point boundary value problem which arises from such a formulation is ill-conditioned and for practical purposes, insoluble by current approaches. In real world cases, where the equations describing a manipulator's motion and energy consumption are complex enough that they are generally described algorithmically, even the formulation of the problem in the classical sense is at best difficult. By searching for an optimal trajectory in a restricted class of functions, namely those which can be described by a spline parameterized by a finite set of control vertices, the problem can be converted

from a functional optimization problem to a function optimization problem for which there are a large number of numerical approaches which can be attempted.

Numerical function optimization problems can be divided into those which find a local minima and those which attempt the more difficult problem of finding a globally optimal solution. Of the local optimization procedures, the direction set methods are the most common. These algorithms start at some point in the n-dimensional space and then perform a series of line minimizations along some vector directions using a one-dimensional line minimization routine. The algorithms differ in the method the use after each line minimization to select the next direction to try. Many of these algorithms require the ability to compute the function's gradient. For practical trajectory planning problems the gradient information is not generally available analytically. For this reason Powell's method is attempted since it does not require gradient evaluations. Powell's method was successful on a simple single link minimum time optimization problem, achieving an optimal or near optimal result with 407, 277, or 1707 function evaluations depending on whether a C1 cubic, C2 cubic or quartic spline representation of the time histories of the joint angle was used to describe the trajectory. When applied to a more difficult three link minimum energy trajectory planning problem, Powell's method was able to find a trajectory with approximately a 25% reduction in energy consumption over the initial trajectory with 1102, 877 or 1162 function evaluations depending on whether a C1 cubic, C2 cubic or quartic spline representation was used.

Four types of global optimization procedures were attempted. These are an interval methods such as the Moore-Skelboe and Hansen's algorithms, genetic algorithms, simulated annealing and dynamic programming. Of these, the only methods guaranteed to find a globally optimal solution, if they converge, are the interval methods. The interval methods depend on the ability to describe the valid set of spline control vertices as an n-dimen-

sional hyperbox. This hyperbox must be able to be mapped, using interval mathematics, to a corresponding interval for the energy consumption of all possible trajectories which can be described by points in the hyperbox. Further, as the size of the hyperbox is decreased the corresponding calculated interval must decrease. In the literature, the theory on how to construct these mapping exists for polynomial and some rational functions. For the extremely complex functions which result from minimum energy path planning problems, no suitable mapping was found.

Genetic algorithms and simulated annealing both take their form from natural processes. Genetic algorithms are modelled on the process of natural selection. Simulated annealing is based on the thermodynamic process by which materials achieve lower energy states when cooled. Both of these types of algorithms were found to require far too many function evaluations. Typically more than 100 times as many function evaluations were required to achieve a worse solution than was obtained using Powell's method. These methods do have the ability to achieve a lower energy trajectory than Powell's method, but the computational price for this improvement is excessive. These methods are more suited to problems which do not have parameters such as end states which change, requiring the solution to be repeated.

Dynamic programming was also considered as a valid approach to obtaining a global optimum. Dynamic programming can obtain a global optimum on a discrete grid. The difficulty is that to obtain a good optimum requires a tight grid spacing. This is not a large concern for a single link arm but dynamic programming does not scale well to manipulators with multiple links. The number of function evaluations required by the algorithm is given by,

$$\text{total function evaluations} = Q^P R^P \{ (N-3)\, Q^P R^P + M + 1 \} \tag{7.1}$$

where, N is the number of stages in the grid, Q is the dimension of joint angle grid points, R is the dimension of joint velocity grid points, M is the number of time scaling factor grid points and P is the number of joints. With a computer with a parallel architecture is available, up to $Q^{2P}R^{2P}$ function evaluations between grid stages may done simultaneously. Even with the possibility of parallel computation, for a three link arm $(P = 3)$, even modest values for M, Q and R result in an unrealistic number of function evaluations. To reduce the number of function evaluations required, a number of changes were made to the basic dynamic programming algorithm.

First, the dimension of the problem was reduced by changing the class of functions over which the optimum is sought. Instead of considering the trajectory to be made up of a sequence of subtrajectories between points on adjacent stages, the trajectory was constructed form the zeroth grid points at all stages before the current stage, the point of interest on the current stage, and the linked list of points generated by the algorithm as it moves backwards from the terminal stage. This representation, requires only the initial and terminal velocities to completely describe the trajectory. Therefore tessellation along a dimension corresponding to the joint velocities at the intermediate stages of the grid is eliminated. The number of function evaluations required by this modified algorithm is given by,

$$\text{total function evaluations} = Q^P \{ (N-3) (Q^P - 1) + M \} \tag{7.2}$$

Here $Q^P (Q^P - 1)$ functions may be evaluated in parallel between intermediate stages. While this is a reduction of approximately R^{2P} times in the number of function evaluations required over the dynamic programming approach, it is still at least three orders of magnitude more than that required for a solution using Powell's algorithm. The fact that parallelization may be used to speed the solution process up is of little value since comput-

ers do not exist which can take advantage of the degree of parallelization which arises for reasonable values of Q and P.

To reduce the computational burden further, an algorithm was developed which applies the modified dynamic programming method over a sequence of grids. Instead of tessellating the entire solution space, a coarse grid is constructed over a portion of it. The optimum over this coarse grid is found. Based on the optimum trajectory found, the grid is either shifted or the grid spacing tightened and the algorithm repeated. This method does give up the guarantee of finding a global optimum and as such should be considered a local method. It is important to note however that this method is not local is the sense of direction set methods. Rather than stepping sequentially in a series of directions, it searches over local regions. As such, the method has the ability to avoid some local minima in favor of deeper minima. The degree to which this ability is exhibited depends upon the size and spacing of the grid chosen by the algorithm's user. This method and Powell's method were applied to the same three link minimum energy trajectory planning problem. Each algorithm was run 100 times, each time from a randomly generated starting point in the solution space. Even with a small grid of three points per stage and a relatively coarse spacing of 0.2 radians at the intermediate stages, the modified dynamic programming algorithm was able to avoid poorer local minima more often. The modified dynamic programming algorithm did require approximately fifteen times as many function evaluations, but the fact that a number of the function evaluations may be done in parallel would allow a modest parallel architecture computer to surpass the speed achievable by Powell's algorithm. Further improvements were observed if the two methods were combined. That is, first the modified dynamic programming algorithm was used, and then Powell's method was started from the trajectory found by the modified dynamic programming method.

It was noted that describing the trajectory by splining the time history of the joint angles often resulted in optimized trajectories which had loops when plotted in phase space. Since it seemed unlikely that it would be truly optimal for a trajectory to exhibit this characteristic, a third modification was made to the basic dynamic programming algorithm. This modification again involves a change in the class of functions over which the algorithm searches. Instead of splining the time history of the joint angles, each joint's phase space trajectory was represented by a spline. This representation provided a couple of programming difficulties. First it is possible that a joint's trajectory in phase space will wrap back on itself. Further when the trajectory crosses the $\dot{\theta} = 0$ axis, it must do so perpendicular to the axis. These representational problems were solved by borrowing techniques from the computer graphics field. Parametric uniform cubic B-splines were used to represent the joint trajectories. Phantom vertices were calculated to ensure perpendicular crossing points. A second difficulty of this approach is that time is represented implicitly in the trajectories. Methods were developed to extract both the time and joint acceleration information necessary for computing the trajectory energy consumption. Another artifact of the implicit representation of time in the phase space representation is that the stages in the tessellated grid for one joint have no correlation to the stages in the grids for the other joints. As a result stages do not represent some scaled time at which one tessellates in one dimension for each joint as is done for the time domain representation. Rather, each joint is tessellated at each of its own stages independently. This reduces the number of function evaluations for one pass of the algorithm to,

$$\text{number of function evaluations} = \sum_{i=1}^{P} Q\,(N_i - 3)\;(Q-1)\,. \tag{7.3}$$

Therefore the algorithm proceeds by perturbing the trajectory of each joint in turn rather that allowing the perturbation of all joint trajectories simultaneously. $Q\,(Q-1)$ function

evaluations may be done in parallel between intermediate stages. A technique for tessellating the grid at each stage of a joints trajectory was developed which keeps the dimension of the tessellation at one but still provides for the radial expansion or contraction of the phase space trajectories that was observed in the progression of the optimization algorithms in the time domain. When run on the example three link minimum energy trajectory planning problem this phase space modified dynamic programming algorithm yielded a thirteen percent improvement over both Powell's method and modified dynamic programming in the time domain. Furthermore, this improvement was realized with fewer function evaluations than either of the other methods. It should be noted that this algorithm can be difficult to start, particularly when one or more of the joint's trajectories begin and end at the same state. In these cases some insight is needed into the shape that the optimized trajectory is likely to take so that a starting trajectory may be generated. Barring this, the algorithm may be started in time domain and then switched to phase space once a trajectory pattern has developed.

The modified dynamic programming algorithms were verified experimentally using a Reis V15 industrial manipulator. To perform the experimental verification, the manipulator's proprietary control system was replaced by one of my own design to allow the necessary access to motor data and the low level control signals. While the model of the Reis V15 used in the optimization algorithms was of limited accuracy due to rough mass and inertia estimates, the model was able to capture sufficient characteristics of the real system to enable optimal trajectory planning. Using the modified dynamic programming algorithm in the domain provided a 55% reduction in energy consumption for a test time constrained pick and place trajectory. Performing the same optimization using modified dynamic programming in phase space yielded a 77% reduction in energy consump-

tion and took less than 2% of the function evaluations required by the time domain approach.

The modified dynamic programming approach developed in this work can be a powerful tool, particularly in its phase space form, for computing energetically optimal trajectories for robotic manipulators. It appears to be an efficient algorithm. Significantly improved trajectories can be arrived at in only a few hundred function evaluations. The compute time can be reduced if a parallel computer architecture is available to exploit the parallelisms in the algorithm. The method, while technically a local optimization procedure, can avoid some poorer local minima in favor of deeper minima. The degree to which the algorithm is able to avoid poor local minima is controlled by the grid size and density parameters in the algorithm. Finally constraints are easily incorporated into the algorithm. In this work, actuator constraints in the form of motor current limits were placed in the system model. A trajectory optimization with a time constraint was performed on a real manipulator. Further constraints which are likely to appear in the remote manipulator applications for which this work was intended are workspace obstacle constraints and reaction force constraints. These constraints are easily incorporated into the modified dynamic programming algorithms. It remains the topic of further study to see how effective these algorithms are in problems incorporating these constraints.

Chapter 8
References

1. Alefeld, Götz and Jürgen Herzberger, *Introduction to Interval Computations*, Academic Press, 1983.

2. Allen, R. R., Dynamics of Mechanisms and Machine Systems in Accelerating Reference Frames, Trans. *ASME Journal of Dynamic Systems Measurement and Control*, vol. 103, December 1981, pp. 395-403.

3. Analog Devices Inc., *User's Manual RTI-600 RTI-602*, Analog Devices Inc., Norwood Massachusetts, 1984.

4. Beer, Ferdinand P. and E. Russel Johnston Jr., *Vector Mechanics for Engineers Statics and Dynamics*, 3rd Edition, McGraw Hill Inc., 1977.

5. Bellman, Richard, *Dynamic Programming*, Princeton U iversity Press, 1957.

6. Bellman, Richard, *Dynamic Programming and Modern Control Theory*, Academic Press Inc., New York, 1965.

7. Bessonnet, G. and J. P. Lallemand, Optimal Trajectories of Robot Arms Minimizing Constrained Actuators and Travelling Time, *in Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, May, 1990, pp. 112-117.

8. Bos, A. M. and M. J. L. Tiernego, Modelling the Dynamics and Kinematics of Mechanical Systems with Multibond Graphs, *Journal of the Franklin Institute*, vol. 319, no. 1/2, 1985, pp. 37-50.

9. Bos, A. M. and M. J. L. Tiernego, Formula Manipulation in the Bond Graph Modelling and Simulation of Large Mechanical Systems, *Journal of the Franklin Institute*, vol. 319, no. 1/2, 1985, pp. 51-65.

10. Bobrow, J. E., *Optimal Control of Robotic Manipulators*, Doctoral Dissertation, UCLA, 1982.

11. Bobrow, J. E., S. Dubowsky and J. S. Gibson, On the Optimal Control of Robotic Manipulators with Actuator Constraints, in *Proceedings of the American Control Conference*, San Francisco, June 1983, pp. 782-787.

12. Bobrow, J. E., S. Dubowsky and J. S. Gibson, Time-Optimal Control of Robotic Manipulators Along Specified Paths, *International Journal of Robotics Research* 4(3), Fall 1985, pp. 3-17.

13. Bobrow, J. E., Optimal Robot Path Planning Using the Minimum Time Criterion, *IEEE Transactions on Robotics and Automation*, vol.4, no. 4, August, 1988, pp. 443-450.

14. Braibant, V. and M. Geradin, Optimum Path Planning of Robot Arms, *Robotica* 5, 1987, pp. 323-331.

15. Breedveld, P. C., Multibond Graph Elements in Physical Systems Theory, *Journal of the Franklin Institute*, vol 319, no. 1/2, 1985, pp. 1-36.

16. Chen, Y. and A. A. Desrochers, Time-Optimal Control of Two-Degree of Freedom Robot Arms., in *Proceedings of the IEEE Conference on Robotics and Automation*, Philadelphia, PA., 1988, pp. 1210-1215.

17. Chou, Li-Shan and Shin-Min Song, Geometric Work of Manipulatros and Path Planning Based on Minimum Energy Consumption, in *ASME Proceedings of the 21st Biennial Mechanisms Conference* DE-Vol. 24, Chicago Ill., Sept. 1990, pp. 319-326.

18. Craig, John J., *Introduction to Robotics Mechanics and Control*, Addison-Wesley Publishing Company, Reading Ma., 1986.

19. Davis, Lawrence, Ed., *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London, 1987.

20. Dubowsky, S. and Z. Shiller, Optimal Dynamic Trajectories for Robotic Manipulators, in *Proceedings of the Fifth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, 1984, pp. 133-143.

21. Field, G. A., *Optimal Trajectory Planning for Robotic Manipulators with Minimum Energy Cost Criteria*, M.A.Sc Thesis, University of Waterloo, 1988.

22. Field, G. A., Energetically Optimal Trajectory Planning for Robotic Manipualtors, in *Proceedings of the 20th Annual Pittsburgh Conference on Modelling and Simulation*, vol. 20, May 1989, Pittsburgh, PA., pp. 1927-1932.

23. Fletcher, R. and M. J. D. Powell, A Rapidly Convergent Method for Minimization, *Computing Journal*, vol. 6, no. 2, 1963, pp. 163-168.

24. Fourquet, Jean-Yves, Optimal Control Theory ans Complexity of the Time Optimal Problem for Rigid Manipulators, in *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Yokohama, Japan, July, 1993, pp. 84-90.

25. Geering, H. P., L. Guzzella, S. A. R. Hepner and C. H. Onder, Time-Optimal Motions of Robots in Assembly Tasks, *IEEE Transactions on Automatic Control* AC-31(6), June, 1986, pp. 512-518.

26. Gonzales-Baños, Hector Hugo and José Luis Gordillo, An Algorithm for Planning Time-Optimal Trajectories for Given Minimum Distance Paths, in *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Yokohama, Japan, July, 1993, pp. 2302-2309.

27. Hansen, E. R., Global Optimization Using Interval Analysis - The One Dimensional Case, *J. Optim. Theor. and Appl.*, vol. 29, no. 3, Nov. 1979, pp. 331-344.

28. Hansen, E. R., Global Optimization Using Interval Analysis - The Multidimensional Case, *Numerische Mathematik*, vol. 34, 1980, pp. 247-270.

29. Hogan, N., Impedance control: An approach to manipulation: Part I - Theory, *ASME Journal of Dynamic Systems, Measurement and Control*, 1985, vol. 107, pp. 1-7.

30. Hogan, N., Impedance control: An approach to manipulation: Part II - Implementation, *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 107, 1985, pp. 8-16.

31. Hogan, N., Impedance control: An approach to manipulation: Part III - Applications, *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 107, 1985, pp. 17-24.

32. IRT Ltd., *IRT Series 600 Description Adjustments Setup.*

33. Jacobsen, D. H., Computation of Optimal Singular Controls, *IEEE Transactions on Automatic Control* AC-15, 1970, pp. 67-73.

34. Kahn, M. E., *The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains*, Doctoral Dissertation, Stanford University, Dec. 1969.

35. Kahn, M. E., and B. Roth, The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains, *ASME Journal of Dynamic Systems, Measurement and Control*, vol 93, no. 3, Sept. 1971, pp. 164-172.

36. Karnopp, D. and R. Rosenberg, *System Dynamics; A Unified Approach*, Wiley, 1975

37. Karnopp, D. and R. Rosenberg, *Analysis and Simulation of Multiport Systems-The Bond Graph Approach to Physical System Dynamics,* MIT Press, Cambridge, Ma., 1968.

38. Kim, B. K. and K. G. Shin, Suboptimal Control of Industrial Manipulators with Weighted Minimum Time-Fuel Criterion, *IEEE Transactions on Automatic Control*, AC-30(1), January 1985, pp. 1-10.

39. Kim, B. K., and K. G. Shin, Minimum-Time Path Planning for Robot Arms and their Dynamics, *IEEE Transactions on Sytems Man and Cybernetics* SMC-15(2), March 1985, pp. 213-223.

40. Lasky, T. A., and Hsia, T. C. "On Force-Tracking Impedance Control of Robot Manipulators", *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento California, April 1991, pp. 274-280.

41. Lee, C. S. G. and B. H. Lee, Planning of Straight Line Manipulator Trajectory in Cartesian Space with Torque Constraints, in *Proceedings of the 23rd IEEE Conference on Decision and Control*, New York, Dec. 1984, pp. 1603-1609.

42. Lewis, F. L., C. T. Abdallah and D. M. Dawson, *Control of Robot Manipulators*, MacMillan Publishing Company, New York, 1993.

43. Lin, C. S., and P. R. Chang, Approximate Optimum Paths of Robot Manipulators Under Realistic Physical Constraints, in *Proceedings of the IEEE International Conference on Robotics and Automation*, New York, 1985, pp. 737-742.

44. Lin C. S., P. R. Chang and J. Y. S. Luh, Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots, *IEEE Transactions on Automatic Control* AC-28(12), Dec. 1983, pp. 1066-1074.

45. Liu, G. J., and Goldenberg, A. A. "Robust Hybrid Impedance Control of Robot Manipulators.", *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento California, April 1991, pp. 287-292.

46. Lord Industrial Automation, Installation and Operations Manual for F/T Series Force/Torque Sensing System, Rev 3, Lord Corporation, Cary, North Carolina, 1987.

47. Lu, W.-S. and Meng, Q.-H. "Impedance Control with Adaptation for Robotic Manipulations", *IEEE Transactions on Robotics and Automation*, vol. 7, no.3, June 1991.

48. Luh, J. Y. S. and M. W. Walker, Minimum-Time Along the Path for a Mechanical arm, in *Proceedings of the 16th IEEE Conference on Decision and Control*, Dec. 1977, New York, pp. 755-759.

49. Luh, J. Y. S. and C. S. Lin, Optimal Path Planning for Mechanical Manipulators, *ASME Journal of Dynamic Systems, Measurement and Control*, vol 102, no. 6, June 1981, pp. 142-151.

50. Luo, Zhi-wei and Ito, Masami, "Control Design of Robot for Compliant Manipulation on Dynamic Environments.", *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento California, April 1991, pp. 42-47.

51. Maier, E. B. and A. E. Bryson, An Efficient Algorithm for the Time-Optimal Control of a Two-Link Manipulator, in *Proceedings of the AIAA Conference on Guidance and Control*, Monteray, CA., 1987, pp. 204-212.

52. Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines, *J. Chem. Phys.*, vol. 21, no. 6, June 1953, pp. 1087-1092.

53. Moore, R. E., *Interval Analysis*, Prentice Hall, Englewood Cliffs, NJ., 1966.

54. Payandeh, S., and Goldenberg, A. A., A Robust Force Controller: Theory and Experiments., *Proceedings of the IEEE Conference on Robotics and Automation,* Sacramento California, April 1991, pp. 36-41.

55. Paynter, H. M., *Analysis and Design of engineering Systems*, MIT Press, Cambridge Mass., 1961.

56. Pfeiffer, F. and R. Johanni, A Concept for Manipulator Trajectory Planning, *IEEE Journal of Robotics and Automation* RA-3(3), April, 1987, pp. 115-123.

57. Pittens, Kenneth, Implementation of a Torque Control System on the Reis RR-V15 Industrial Robot, *Mech 590 : Directed Studies - Project Report*, University of Victoria, December 1991.

58. Podhorodeski, R. P. and W. L. Cleghorn, Optimization of Manipulator Point to Point Motion Considering Actuator Output Capability Constraints, *Mechanism and Machine Theory*, vol. 23, no. 5, 1988, pp 409-419.

59. Pontryagin, L. S., V. G. Boltyanski, R. V. Gamkrelidze and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*, John Wiley and Sons Inc., John Wiley and Sons Inc., New York, 1962.

60. Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, 1988.

61. Raibert, M. H., and Craig, J. J., Hybrid Position/Force Control of Manipulators., *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 102, June 1981, pp. 126-133.

62. Rajan, V. T., Minimum Time Trajectory Planning, in *Proceedings of the IEEE International Conference on Robotics and Automation*, New York, 1985, pp. 759-764.

63. Ratschek, H. and J. Rokne, *Computer Methods for the Range of Functions*, Ellis Horwood Limited, 1984.

64. Ratschek, H. and J. Rokne, *New Computer Methods for Global Optimization*, Ellis Horwood Limited, 1988.

65. Regan, Tim, A DMOS 3A 55V H-Bridge: The LMD 18200, *National Semiconductor: Linear Application Handbook, Application Note 694*, 1991, pp. 1107-1113.

66. Sahar, G. and J. M. Hollerbach, Planning of Minimum-Time Trajectories for Robot Arms, in *Proceedings of the IEEE International Conference on Robotics and Automation*, New York, 1985, pp. 751-758..

67. Salisbury, J. K., Active Stiffness Control of a Manipulator in Cartesian Coordinates, *Proceedings of the 19th IEEE Conference on Decision and Control*, December 1980.

68. Salisbury, J. K., and Craig, J. J., Articulated Hands: Force Control and Kinematic Issues, *International Journal of Robotics Research*, 1982, vol. 1, no. 1, pp. 4-17.

69. Shahinpoor, Moshen, *A Robot Engineering Textbook*, Harper and Row Poblishers, New York, 1987.

70. Shiller, Z. and S. Dubowsky, On the Optimal Control of Robotic Manipulators with Actuator and End Effector Constraints, in *Proceedings of the IEEE Conference on Robotics and Automation*, New York, 1985, pp. 614-620.

71. Shiller, Z. and S. Dubowsky, Global Time-Optimal Path Planning for Robotic Manipulators in the Presence of Obstacles, in *Proceedings of the IEEE Conference on Robotics and Automation*, Philadelphia, PA., 1988, pp. 370-375.

72. Shiller, Zvi and Hsueh-Hen Lu, Robust Computation of Constrained Time Optimal Motions, in *Proceedings of the IEEE Conference on Robotics and Automation*, Cincinati, Ohio, May, 1990, pp. 144-149.

73. Shin, K. G. and N. D. McKay, An Efficient Robot Arm Control Under Geometric Path Constraints, in *Proceedings of the 22nd IEEE Conference on Decision and Control*, New York, 1983, pp. 1449-1457.

74. Shin, K. G. and N. D. McKay, Open-Loop Minimum-Time Control of Mechanical Manipulators and its Application, in *Proceedings of the American Control Conference*, San Diego, 1984, pp. 1231-1236.

75. Shin, K. G. and N. D. McKay, Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints, *IEEE Transactions on Automatic Control* AC-30(6), June 1985, pp. 531-541.

76. Shin, K. G., and McKay, N. D., A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators, *IEEE Transactions on Automatic Control* AC-31(6), June 1986, pp. 491-500.

77. Shin, K. G. and N. D. McKay, Selection of Near-Minimum Time Geometric Paths for Robot Manipulators, *IEEE Transaction on Automatic Control* AC-31(6), June 1986, pp. 501-511.

78. Singh, S. and M. C. Leu, Optimal Trajectory Generation for Robotic Manipulators Using Dynamic Programming, *ASME Journal of Dynamic Systems, Measurement and Control*, vol 109. June 1987, pp. 88-96.

79. Skelboe, S., Computation of Rational Interval Functions, *BIT*, 14, pp. 87-95, 1974.

80. Stepanenko, Y. A., Dissipative Properties and Optimal Control of Manipulators, in *Proceedings of the Second Symposium on the Control of Artificial Limbs*, 1970, Dubrovnic Yugoslavia.

81. Thoma, Jean U., *Simulation by Bondgraphs, Introduction to a Graphical Method*, 1990.

82. Thoma, Jean U., Simulation, Realistic (Engineering), *Ecyclopedia of Physical Science and Technology*, vol. 12, pp. 680-698, Academic Press Inc., 1987.

83. *Unix Programmer's Manual Supplementary Documents Volume 1*, Computer Systems research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkley, California, 1986.

84. Valasek, M., Energetically Optimal Control of Industrial Robots, in *The Theory of Machines and Mechanisms, Proceedings of the 7th World Congress*, Sevilla, Spain, Sept. 1988, pp. 1465-1468.

85. Vukobratovic, M. and M. A. Kircanski, A Method for Optimal Synthesis of Manipulation Robot Trajectories, *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 104, June 1982, pp. 188-193.

86. Weinreb, A. and A. E. Bryson, Optimal Control of Systems with Hard Control Bounds, *IEEE Transactions on Automatic Control* AC-30(11), Nov. 1985, pp. 1135-1138.

87. Whitney, D. E., Force Feedback Control of Manipulator Fine Motions, *ASME Journal of Dynamic Systems, Measurement and Control*, June 1977, pp. 91-97.

88. Wind River Systems Inc., *VxWorks Programers Guide,* Wind River Systems, Alameda, Ca., 1984 - 1993.

89. Wind River Systems Inc., *VxWorks Reference Manual,* Wid River Systems, Alameda, Ca., 1984 - 1992.

Appendix A
Bond Graph Notation

A bond graph is a dual-signal flow diagram representation of a physical system. Bond graphs depict how power is transmitted through the system. What follows is a brief description of the most common bond graph notation. More detailed descriptions may be found in [36][37][81] and [82].

A.1 Basic Elements

The main elements used to construct system models with bond graphs are shown in Table A.1. Power bonds carry power through the system. Each power bond has an effort and a flow variable associated with it. By convention, the effort variable is written above the bond and the flow variable below. The power flowing through is bond is given by the product of its effort and flow variables. Examples of effort and flow variables for several types of systems are given in Table A.2.; A half arrow on the power bond is used to indicate the direction of positive power flow. This does not mean that the power flow in a bond is always in the direction indicated but rather is equivalent to choosing a sign convention. If the instantaneous power is negative it simply means that power is flowing in the direc-

Table A.1 Main Bond Graph Elements

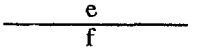
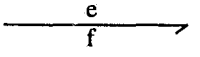
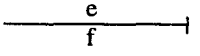
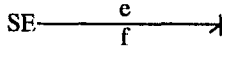
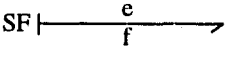
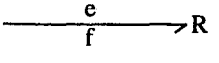
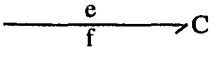
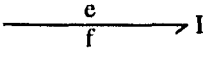
Power Bonds and Connections		
	Power bond	$P = e \times f$
	Power bond with power flow direction	
	Power bond with causality assignment. effort pushes to right, flow points to left	
	Connection, no power flow	
One-Port Elements		
	Effort source	$e = e(t)$
	Flow source	$f = f(t)$
	R - element (resistance); dissipates power	$e = Rf$ $f = e/R$
	C - element (capacitance); stores energy	$e = \frac{1}{C}\int f dt$ $f = C\frac{de}{dt}$
	I - element (inertance); stores energy	$f = \frac{1}{I}\int e dt$ $e = I\frac{df}{dt}$

Table A.1 Main Bond Graph Elements

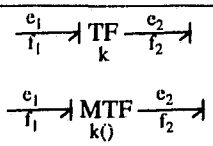
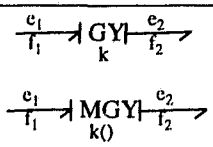
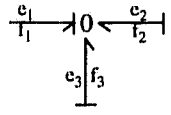
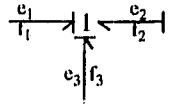
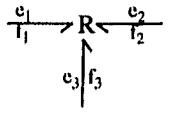
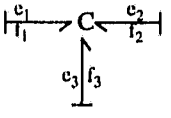
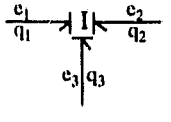
Two-Port Elements		
	transformer modulated transformer -> conserves power	$e_2 = ke_1$ $f_1 = kf_2$
	gyrator modulated gyrator -> conserves power	$e_2 = kf_1$ $e_1 = kf_2$
Multiport Elements		
	zero (parallel) junction sometimes denoted with 'p' -> conserves power	$e_1 = e_2 = e_3$ $f_1 + f_2 + f_3 = 0$
	one (series) junction sometimes denoted with 's' -> conserves power	$f_1 = f_2 = f_3$ $e_1 + e_2 + e_3 = 0$
	R field -> dissipates power	$e_1 = g_{01}(f_1, f_2, f_3)$ $e_2 = g_{02}(f_1, f_2, f_3)$ $e_3 = g_{03}(f_1, f_2, f_3)$
	C field -> stores energy	$e_1 = g_{01}(q_1, q_2, q_3)$ $e_2 = g_{02}(q_1, q_2, q_3)$ $e_3 = g_{03}(q_1, q_2, q_3)$
	I field -> stores energy	$f_1 = g_{01}(p_1, p_2, p_3)$ $f_2 = g_{02}(p_1, p_2, p_3)$ $f_3 = g_{03}(p_1, p_2, p_3)$

Table A.2 Effort and Flow Variables in Various Types of Systems

System	Effort	Flow
electric	voltage	current
linear mechanical	force	velocity
rotary mechanical	torque	angular velocity
hydraulic	pressure	volume flow rate
chemical	chemical potential	molar flow
thermal	temperature	entropy flow

tion opposite to that indicated by the half arrow. By convention, power is shown flowing into each resistive, capacitive and inertial element. The direction of power flow in the remaining bonds follows physical common sense. A transverse bar at one end of a power bond is used to make a causal selection. That is to say, the causal bar indicates which of the effort or flow variable is the independent variable for each element. More will be said about the assignment of causality in coming sections. A connection (or active bond or modulation bond) has only one of the two power variables associated with it. The other is assumed to be zero. As such, a connection does not transmit any power. Connections are often used to indicate that an element value is modulated by the variable carried by the connection.

Bond graph elements may be grouped according to the following properties.

1. The number of bonds attached to the element. There are one-port, two-port and multiport elements for elements with one, two or multiple bonds
2. Elements may conserve power, dissipate power, supply power, absorb power or store power.
3. The output power variable can be expressed either algebraically in terms of the input power variable or as a function of the integral of the input power variable.

One-port elements can not conserve power but there are two source elements which may supply or absorb power. An effort source maintains a constant effort in its bond

and supplies flow as necessary. Conversely a flow source maintains a constant flow in its bond. Effort and flow sources are bond graph analogs of the well know voltage and current sources from electric circuit analysis.

Resistive elements dissipate power irreversibly. Examples of resistive elements include friction in mechanical or hydraulic systems and resistive loses in electrical systems.

Capacitive elements absorb power and store it as energy by changing their state. The stored power is returned to the system in the transition back to the original state. A device is said to be capacitive if a net flow into the device tends to increase the effort within the device. Examples of capacitive devices include mechanical springs, hydraulic accumulators and electric capacitors.

Inertia elements also absorb power and store it as energy. Inertive devices increase their flow in response to an applied effort. Examples include mass, rotary inertia and electrical inductors.

Two-port elements may be power conserving since it is possible that the power going into the element is equivalent to the power leaving element. Two examples of power conserving two-ports are transformers and gyrators. Transformers, transform the effort variable to an effort variable and the flow variable to a flow variable. Examples are electric transformers, gear reducers and hydrostatic pumps. Gyrators are sometimes call cross-transformers since the effort variable is transformed to a flow and the flow variable is transformed to an effort. An electric motor is an example of a gyrator. Both transformers and gyrators are commonly seen in a modulated form. Here the transformation between the ports of the element is expressed as a function of some other system variable. A con-

nection bond is often seen as input to a modulated transformer (or gyrator) to indicate the system variable on which the transformation depends.

The most common multiport elements are the junctions. A zero junction is a junction of power bonds at which the flow variables sum to zero and the effort variables are equal. Sometimes zero junctions are written as parallel junctions (p-junction) in deference to the electrical analog of a parallel connected circuit. At a one junction the effort variables sum to zero and the flow variables are equal. An alternative notation uses a series junction (s-junction) in accordance with the electric analog of a series circuit. Resistive, capacitive and inertive elements also exist as multiports and are commonly referred to as R, C and I fields respectively. Examples of three-port fields are given in Table A.1. In the constitutive equations for the fields 'p' represents momentum and 'q' represents displacement.

The actions of bond graph elements can be represented by what is called the bond graph carousel. See Figure A.1. Starting from the momentum variable 'p', and moving counterclockwise through an I element yields the flow 'f'. The I element provides a functional relationship (linear or nonlinear) between momentum and flow. Continuing in a counterclockwise direction the flow is integrated to obtain the position 'q' which then goes through the functional relationship described by the C element to yield effort 'e'. The circle is closed by integrating the effort variable to momentum. The R element forms a direct functional relation between the effort and flow variables. Since integration is a more stable operation on a digital computer than integration, variables and equations should be arranged such that they traverse the carousel in a counterclockwise direction as indicated by the arrows. This is called integral causality. The exception to the counterclockwise rule is the R element which can be entered in either direction, from flow to effort (resistance causality) or from effort to flow (conductance causality). In systems with

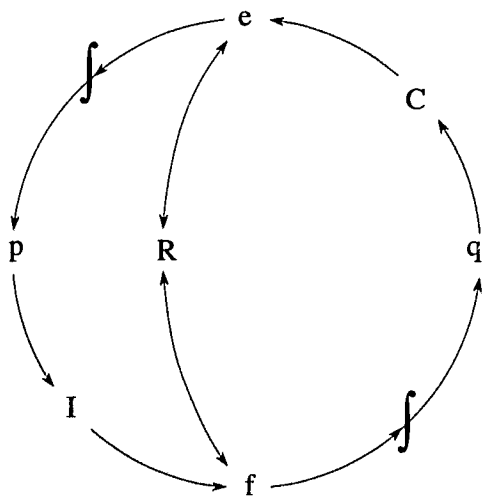


Figure A.1 Bond Graph Carousel

nonlinear resistances the distinction between these two causal forms may be significant since some types of resistive elements have a preferred computational form.

A.2 Causality

Assigning causality designates which of the power flow variables are cause variables and which are effect variables. For the purpose of digital simulation it is most appropriate to write $effect = f(cause)$. A causal decision in indicated by a short transverse bar across the power bond. Effort travels on the bond toward this causal bar and flow travels in the reverse direction. Think of the bond with the causal bar as a hypodermic needle. When the flow variable is injected into an element it is the cause, when it comes out of the element it is the effect.

The physical nature of some elements represented by bond graph notation requires the *mandatory causality* assignments shown in Table A.3. At a '0-junction' the effort vari-

ables must be equal. Therefore there can only be one effort variable at a '0-junction'. Conversely, at a '1-junction' the flow variables are equal, so only one flow variable can be input to a '1-junction'

In a transformer, the effort variable transforms to an effort variable and the flow variable to a flow variable. This means that there can only be one causal bar at a transformer. In a gyrator, the effort variable is transformed to a flow variable and the flow variable is transformed to an effort variable. Therefore there may be two or zero causal bars at a gyrator.

The final mandatory causality assignments are for the source elements. Recall that effort sources supply an effort variable and must therefore accept a flow variable. Conversely, flow sources supply a flow variable and accept an effort variable.

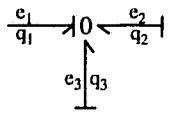
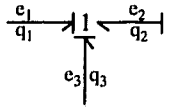
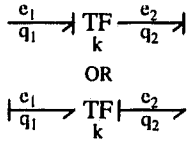
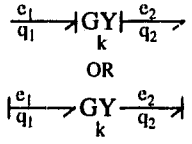
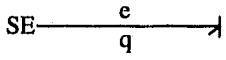
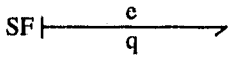
If the system is to simulated on a digital computer, then there is also a set of *preferred causality* assignments. Elements with differential relationships should be expressed in integral form since integration is a preferable operation to differentiation in a digital simulation. Preferred causality assignments are shown in Table A.3.

A.3 Multibond Graph Elements

Modelling robotic systems requires the representation of rigid body motion in Cartesian space. While this can be accomplished with standard bond graph notation (see [37] and [69]), an extension to bond graph notation called multibond graphs provides a more compact description. As shown in Figure A.2, a multibond is simply a combination of single power bonds. The power flow in the multibond is given by the sum of the power flows in the individual bonds in contains or, in vector notation,

$$P = \dot{e}^T f, \tag{A.1}$$

Table A.3 Mandatory Causality Assignment

zero junction		only one causal bar at a zero junction
one junction		only one flow variable at a one junction
transformer		only one causal bar at a transformer
gyrator		two or zero causal bars at a gyrator
effort source		no causal bar at an effort source
flow source		one causal bar at an effort source

The bond graph elements of Table A.1 have corresponding multibond forms. The elements become multiports with matrix relationships between the effort and flow variables. A complete multibond graph notation in provided in [15]. One element unique to multibond graph notation is the direct sum; shown in Figure A.3. This operation, indicated by a perpendicular stroke between separated power bonds, allows for the decomposition of multibonds into single bonds, or, conversely, the combination of single bonds into multibonds.

Figure A.2 Multibond

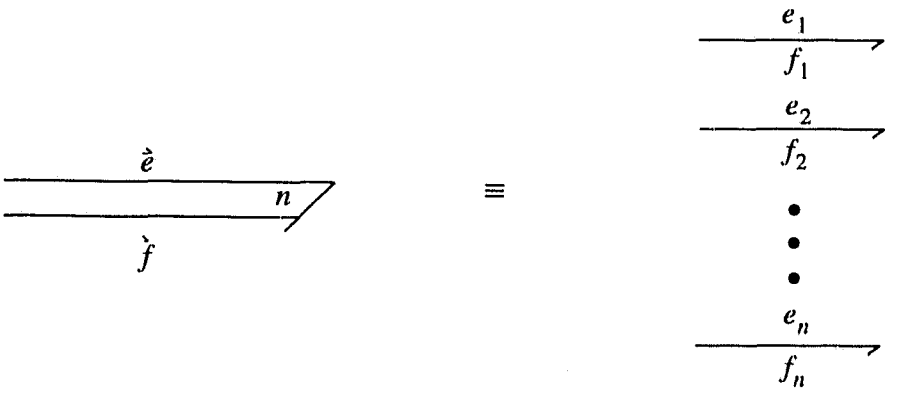


Figure A.3 Direct Sum

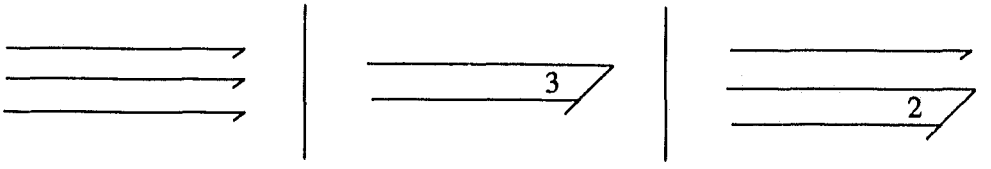
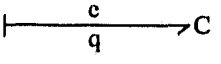
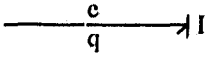


Table A.4 Preferred Causality Assignment

C-element		$E = C(Q) = \frac{1}{C}\int Q dt + E(0)$
I-element		$Q = I(E) = \frac{1}{I}\int E dt + Q(0)$

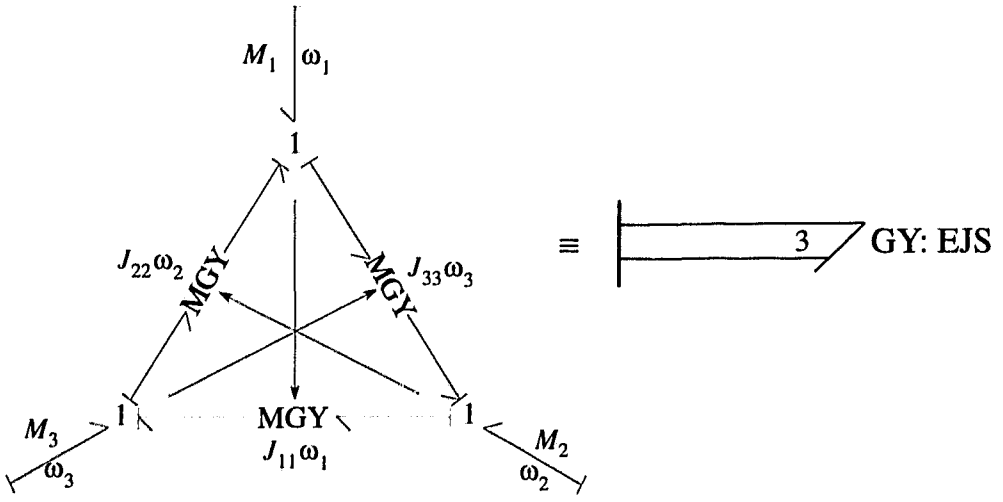


Figure A.4 Eulerian Junction Structure

Sometimes basic elements and junctions are found to commonly occur with a certain structure. It is often useful to replace the more detailed bond graph description with a more compact multibond notation. One such example is a triangular structure of one junctions and gyrators which are internally modulated by their opposing junctions. This junction structure, along with its multibond equivalent multibond representation, is shown in Figure A.4. The junction structure represents Euler's equations of motion for a rigid body in rotating coordinate frames under the following assumptions:

1. The origin of the coordinate frame is either at the body's mass center or at a point on the body fixed in inertial space.
2. The coordinate system is a body coordinate system so that its angular velocity is the same as the angular velocity of the body.
3. The axes are the principle axes of inertia.

Writing the sum of the effort variables (moments) at each 1-junction yields Euler's equations.

$$
\begin{aligned}
M_1 + J_{22}\omega_2\omega_3 - J_{33}\omega_3\omega_2 &= 0 \\
M_2 - J_{11}\omega_1\omega_3 + J_{33}\omega_3\omega_1 &= 0 \\
M_3 + J_{11}\omega_1\omega_2 - J_{22}\omega_2\omega_1 &= 0
\end{aligned}
\qquad \text{(A.2)}
$$

This structure occurs often enough that is given its own designation; "Eulerian junction structure" or EJS. It is in effect a modulated three port gyrator.

Appendix B
Reis V15 Model Parameter Estimation

B.1 Frame Assignment and Notation

Each link is assigned a body fixed frame (body frame) whose z-axis is aligned with the axis of rotation between the link and the previous link. These frames are numbered sequentially starting at the base $i = 0, 1, 2, 3$, where the zeroth frame is attached to a fixed base which we will refer to as the zeroth link. Each link, $i = 1, 2, 3$, is also assigned a body fixed frame whose origin is located at the link's mass center (mass center frame). These links are labelled iG, where i is the corresponding link number. For modelling purposes, each link is considered to be composed of a number of elements. Each element has associated with it a body fixed frame with origin at the element's mass center (element frame). These frames are denoted ijG, where i is the link number and j is the element number. Due to the symmetry of the Reis V15 structure each of the three types of frames, body, mass center and element for a given link i, can be chosen with the same orientation. Therefore to describe one frame with respect to another requires only a position vector describing the location of one frame's origin with respect to another. For this purpose we define $\vec{r}^{\,k}_{i,j}$ to be the position of point i with respect to point j written in frame k. Therefore, for example, $\vec{r}^{\,i}_{ijG,i}$ would represent the position of the origin of element frame ijG with respect to the origin of body frame i written in body frame i. The Cartesian compo-

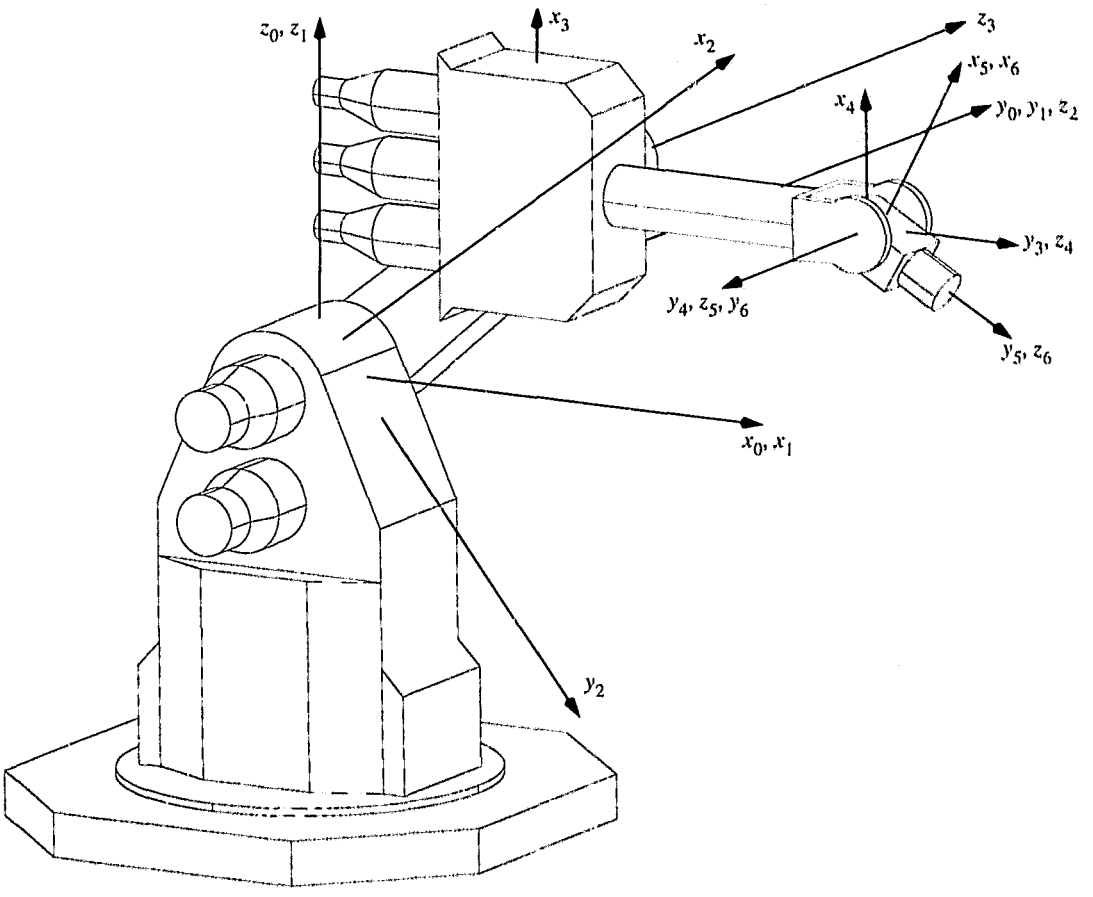


Figure B.1 Reis V15 Frame Assignment

nents of these position vectors are represented by r^k_{i,j_x}, r^k_{i,j_y}, and r^k_{i,j_z}. The link frame assignment for the Reis V15 manipulator is shown in Figure B.1.

Each link and element also have mass and rotary inertia properties associated with them. Each link mass is denoted m_i where i is the link number. Each element mass is denoted m_{ij} where i is the link number and j is the element number. Rotary inertia matrices are denoted J^k_i and J^k_{ij} for link and element inertias respectively. The letter k, represents the frame in which the inertia is specified. Since the rotary inertias will be specified in frames where axes align with the principle axes of inertia of the corresponding body, the

inertia matrices will be diagonal. It is often convenient to specify a single component of the rotary inertia by subscripting with the appropriate axis label. For example, $J^k_{ij_x}$, $J^k_{ij_y}$, $J^k_{ij_z}$.

B.2 Link Inertial Parameters

B.2.1 Link 1

The first link is modelled as a combination of three elements; an aluminum cylindrical shell to represent the link structure and two steel cylinders to represent the motor/tach/encoder units which drive joints two and three.

Table B.1 Link 1 Element Descriptions

	element #1	**element #2**	**element #3**
type	cylindrical shell	cylinder	cylinder
material	aluminum[a]	steel[b]	steel
length l (m)	0.77	0.16	$0.16m$
radius a (m)	0.13	0.07	$0.07m$
thickness t (m)	0.008		
mass m_{ij} (kg)	13.5	19.4	19.4
mass center $\hat{r}^1_{1jG,1}$ (m)	$(0, 0, -0.245)$	$(0, -0.16, -0.215)$	$(0, -0.16, 0)$

a. $\rho_{aluminum} = 2770\ kg/m^3$
b. $\rho_{steel} = 7860\ kg/m^3$

The first link's mass is

$$m_1 = m_{11} + m_{12} + m_{13} = 52.3\ kg. \tag{B.1}$$

The first links mass center is located by,

$$m_1 \vec{r}^1_{1G,1} = \sum_{i=1}^{3} m_{1i} \vec{r}^1_{1iG,1}$$

$$m_1 \vec{r}^1_{1G,1} = m_{11} \vec{r}^1_{11G,1} + m_{12} \vec{r}^1_{12G,1} + m_{13} \vec{r}^1_{13G,1}. \tag{B.2}$$

Therefore,

$$\vec{r}^1_{1G,1} = \begin{bmatrix} 0 & -0.119 & -0.143 \end{bmatrix}. \tag{B.3}$$

The rotational inertia of the link is given by combining the rotational inertia of each element of the link model. The rotary inertia of each basic element with respect to its own frame, $1iG$, is determined from formulae found in most rigid body dynamics texts.[4] The parallel axis theorem is used to compensate for the fact that each element's mass center is not coincident with the link's mass center.

Element #1

The position of the link's mass center with respect to the first elements mass center is given by,

$$\vec{r}^1_{1G,11G} = \vec{r}^1_{1G,1} - \vec{r}^1_{11G,1} = \begin{bmatrix} 0 & -0.119 & 0.102 \end{bmatrix}. \tag{B.4}$$

Therefore the inertia about each axes of frame $1G$ is,

$$J^{1G}_{11x} = \{ \frac{1}{12} \pi a^2 l \rho_{aluminum} (3a^2 + l^2) - \frac{1}{12} \pi (a-t)^2 l \rho_{aluminum} (3[a-t]^2 + l^2) \} + m_{11} (0.119^2 + 0.102^2) = 2.92 \ \ kgm^2 \tag{B.5}$$

$$J^{1G}_{11y} = \{ \frac{1}{12} \pi a^2 l \rho_{aluminum} (3a^2 + l^2) - \frac{1}{12} \pi (a-t)^2 l \rho_{aluminum} (3[a-t]^2 + l^2) \} + m_{11} (0.102^2) = 2.73 \ \ kgm^2 \tag{B.6}$$

$$J_{11z}^{1G} = \{\frac{1}{2}\pi a^2 l \rho_{aluminum} a^2 - \frac{1}{2}\pi (a-t)^2 l \rho_{aluminum} (a-t)^2\} + m_{11}(0.119^2)$$
$$= 0.406 \ \ kgm^2 \tag{B.7}$$

A similar procedure is followed for elements two and three.

Element #2

$$\vec{r}_{1G,12G}^{1} = \vec{r}_{1G,1}^{1} - \vec{r}_{12G,1}^{1} = [0 \ -0.041 \ 0.072]. \tag{B.8}$$

$$J_{12x}^{1G} = \frac{1}{12} m_{12}(3a^2 + l^2) + m_{12}(0.041^2 + 0.072^2) = 0.198 \ \ kgm^2 \tag{B.9}$$

$$J_{12y}^{1G} = \frac{1}{2} m_{12} a^2 + m_{12}(0.072^2) = 0.148 \ \ kgm^2 \tag{B.10}$$

$$J_{12z}^{1G} = \frac{1}{12} m_{12}(3a^2 + l^2) + m_{12}(0.041^2) = 0.099 \ \ kgm^2 \tag{B.11}$$

Element #3

$$\vec{r}_{1G,13G}^{1} = \vec{r}_{1G,1}^{1} - \vec{r}_{13G,1}^{1} = [0 \ 0.041 \ -0.143]. \tag{B.12}$$

$$J_{13x}^{1G} = \frac{1}{12} m_{13}(3a^2 + l^2) + m_{13}(0.041^2 + 0.143^2) = 0.494 \ \ kgm^2 \tag{B.13}$$

$$J_{13y}^{1G} = \frac{1}{2} m_{13} a^2 + m_{13}(0.143^2) = 0.444 \ \ kgm^2 \tag{B.14}$$

$$J_{13z}^{1G} = \frac{1}{12} m_{13}(3a^2 + l^2) + m_{13}(0.041^2) = 0.098 \ \ kgm^2 \tag{B.15}$$

Combining the estimated parameters for the elements gives the following estimated inertial parameters for the first link.

$$m_1 = 52.3 \ kg \tag{B.16}$$

$$J_1^{1G} = \begin{bmatrix} 3.61 & 0 & 0 \\ 0 & 3.32 & 0 \\ 0 & 0 & 0.60 \end{bmatrix}. \tag{B.17}$$

B.2.2 Link 2

The second link is modelled as a combination of two elements; an aluminum rectangular prism shell to represent the link structure and a point mass to represent the harmonic drive gear reducer which drives the third link.

Table B.2 Link 2 Element Descriptions

	element #1	**element #2**
type	rectangular prism shell	point mass
material	aluminum	
length l (m)	0.6	
width w (m)	0.175	
height h (m)	0.3	
thickness t (m)	0.008	
mass m_{ij} (kg)	14.2	4.1
mass center $\hat{r}^2_{2jG,2}(m)$	[0.3 0 −0.18]	[0.6 0 −0.18]

The link's mass is

$$m_2 = m_{21} + m_{22} = 18.3 \ kg. \tag{B.18}$$

The mass center is located by,

$$m_2 \hat{r}^2_{2G,2} = \sum_{i=1}^{2} m_{2i} \hat{r}^2_{2iG,2}$$

$$m_2 \hat{r}^2_{2G,2} = m_{21} \hat{r}^2_{21G,2} + m_{22} \hat{r}^2_{22G,2} \tag{A.1}$$

Therefore,

$$\hat{r}^2_{2G,2} = \begin{bmatrix} 0.37 & 0 & -0.18 \end{bmatrix}. \tag{B.2}$$

The rotational inertias due to each element are as follows.

Element #1

$$\hat{r}^2_{2G,21G} = \hat{r}^2_{2G,2} - \hat{r}^2_{21G,2} = \begin{bmatrix} 0.07 & 0 & 0 \end{bmatrix}. \tag{B.3}$$

$$J^{2G}_{21x} = \frac{1}{12} wlh\rho_{aluminum} (w^2 + h^2) - \frac{1}{12} (w - 2t)\ (l - 2t)\ (h - 2t)\, \rho_{aluminum}$$

$$(\,[w - 2t]^2 + [h - 2t]^2) = 0.231 \ \ kgm^2 \tag{B.4}$$

$$J^{2G}_{21y} = \frac{1}{12} wlh\rho_{aluminum} (w^2 + l^2) - \frac{1}{12} (w - 2t)\ (l - 2t)\ (h - 2t)\, \rho_{aluminum}$$

$$(\,[w - 2t]^2 + [l - 2t]^2) + m_{21} (0.07^2) = 0.676 \ \ kgm^2 \tag{B.5}$$

$$J^{2G}_{21z} = \frac{1}{12} wlh\rho_{aluminum} (l^2 + h^2) - \frac{1}{12} (w - 2t)\ (l - 2t)\ (h - 2t)\, \rho_{aluminum}$$

$$(\,[l - 2t]^2 + [h - 2t]^2) + m_{21} (0.07^2) = 0.766 \ \ kgm^2 \tag{B.6}$$

Element #2

$$\hat{r}^2_{2G,22G} = \hat{r}^2_{2G,2} - \hat{r}^2_{22G,2} = \begin{bmatrix} -0.23 & 0 & 0 \end{bmatrix}. \tag{B.7}$$

$$J^{2G}_{22x} = 0.0 \ \ kgm^2 \tag{B.8}$$

$$J_{22y}^{2G} = m_{22}(0.23^2) = 0.216 \ \ kgm^2 \tag{B.9}$$

$$J_{22z}^{2G} = m_{22}(0.23^2) = 0.216 \ \ kgm^2 \tag{B.10}$$

Combining the estimated parameters for the elements gives the following estimated inertial parameters for the second link.

$$m_2 = 18.3 \ \ kg \tag{B.11}$$

$$J_2^{2G} = \begin{bmatrix} 0.231 & 0 & 0 \\ 0 & 0.892 & 0 \\ 0 & 0 & 0.982 \end{bmatrix}. \tag{B.12}$$

B.2.3 Link 3

The third link is modelled as a combination of nine elements; an aluminum rectangular prism shell and a cylindrical shell to represent the link structure, three steel cylinders to represent the motor/tach/encoder units which drive the last three joints of the manipulator, three point masses to represent the harmonic drive units for the last three joints of the arm and a point mass for the end effector. These elements are described in Table B.3.

The link's mass is

$$m_3 = \sum_{i=1}^{9} m_{3i} = 28.5 \ \ kg. \tag{B.13}$$

The link's mass center is located by,

$$m_3 \hat{r}_{3G,3}^{3} = \sum_{i=1}^{9} m_{3i} \hat{r}_{3iG,3}^{3}. \tag{B.14}$$

Therefore,

Table B.3 Link 3 Element Descriptions

	element #1	**element #2**	**element #3**	**element #4**	**element #5**	**element #6**	**element #7**	**element #8**	**element #9**
type	rectangular prism shell	cylinder	cylinder	cylinder	cylindrical shell	point mass	point mass	point mass	point mass
material	aluminum	steel	steel	steel	aluminum				
length l (m)	0.25	0.13	0.13	0.13	0.38				
width w (m)	0.135								
height h (m)	0.32								
thickness t (m)	0.01				0.01				
radius a (m)		0.05	0.05	0.05	0.05				
mass m_{ij} (kg)	7.9	8.0	8.0	8.0	3.0	1.0	1.0	1.0	2.0
mass center $\hat{r}^3_{3jG,3}$ (m)	$[0\ 0\ 0]$	$[-0.18\ 0\ 0]$	$[-0.18\ 0.13\ 0]$	$[-0.18\ -0.13\ 0]$	$[0.35\ 0\ 0]$	$[0.18\ 0\ 0]$	$[0.551\ 0\ 0]$	$[0.63\ 0\ 0]$	$[0.75\ 0\ 0]$

$$\vec{r}^{\,3}_{3G,3} = \begin{bmatrix} -0.0314 & 0 & 0 \end{bmatrix}. \tag{B.15}$$

The rotational inertias due to each element are as follows.

Element #1

$$\vec{r}^{\,3}_{3G,31G} = \vec{r}^{\,3}_{3G,3} - \vec{r}^{\,3}_{31G,3} = \begin{bmatrix} -0.0314 & 0 & 0 \end{bmatrix}. \tag{B.16}$$

$$J^{3G}_{31x} = \frac{1}{12} wlh\rho_{aluminum}(w^2 + h^2) - \frac{1}{12}(w-2t)\,(l-2t)\,(h-2t)\,\rho_{aluminum}$$

$$(\,[w-2t]^2 + [h-2t]^2) = 0.116 \;\; kgm^2 \tag{B.17}$$

$$J^{3G}_{31y} = \frac{1}{12} wlh\rho_{aluminum}(w^2 + l^2) - \frac{1}{12}(w-2t)\,(l-2t)\,(h-2t)\,\rho_{aluminum}$$

$$(\,[w-2t]^2 + [l-2t]^2) + m_{31}(0.0314^2) = 0.088 \;\; kgm^2 \tag{B.18}$$

$$J^{3G}_{31z} = \frac{1}{12} wlh\rho_{aluminum}(l^2 + h^2) - \frac{1}{12}(w-2t)\,(l-2t)\,(h-2t)\,\rho_{aluminum}$$

$$(\,[l-2t]^2 + [h-2t]^2) + m_{31}(0.0314^2) = 0.15, \;\; kgm^2 \tag{B.19}$$

Element #2

$$\vec{r}^{\,3}_{3G,32G} = \vec{r}^{\,3}_{3G,3} - \vec{r}^{\,3}_{32G,3} = \begin{bmatrix} 0.149 & 0 & 0 \end{bmatrix}. \tag{B.20}$$

$$J^{3G}_{32x} = \frac{1}{2} m_{32} a^2 = 0.00526 \;\; kgm^2 \tag{B.21}$$

$$J^{3G}_{32y} = \frac{1}{12} m_{32}(3a^2 + l^2) + m_{32}(0.149^2) = 0.105 \;\; kgm^2 \tag{B.22}$$

$$J^{3G}_{32z} = \frac{1}{12} m_{32}(3a^2 + l^2) + m_{32}(0.149^2) = 0.105 \;\; kgm^2 \tag{B.23}$$

Element #3

$$\vec{r}^{\,3}_{3G,\,33G} = \vec{r}^{\,3}_{3G,\,3} - \vec{r}^{\,3}_{33G,\,3} = \begin{bmatrix} 0.149 & -0.13 & 0 \end{bmatrix}. \quad \text{(B.24)}$$

$$J^{3G}_{33x} = \frac{1}{2} m_{33} a^2 + m_{33}(0.13^2) = 0.0764 \ \ kgm^2 \quad \text{(B.25)}$$

$$J^{3G}_{33y} = \frac{1}{12} m_{33}(3a^2 + l^2) + m_{33}(0.149^2) = 0.105 \ \ kgm^2 \quad \text{(B.26)}$$

$$J^{3G}_{33z} = \frac{1}{12} m_{33}(3a^2 + l^2) + m_{33}(0.149^2 + 0.13^2) = 0.176 \ \ kgm^2 \quad \text{(B.27)}$$

Element #4

$$\vec{r}^{\,3}_{3G,\,34G} = \vec{r}^{\,3}_{3G,\,3} - \vec{r}^{\,3}_{34G,\,3} = \begin{bmatrix} 0.149 & 0.13 & 0 \end{bmatrix}. \quad \text{(B.28)}$$

$$J^{3G}_{33x} = \frac{1}{2} m_{33} a^2 + m_{33}(0.13^2) = 0.0764 \ \ kgm^2 \quad \text{(B.29)}$$

$$J^{3G}_{33y} = \frac{1}{12} m_{33}(3a^2 + l^2) + m_{33}(0.149^2) = 0.105 \ \ kgm^2 \quad \text{(B.30)}$$

$$J^{3G}_{33z} = \frac{1}{12} m_{33}(3a^2 + l^2) + m_{33}(0.149^2 + 0.13^2) = 0.176 \ \ kgm^2 \quad \text{(B.31)}$$

Element #5

$$\vec{r}^{\,3}_{3G,\,35G} = \vec{r}^{\,3}_{3G,\,3} - \vec{r}^{\,3}_{35G,\,3} = \begin{bmatrix} -0.3814 & 0 & 0 \end{bmatrix}. \quad \text{(B.32)}$$

$$J^{3G}_{35x} = \left\{ \frac{1}{2}\pi a^2 l \rho_{aluminum} a^2 - \frac{1}{2}\pi (a-t)^2 l \rho_{aluminum} (a-t)^2 \right\} = 0.0061 \ \ kgm^2 \quad \text{(B.33)}$$

$$J_{35y}^{3G} = \{\frac{1}{12}\pi a^2 l\rho_{aluminum}(3a^2+l^2) - \frac{1}{12}\pi(a-t)^2 l\rho_{aluminum}(3[a-t]^2+l^2)\} + m_{35}(0.3814^2) = 0.475 \ \ kgm^2 \quad \text{(B.34)}$$

$$J_{35z}^{3G} = \{\frac{1}{12}\pi a^2 l\rho_{aluminum}(3a^2+l^2) - \frac{1}{12}\pi(a-t)^2 l\rho_{aluminum}(3[a-t]^2+l^2)\} + m_{35}(0.3814^2) = 0.475 \ \ kgm^2 \quad \text{(B.35)}$$

Element #6

$$\hat{r}_{3G,36G}^{3} = \hat{r}_{3G,3}^{3} - \hat{r}_{36G,3}^{3} = \begin{bmatrix}-0.2114 & 0 & 0\end{bmatrix}. \quad \text{(B.36)}$$

$$J_{36x}^{3G} = 0.0 \ \ kgm^2 \quad \text{(B.37)}$$

$$J_{36y}^{3G} = m_{36}(0.2114^2) = 0.045 \ \ kgm^2 \quad \text{(B.38)}$$

$$J_{36z}^{3G} = m_{36}(0.2114^2) = 0.045 \ \ kgm^2 \quad \text{(B.39)}$$

Element #7

$$\hat{r}_{3G,37G}^{3} = \hat{r}_{3G,3}^{3} - \hat{r}_{37G,3}^{3} = \begin{bmatrix}-0.5284 & 0 & 0\end{bmatrix}. \quad \text{(B.40)}$$

$$J_{37x}^{3G} = 0.0 \ \ kgm^2 \quad \text{(B.41)}$$

$$J_{37y}^{3G} = m_{37}(0.5284^2) = 0.279 \ \ kgm^2 \quad \text{(B.42)}$$

$$J_{37z}^{3G} = m_{37}(0.5284^2) = 0.279 \ \ kgm^2 \quad \text{(B.43)}$$

Element #8

$$\hat{r}_{3G,38G}^{3} = \hat{r}_{3G,3}^{3} - \hat{r}_{38G,3}^{3} = \begin{bmatrix}-0.6614 & 0 & 0\end{bmatrix}. \quad \text{(B.44)}$$

$$J_{38x}^{3G} = 0.0 \ \ kgm^2 \tag{B.45}$$

$$J_{38y}^{3G} = m_{38}(0.6614^2) = 0.437 \ \ kgm^2 \tag{B.46}$$

$$J_{38z}^{3G} = m_{38}(0.6614^2) = 0.437 \ \ kgm^2 \tag{B.47}$$

Element #9

$$\hat{r}_{3G,\,39G}^{3} = \hat{r}_{3G,\,3}^{3} - \hat{r}_{39G,\,3}^{3} = \begin{bmatrix} -0.7814 & 0 & 0 \end{bmatrix}. \tag{B.48}$$

$$J_{39x}^{3G} = 0.0 \ \ kgm^2 \tag{B.49}$$

$$J_{39y}^{3G} = m_{39}(0.7814^2) = 1.22 \ \ kgm^2 \tag{B.50}$$

$$J_{39z}^{3G} = m_{39}(0.7814^2) = 1.22 \ \ kgm^2 \tag{B.51}$$

Combining the estimated parameters for the elements gives the following estimated inertial parameters for the third link.

$$m_3 = 28.5 \ \ kg \tag{B.52}$$

$$J_3^{3G} = \begin{bmatrix} 0.50 & 0 & 0 \\ 0 & 3.29 & 0 \\ 0 & 0 & 3.69 \end{bmatrix}. \tag{B.53}$$

Appendix C
Reis V15 I/O Card Memory Map

Table C.1 shows the memory map as seen from the VME bus for the two I/O cards used to interface to the Reis V15 hardware. The table gives the device, the card on which the device interface is located, the devices hexadecimal address and the variable type used to access the address. Note that some devices appear to have the same address. This apparent double mapping is possible since some of the devices also require that the read or write lines be high depending on whether they are read only or write only devices. This is also indicated in the table where, 'r/w' means the address can be both read from and written to, 'r' indicates a read only address and 'w' indicates a write only address.

Each I/O card has a single 8 bit analog to digital converter these are unused at this time.

The servo enable relays must be closed for there to be power to the motor amplifier cards and thereby to the motors and the actuators which hold the brakes on the second and third joints off. The relays are enabled by writing one to the servo enable address on each

Table C.1 Reis V15 I/O Card Memory Map

Device	Card No.	Address	Type	R/W
A/D Converters	card 0	0x00fc0180	unsigned char *	r/w
	card 1	0x00fc0188	unsigned char *	r/w
Servo Enable Relays	card 0	0x00fc0281	unsigned char *	w
	card 1	0x00fc0289	unsigned char *	w
Joint Encoders	card 0, joint 1	0x00fc0000	unsigned short *	r/w
	card 0, joint 2	0x00fc0002	unsigned short *	r/w
	card 0, joint 3	0x00fc0004	unsigned short *	r/w
	card 0, joint 4	0x00fc0006	unsigned short *	r/w
	card 1, joint 5	0x00fc0008	unsigned short *	r/w
	card 1, joint 6	0x00fc000a	unsigned short *	r/w
Joint Servo Drivers	card 0, joint 1	0x00fc0080	short *	r/w
	card 0, joint 2	0x00fc0082	short *	r/w
	card 0, joint 3	0x00fc0084	short *	r/w
	card 0, joint 4	0x00fc0086	short *	r/w
	card 1, joint 5	0x00fc0088	short *	r/w
	card 1, joint 6	0x00fc008a	short *	r/w
Status Register	card 0	0x00fc0301	unsigned char *	r
	card 1	0x00fc0309	unsigned char *	r
Watchdog Timer Reset	card 0	0x00fc0301	unsigned char *	w
	card 1	0x00fc0309	unsigned char *	w
Home Status	card 0, joint 1 to 4	0x00fc0381	unsigned char *	r
	card 1, joint 5 to 6	0x00fc0389	unsigned char *	r
Home Latch Enable	card 0, joint 1 to 4	0x00fc0001	unsigned char *	w
	card 1, joint5 to 6	0x00fc0009	unsigned char *	w
Binary Input Latch	card 0	0x00fc0207	unsigned char *	w
	card 1	0x00fc020f	unsigned char *	w
Binary Input Bytes		0x00fc0201	unsigned char *	r
Binary Output Bytes		0x00fc0201	unsigned char *	w

of the I/O cards. In the event that the software detects a fatal error condition, zero is written to these addresses to stop the robot's motion.

The joint encoders keep track of the manipulator's joint angles. The trains of pulses from the joint encoders increment or decrement sixteen bit counters on the Reis I/O cards. Since it is possible that the counters may roll under their bottom ends or over their top ends, a software encoder count must be maintained. At each cycle of the controller, the difference between the current and last hardware encoder count is added to the software encoder count. If the change in the hardware encoder count has increased by more than half of the counter range, 32767, then the counter has rolled under the bottom end and 65536 is subtracted from the change before adding it to the software encoder count. If the change is less than -32768, the hardware encoder count has rolled over the top end and 65536 is added to the change before it is added to the software count. The constant of proportionality between the software encoder readings and the joint angles is $5.0 \times 10^{-6}\pi$ radians. Note that to obtain the actual joint angles, offsets equal to the joint angles at the home position must be added. In addition some joints require compensation for the kinematic coupling in their transmissions as outlined in Section 6.3.

The joint servo drivers are the addresses to which the controller writes integer numbers proportional to the desired level of drive signal for each joint. The Reis amplifier cards allows input voltages in the range $\pm 10\ V$ which corresponds to a motor velocity command range of ±4000 r.p.m.. Integer values of ±2000 are written to the servo driver addresses to provide this range of commanded velocity. The integer drive value has been limited in software to have a maximum absolute value of 1000 as a safety precaution.

Each of the Reis I/O cards contains an eight bit register which reports on the status of the interface between the Reis I/O cards and the motor amplifiers. If an interface failure

occurs the first bit of the register will be set to one. As an added precaution the fourth bit of the register is set to one whenever a count down counter reaches zero. The input to this counter in tied to an on board clock pulse. If either of these bits are set the power to the servo amplifiers is turned off. The count down timer forces the control program to reset the counter periodically before they reach zero. This ensures that if the control program fails, the manipulator will be shut down within a known maximum time period. The maximum time to shutdown without the counters being reset in controlled by the number written to the watchdog timer reset address. A hexadecimal value of 0xff provides for a maximum time before a counter reset must occur of approximately 20 milliseconds.

The home status and home latch enable registers are used in the control of the homing process. When the manipulator is first powered up the contents of the software encoder variables bear no relation to the actual joint angles. The manipulator must find a reference pulse on each of its motor's encoders so that the encoder counts can be referenced to a known absolute angular position. The upper four bits of the home status register indicate which side of home each of the four joints allocated to the I/O card are on. A bit set to zero indicates that the corresponding joint is on the positive side of home. A bit set to one indicates that the corresponding joint is on the negative side of home. The lower four bits of the home status register, if enabled, latch low on the home reference pulse from the encoder. The upper four bits of the home latch enable must be kept high. The lower four bits enable (set to one), or disable (set to zero) the latching of the lower four bits of the home status register when the home reference pulse in detected. Each joint is homed by driving it towards the +/- edge indicated by its bit in the upper four bits of the home status register. When this bit flips, the joint will either be just on the positive or negative side of this edge. Since the reference home pulse is always located on the negative side of this edge, the homing program will continue to drive until the joint ends up on the

positive side of the edge. This means that if after the first time the bit flips it is one, the homing routine will drive the joint back in the negative direction until the bit flips back to zero. At this point the joint will be just on the positive side of its +/- edge. This procedure ensures that the home reference will always be approached from the same side. The bit in the lower four bits of the home latch enable register which corresponds to the joint is set to enable (set to one) and the joint is driven slowly in the negative direction. When the reference home pulse is detected the corresponding bit in the lower four bits of the home status register latches to zero. At this point the software encoder is zeroed and the joint commanded to hold its position until the remaining joints have completed the homing sequence.

The Reis V15 has 64 twenty four volt binary inputs connected to the external world with solid state relays. The state of these relays is latched into the binary input bytes by writing zero to the binary input latch. The bits are assigned as shown in Table C.2.

Table C.2 Binary Inputs

Bit	Address	Name	Function
0	0x00fc0201	system_L	emergency stop
1			air off
2			servo off
3			program stop
4			program auto
5			program start
6			error acknowledge
7			reserved

Table C.2 Binary Inputs

Bit	Address	Name	Function
8	0x00fc0202	system_H	reserved
9			reserved
10			reserved
11			reserved
12			power fail
13			reserved
14			reserved
15			reserved
16	0x00fc0203	peri_1_L	cycle enable
17			unused
18			unused
19			unused
20			unused
21			unused
22			unused
23			unused
24	0x00fc0204	peri_1_H	unused
25			unused
26			unused
27			unused
28			unused
29			unused
30			unused
31			unused
32	0x00fc0205	peri_2_L	unused
33			unused
34			unused
35			unused
36			unused
37			unused
38			unused
39			unused

Table C.2 Binary Inputs

Bit	Address	Name	Function
40	0x00fc0206	peri_2_H	unused
41			unused
42			unused
43			unused
44			unused
45			unused
46			unused
47			unused
48	0x00fc0207	peri_3_L	unused
49			unused
50			unused
51			unused
52			unused
53			unused
54			unused
55			unused
56	0x00fc0208	peri_3_H	unused
57			unused
58			unused
59			unused
60			unused
61			unused
62			unused
63			unused

The Reis V15 may also drive 48, twenty-four volt binary outputs through a set of solid state output relays. These are driven by simply writing a one or zero to the corresponding bit. The binary output bits are assigned as shown in Table C.3.

Table C.3 Binary Outputs

Bit	Address	Name	Function
0	0x00fc0201	system_L	enable controller
1			program stop lamp
2			error lamp
3			hour meter
4			program start lamp
5			reserved
6			reserved
7			reserved
8	0x00fc0202	system_H	reserved
9			reserved
10			reserved
11			brake
12			reserved
13			reserved
14			reserved
15			reserved
16	0x00fc0203	peri_1_L	gripper line 1
17			unused
18			unused
19			unused
20			unused
21			unused
22			unused
23			unused
24	0x00fc0204	peri_1_H	unused
25			unused
26			unused
27			unused
28			unused
29			unused
30			unused
31			unused

Table C.3 Binary Outputs

Bit	Address	Name	Function
32	0x00fc0205	peri_2_L	unused
33			unused
34			unused
35			unused
36			unused
37			unused
38			unused
39			unused
40	0x00fc0206	peri_2_H	unused
41			unused
42			unused
43			unused
44			unused
45			unused
46			unused
47			unused

Appendix D
RTI-600 A/D Card Setup

The RTI-600 setup includes multiplexer configuration, gain selection, conversion delay, A/D input range, output code selection, and address mapping. A detailed description of the board and its setup can be found in the RTI-600 User's Manual.[3] This appendix gives the setup parameters for this experimental work.

The RTI-600 is configured for 32 single ended inputs. This requires the addition of two additional multiplexer integrated circuits (part ADI PIN 0A10, from Analog Devices) and the following wire wrap jumper configurations.

Table D.1 Multiplexer Jumper Settings

34 to 37 to 41
35 to 36

The resistor programmable gain amplifier (PGA) allows the user to amplify low level input signals to the full scale range of the input range of the A/D converter. Setting

the gain is accomplished by installing a single high stability metal film type resistor. The relationship between the gain setting G and the gain setting resistor R_G is,

$$R_G = \frac{20k\Omega}{G-1} \tag{D.54}$$

A gain of one is achieved by leaving the resistor connections for R_G open. This provides an effective input range of 0 to 10 V in unipolar mode or ± 10 V in bipolar mode.

The A/D conversion delay provides a delay from the time the multiplexer channel is selected and the time the A/D conversion takes place. It is used to allow time for the multiplexer and sample and hold to settle before conversion. This delay time must be increased for higher input gains by adding a capacitor. For the desired unity gain, the manufacturer advises that the capacitor be left open, resulting in a conversion delay of $13\mu s$.

The actual input range of the A/D is either 0 to 10 V (unipolar mode) or ± 10 V (bipolar mode). Bipolar mode is selected with the following jumper settings.

Table D.2 Input Range Jumper Settings

27 to 29
39 to 40

With the range selected, the output code of the digital data presented to the computer must be selected. The offset binary format is selected with the following jumpers.

Table D.3 A/D Output Code Jumper Settings.

19 to 20
22 to 23

The base address is selected by wire wrap jumpers. The RTI-600 is positioned in the VME short I/O address space. It base address has the form FFFFYZ00, where Y and Z are jumper selectable. In our application Y is set to B and Z to 4 using the following jumper settings.

Table D.4 Base Address Selection Jumper Settings.

3 to 4
9 to 10
13 to 14
15 to 16

Appendix E
Force Sensor Modifications

The force sensor is a Lord F/T Series Model 30/100.[1] It measures forces up to 30 lb along three orthogonal axes and torques up to 100 $in \cdot lb$ about these same axes. The system has two major components; a transducer unit and a controller unit.

The transducer unit is mounted on the robot's wrist. The robot's end effector is mounted on the transducer's end plate which is attached to a cross supported at four points around the transducer's periphery. Eight pairs of strain gauges epoxied to the cross structure are used to provide a measure of the applied forces and torques. Each of the eight strain gauge pair signals are made available to the controller unit one at a time through a 50 foot analog cable connected to the controller unit. The transducer unit contains the necessary electronics to multiplex the eight strain gauge pairs and amplify their signals to the level required.

---

1. Lord Technologies is now Assurance Technologies Inc.

The controller unit reads each of the eight strain gauge pair signals in turn. It converts these voltages into digital form and then premultiplies the resulting 8x1 vector of readings with a 6x8 calibration matrix to give a 6x1 vector of forces and torques.

$$\vec{F} = \left[f_x f_y f_z \tau_x \tau_y \tau_z\right]^T \quad \text{(E.1)}$$

The controller unit can further process the data to provide reads with a specified sensor bias or with a given tool transformation. The data is made available in one of a variety of user selectable formats through 19,200 baud serial port. The user selectable settings are communicated to the controller unit through the serial port using ATI's proprietary Force Torque Language (FTL). Details of the language and the sensors operation as shipped by the manufacturer can be found in the sensors's manual.[46]

This configuration provided by the manufacturer was not suitable for implementation on our system. The biggest problem was the level of noise. Unmodified, this system has $\pm 10\ N$ of noise on the force signals when running with our Reis V15 robot. The majority of this noise comes from the pulse width modulated drives. A secondary problem was that reading force data through the serial port provided us with a maximum update rate of only 80 Hz. This is on the low side for most force control strategies even if the signal is clean. It is particularly low if additional signal filtering is required to reduce noise levels.

To reduce the level of noise getting into the system, the analog to digital conversion was moved as close as possible to transducer unit. This is accomplished using a three component system; the transducer unit, an A/D converter board and a VME bus A/D controller board. The transducer unit remains unchanged. Power to the transducer is supplied by a DC to DC converter on the A/D conversion board. The A/D conversion board is mounted on the third link of the robot, thereby shortening the analog connecting cable

between the transducer unit and the A/D converter to three feet. Each of the eight strain gauge pairs on the transducer is selected in turn by a three bit address passed though the A/D converter board from the VME bus A/D controller board. As each strain gauge pair is addressed, its voltage signal is made available to the A/D conversion board, which performs the analog to digital conversion and then passes the digital data back to the VME bus A/D controller board where its is stored is bus addressable memory. One of the VME bus CPU's is then assigned the task of multiplying by the calibration matrix and performing any tool transformations or sensor biasing.

Details of the hardware design done by research assistant Qing Zhang are shown in Figure E.1 through Figure E.5. The essential idea of the design is shown in the block diagrams of Figure E.1 and Figure E.2. The important concept in the design is the idea of address advance. Soon after a strain gauge pair voltage signal is sampled and held, and the conversion begins, the address on the A/D converter board is advanced to the next strain gauge pair so that the signal stabilizes while the conversion process for the previous strain gauge pair is going on. This provides a significant times savings. Consider that a single A/D conversion takes 25 μs and that is takes 20 μs for the strain gauge signal to stabilize. With the addition of other system costs, a single conversion would take approximately 50 μs. if done in this serial manner. By using the address advance technique, stabilizing the next strain gauge signal is overlapped with performing the conversion on the current strain gauge signal, for a total time of approximately 30 μs. This becomes even more significant given that it takes eight strain gauge pair readings to completely update the force sensor data.

The schematic of the A/D controller board is shown in Figure E.3. It is assembled on a VME6159 prototyping board. This makes the task simpler since the VME bus inter-

face is already provided on the board. The base address of the board is mapped to FFFFC000.

The schematic of the A/D converter board is shown in Figure E.4. It is assembled on a two layer printed circuit board manufactured by technicians in the Department of Electrical Engineering at the University of Victoria.

The cabling used to connect the three components of the force/torque sensing system is shown if Figure E.5. The cable between the VME bus A/D controller board and the A/D converter board is a straight through cable with Amphenol DB50 connectors on each end. The pin out is shown on the P1 connector attached to A/D controller card. The cable which carries the analog signals between the transducer unit has an Amphenol DB15 connector at the A/D converter board end and the original round connector shown in the figure at the transducer end. The connections are as indicated in the figure.

The analog to digital conversion is controlled in software by performing only reads. The memory map is shown in Table E.1. Reading from AD_Start starts the sequence of conversion process on each strain gauge pair voltage. The strain gauge voltages are sampled and converted in sequence until a short integer is read from AD_Stop. RD_Start and RD_Stop are used to prevent two pieces of hardware from attempting to access Force_Data simultaneously. The A/D controller hardware writes the converted data into the Force_Data address space. The user's program on one of the CPU's reads the raw data from the same address space. To prevent a conflict, the user must perform a short integer read on RD_Start. This gives the CPU exclusive access to the strain gauge data. The user signals that he is done reading the data by performing a short integer read on RD_Stop, thereby allowing the A/D controller board to write new data into the Force_Data memory space.

Table E.1 A/D Controller Board Memory Map

Name	Address	variable type	read/write
Force_Data	FFFFC000	* short int	read
AD_Stop	FFFFC010	short int	read
AD_Start	FFFFC012	short int	read
RD_Start	FFFFC014	short int	read
RD_Stop	FFFFC016	short int	read

The eight element raw strain gauge pair data in the Force_Data array is converted into force data by premultplying by the 6x8 calibration matrix. The calibration matrix for our sensor as supplied by Assurance Technologies Inc. is,

$$\begin{bmatrix} 0.007969 & -0.506204 & -0.002638 & -0.016756 & 0.001549 & 0.506114 & 0.010923 & -0.011571 \\ 0.002893 & -0.004708 & -0.005842 & -0.521034 & -0.001095 & -0.007040 & 0.010497 & 0.522322 \\ 0.736267 & 0.007211 & 0.726695 & -0.012143 & 0.742474 & 0.001375 & 0.732230 & -0.017379 \\ 0.977210 & -0.002424 & -0.005051 & -0.001627 & -0.984111 & 0.020865 & -0.007911 & -0.017468 \\ -0.001055 & 0.012393 & 1.006657 & -0.020835 & -0.001574 & -0.011087 & -1.010546 & 0.023853 \\ -0.006582 & 0.552776 & 0.005137 & 0.532507 & 0.007445 & 0.522079 & 0.005902 & 0.513380 \end{bmatrix}$$

The resulting force/torque data is converted to units of Newtons and Newton meters by multiplying the force data by 44.48 and the torque data by 1.1298. Further processing can than be done to transform the readings into the desired tool frame or to apply a desired bias.

With these modifications to the force sensor noise is reduced to ± 0.5 N before filtering and provides a data rate, without tool transformation, of 1200 Hz.

Figure E.1 A/D Controller Board Block Diagram

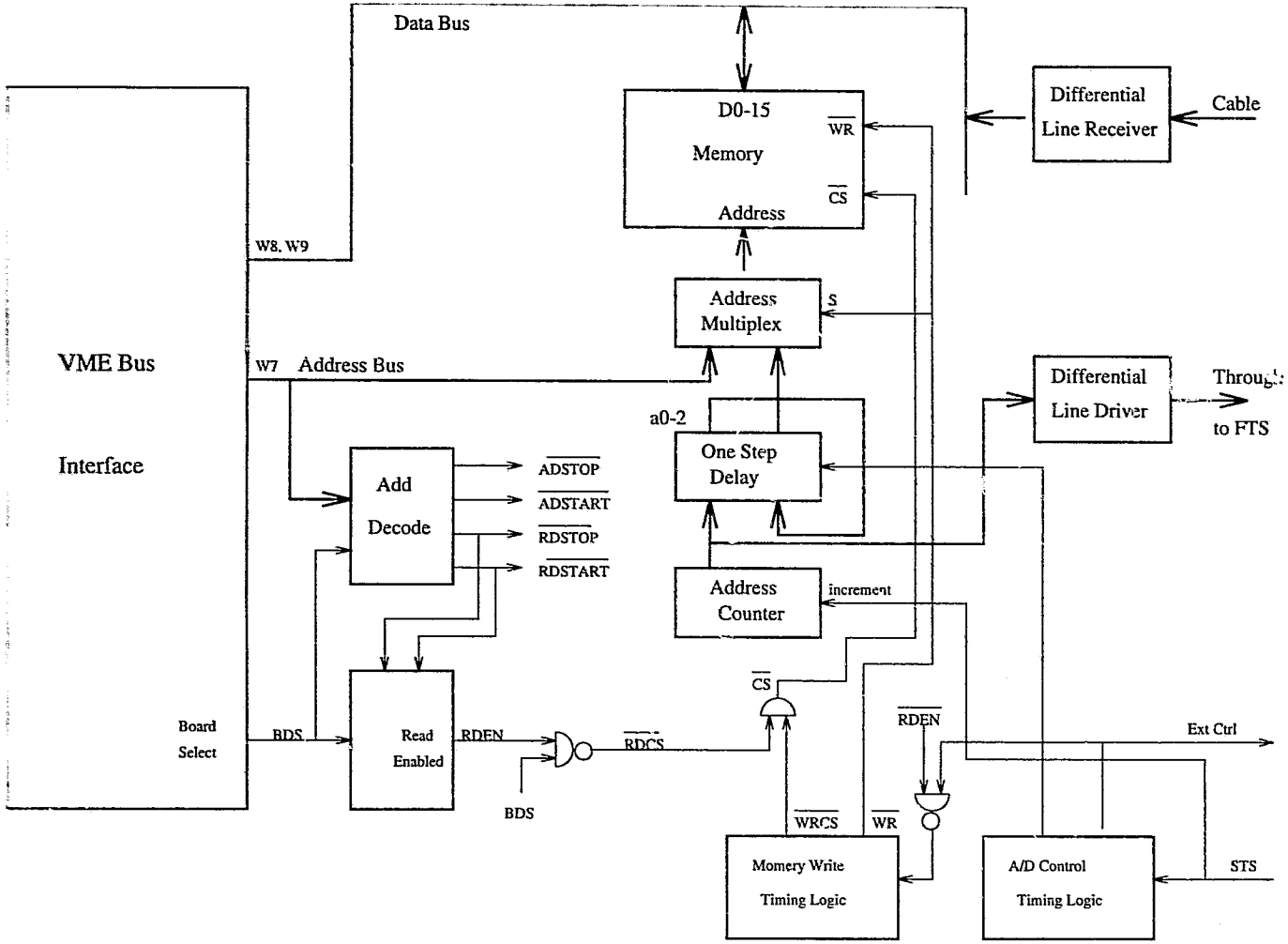


Figure E.2 A/D Converter Block Diagram

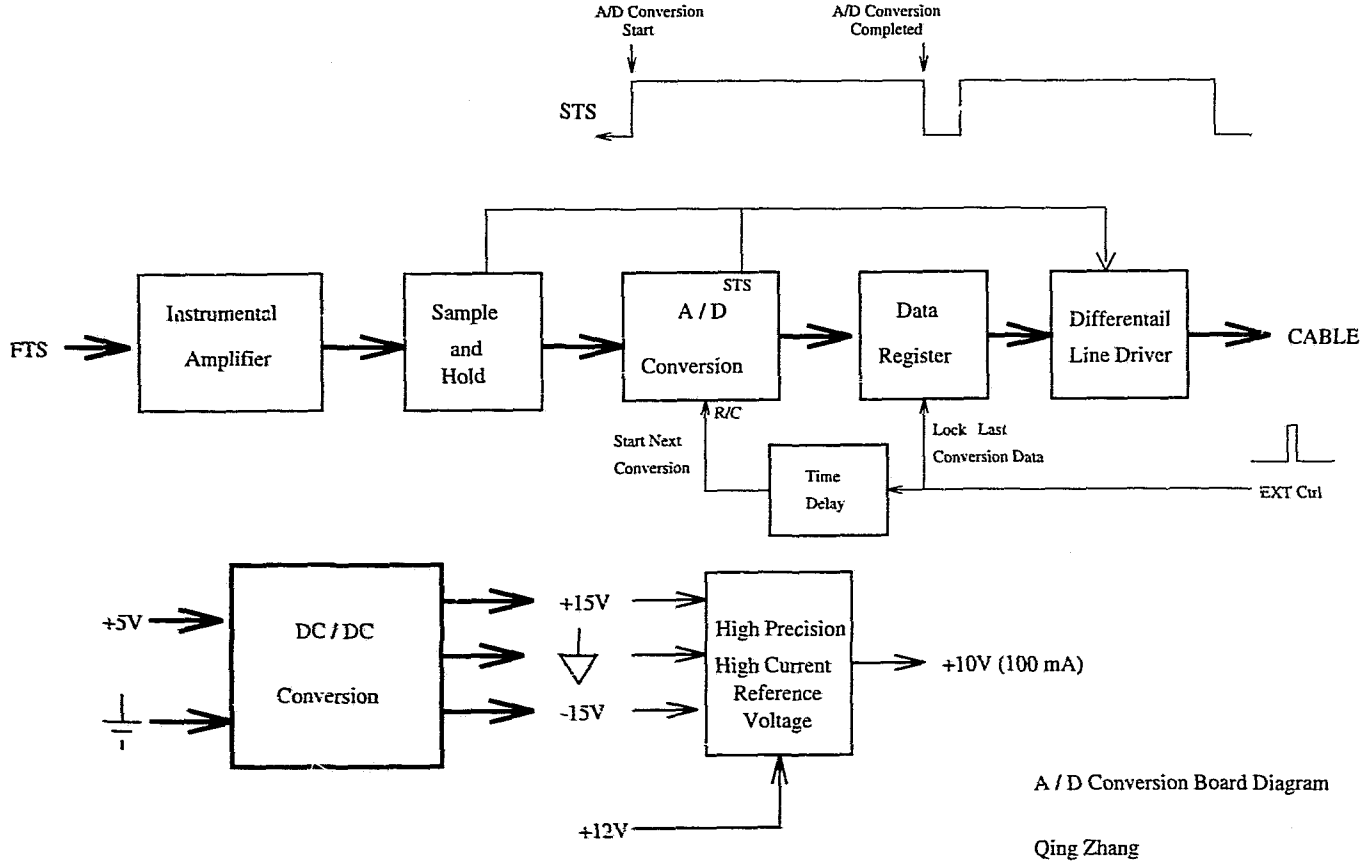


Figure E.3 A/D Controller Board Schematic

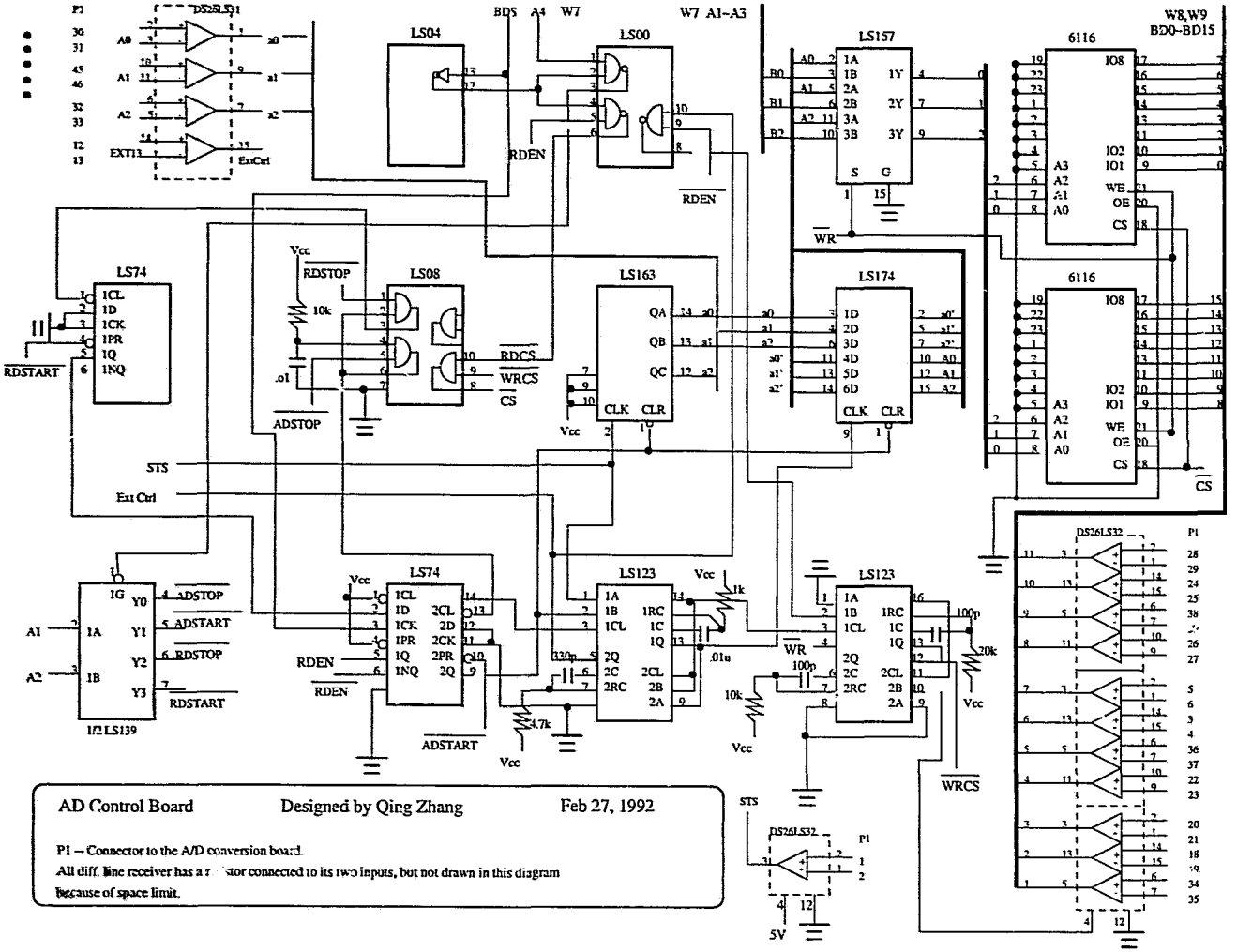


Figure E.4 A/D Converter Schematic

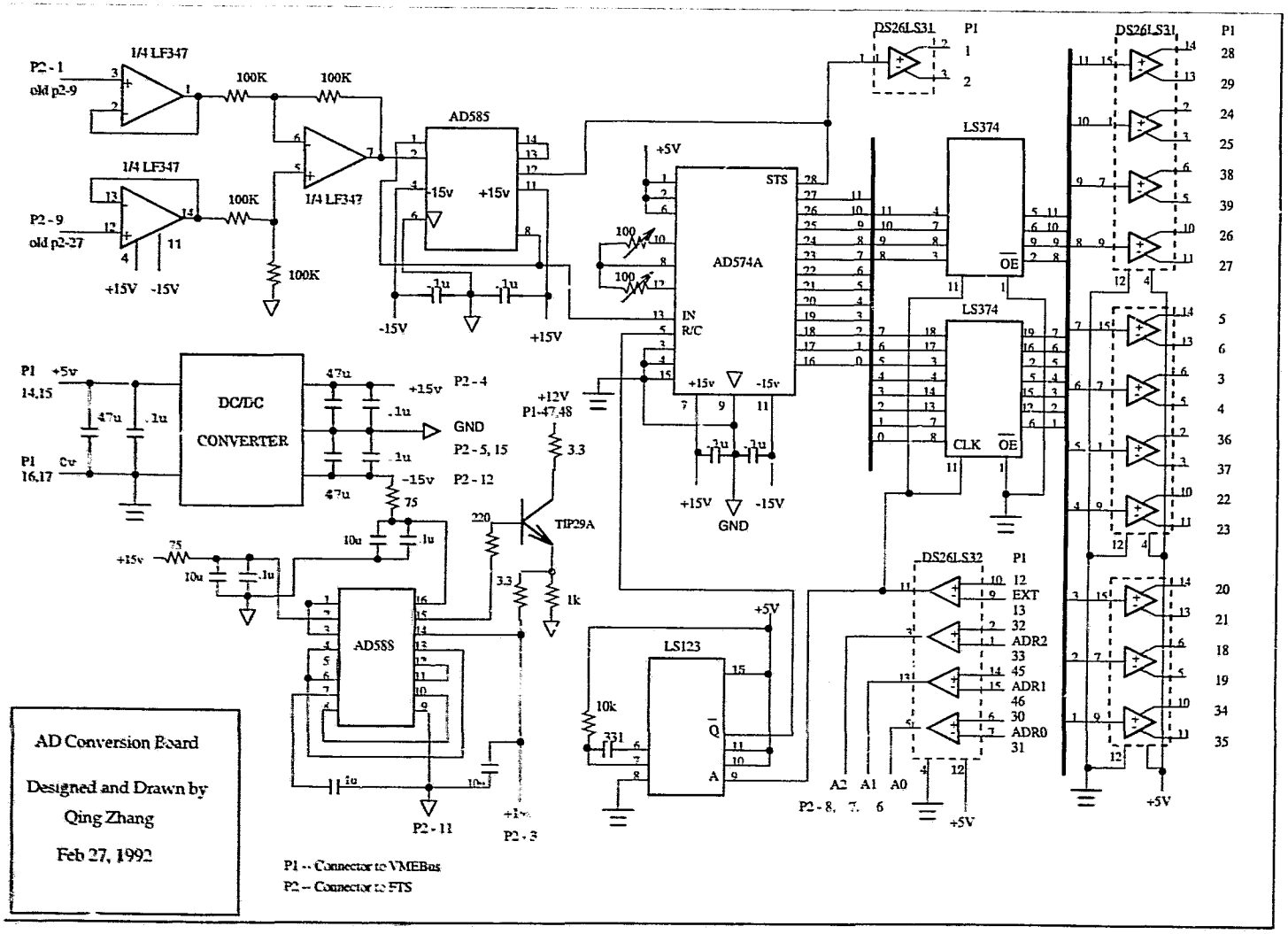


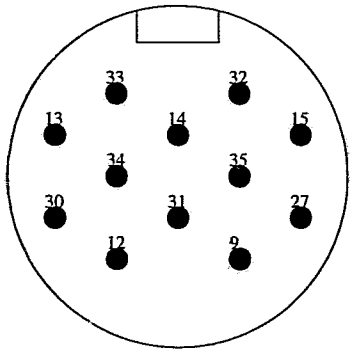
Figure E.5 Cable Pin Outs

Connector to VMEBus (P1)

1	STS+	18	D2+	34	D1+
2	STS-	19	D2-	35	D1-
3	D6+	20	D3+	36	D5+
4	D6-	21	D3-	37	D5-
5	D7+	22	D4+	38	D9+
6	D7-	23	D4-	39	D9-
7		24	D10+	40	
8		25	D10-	41	
9		26	D8+	42	
10		27	D8-	43	
11		28	D11+	44	
12	EXT+	29	D11-	45	A1+
13	EXT-	30	A0+	46	A1-
14	+5V	31	A0-	47	+12V
15	+5V	32	A2+	48	+12V
16	GND	33	A2-	49	unused
17	GND			50	unused

Connector to FTS (P2)

1	fts out-	9	fts out+
2		10	
3	+10V	11	GND
4	+15V	12	-15V
5	GND	13	
6	A0	14	
7	A1	15	GND
8	A2		



Old Connector to FTS

1		19	
2		20	
3		21	
4		22	
5		23	
6		24	
7		25	
8		26	
9	fts out-	27	fts out+
10		28	
11		29	
12	+10V	30	GND
13	+15V	31	-15V
14	GND	32	A0
15	A1	33	A2
16		34	
17		35	
18		36	GND

Appendix F
Model Reference Impedance Control

F.1 Manipulator Interaction With the Environment

To be useful, manipulators must interact with their environment. To do this effectively they must deal with forces due to contact in a controlled and predictable manner. The two main approaches to the force control problem may be broadly classified as hybrid position/force control,[61] and impedance control.[87][67][29][30][31] Hybrid position/force control splits the task space into two orthogonal subspaces. Position is controlled along those directions in which it is impossible to apply an arbitrary force. Force is controlled along those directions in which arbitrary motion is not possible. Impedance control attempts to make the robot end effector act with a specified impedance. Typically, this means forcing the end effector to behave as a multidimensional linear spring-mass-damper system. The goal of impedance control is to provide a controlled compliance in the event of uncertainties in point of contact location and stiffness characteristics of both the environment and robot. No attempt is made to follow a commanded force trajectory, rather, position is commanded and the desired force response is achieved by adjusting the impedance.

The approach currently used to develop an impedance controller requires a dynamic model of the robot and torque controlled actuators. In practice, neither may be available. The controller described here uses a model reference approach, which requires neither a dynamic model, nor torque controlled actuators. While many papers in the literature give simulation results for their force controllers or describe implementations on one or two link direct drive arms,[54][50][40][45][47] there are few reports of practical implementation of impedance control on existing manipulators. This is due mainly to the lack of good dynamic models and unsuitable actuator systems. Both of these problems are overcome with the model reference approach which lends itself to implementation on existing robot designs.

F.2 Impedance Control

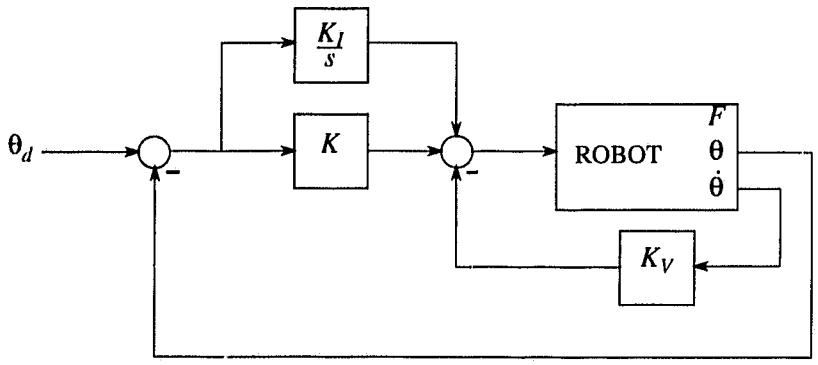
Impedance control was developed as a general approach to the control of manipulation. The method is intended to encompass the traditional positioning tasks performed by robots while including the ability to handle static and dynamic interactions between the manipulator and environment. Although the term "impedance control" did not come into popular usage until after the publication of the three part paper by Hogan in 1985,[29][30][31] earlier research had made steps in this direction. In 1977, Whitney introduced the concept of damping control, which he called "accommodation".[87] In damping control, sensed forces at the end effector give rise to velocity vector which is used to modify a commanded velocity trajectory. The degree of modification is determined by a gain matrix K_D. A second approach introduced in the 1980 is called stiffness control.[67][68] Here instead of interpreting the sensed forces as velocity modifications, they are treated as position modifications in the end effector coordinate system. The apparent stiffness at the end effector is set by the stiffness matrix K_S. Figure F.1 shows a typical control scheme for a robotic manipulator and modifications which can be made to imple-

ment damping and stiffness control. Impedance control expands upon the above two methods by combining them and adding an inertial term. Hogan realized that no physically realizable system could eliminate the inertial effects of the manipulator. By adding this inertial term however, the apparent inertia as seen at the end effector could be modified.

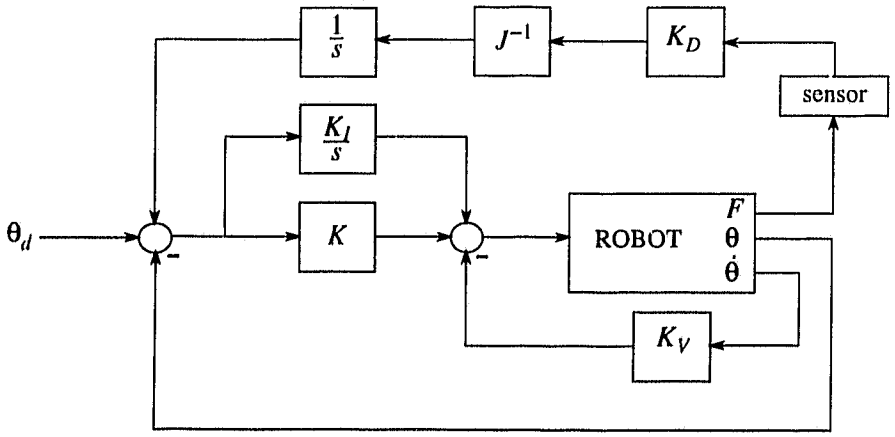
The unmodified control scheme in Figure F.1(a) could already be considered to have implemented a form of impedance control. If the actuators are assumed torque controlled, the gain matrix K controls joint stiffness and the velocity feedback gain K_V controls joint damping. If these matrices are modified based on the robot's joint angles an apparent stiffness and damping in the end effector coordinate system could be generated. The problem with this scheme is that we are relying on the applied load to be fed back through the robot actuators and transmissions. If the actuators are of the direct drive type this assumption may not be unrealistic. However, most robotic transmission systems are not so easily back driven. As a result an applied load at the end effector will not necessarily be reflected by an increased load on the joint actuator. To circumvent this problem a force sensor may be attached to wrist of the manipulator. In the damping control scheme of Figure F.1(b) this force is fed back through the gain matrix K_D and the manipulator's inverse Jacobian J^{-1} to create a vector of joint rate modifications $\dot{\theta}$. This vector is integrated to modify the commanded joint trajectory. In the stiffness control scheme of Figure F.1(c) the gain matrix K_D is replaced by the stiffness matrix K_S. Since the sensed force is treated as a position modifier the integrator in the feedback path can be eliminated. Impedance control is most often implemented by computing a control torque which transforms the existing robot dynamics into a desired form. This technique requires knowledge of the robot model. Consider and n degree of freedom manipulator whose dynamics in Cartesian space are described by

$$\ddot{x} + C_x(x, \dot{x})\,\dot{x} + g_x(x) + f_x(\dot{x}) = J^{-t}\tau \qquad \text{(F.1)}$$

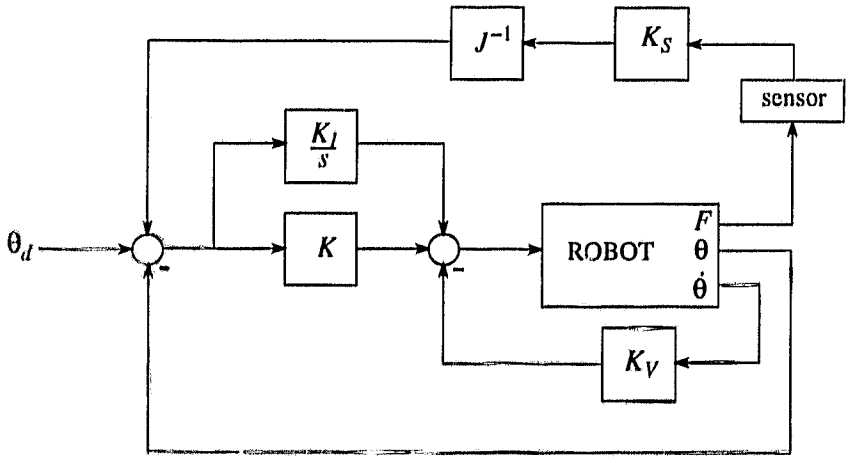
Figure F.1 Damping and Stiffness Control of A Robot



(a) PI Control on Joint Position



(b) Damping Control



(c) Stiffness Control

where τ is the $n \times 1$ vector of joint torques supplied by the actuators; x is the vector which represents the position and orientation of the manipulator's gripper, $H_x(x)$ is the mass matrix and $C(x, \dot{x})$, $g(x)$, and $f(\dot{x})$ represent forces due to centrifugal and coriolis forces, gravity forces and friction forces respectively. In impedance control the desired dynamics at the end effector are often specified by the second order dynamics

$$M(\ddot{x} - \ddot{x}_d) + B(\dot{x} - \dot{x}_d) + K(x - x_d) = -F_{ext} \tag{F.2}$$

where $(x - x_d)$ is the change in Cartesian position from the commanded trajectory and M, B, and K are mass, damping, and stiffness matrices respectively. Given the manipulator dynamics, equation F.1, the appropriate control to achieve the target dynamics, equation F.2, is given by

$$\begin{aligned} \tau = J^T(&F_{ext} + C_x\dot{x} + g_x + f_x + \\ &H_x[\ddot{x}_d - M^{-1}(B(\dot{x} - \dot{x}_d) + K(x - x_d) + F_{ext})]) \end{aligned} \tag{F.3}$$

An alternative implementation of impedance control is shown in Figure F.2. This implementation could be termed a model reference impedance controller since both the force sensed by a wrist mounted force sensor and the commanded input position are used to compute the corresponding motion of a spring-mass-damper model. The existing robot controller is then used to track the model's output. This type of implementation does not require a robot model or even torque driven actuators. All that is required is an existing subcontroller, such as that depicted in Figure F.1(a), which is capable of path tracking.

F.3 Model Reference Impedance Control Implementation

The model reference impedance control scheme depicted in Figure F.2 is implemented on the Reis V15 manipulator using the TEST system described in Chapter 6. The

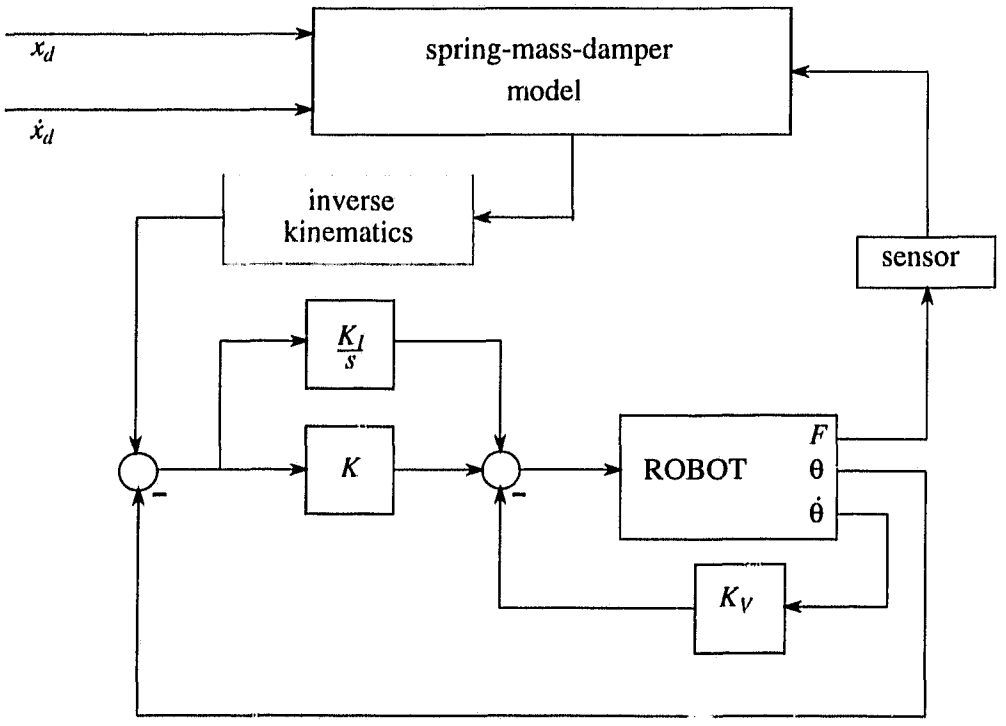


Figure F.2 Model Reference Impedance Control

commanded end effector position and the forces and torques measured by a wrist mounted force sensor are input to a six degree of freedom spring-mass-damper model. The output of the model represents the desired path of the end effector in the Cartesian end effector frame. An inverse displacement routine converts this Cartesian path to target joint angles which are then tracked by the existing robot controller. Considering a single degree of freedom, the model used corresponds to a mass attached to a movable cart by a spring and damper. The commanded input position and velocity are applied to the cart and the sensed contact force is applied to the mass. The net forces and moments applied to the inertia are given by the sum of those due to the spring, damper and applied forces. Euler's equations for rigid body motion may then be applied to compute the model's linear and angular velocities in end effector frame. For the purposes of the present impedance controller the

coupling terms in Euler's equations have been omitted. This is not unreasonable since their effect is small, and we did not want the impedance controller to exhibit gyroscopic characteristics. The velocities are integrated to create differential changes in position and orientation in the end effector frame. The resulting differential transformation is then applied to update the end effector position.

F.4 Experimental Results

To demonstrate the ability of the implemented impedance controller to track specified second order dynamics, a step force input, shown in Figure F.3(a), was applied to the robot end effector by releasing a dead weight suspended from the robot's gripper. For the controller running at 100 Hz., the response of the arm for the model parameters in the vertical 'z' direction; $m = 20$ kg, $b = 20$ kg m/s, $k = 200$ N/m, is shown in Figure F.3(b). For these parameters the model's natural frequency and damping ratio are $\omega_n = 0.5$ Hz. and $\xi = 0.16$ respectively. It is clear from the plot that the robot is tracking the path generated by the impedance model. The deflection at steady state is clearly correct. A load of 25 N. should cause a deflection of 0.125 m. at steady state for a spring stiffness of 200 N/m. From the recorded graph data this deflection is 0.123 m., which is within the tolerance of the applied load. From the figure one can see that the natural frequency matches the model value of 0.5 Hz. and it is apparent that the damping coefficient from the model is having the desired effect. Also note that since the second order model acts as a low pass filter for the force sensor signal the end effector response is very smooth despite the ± 0.5 N of noise on the force sensor readings. When the damping is increased to $b = 130$ kg m/s, the response becomes critically damped as shown in Figure F.3(c). The other input to the model is the end effector position. For the same model parameters a step input of -0.2 m in the 'z' direction generates the responses shown in Figure F.4(a) and

Figure F.4(b). The damping parameter is increased further to provide the overdamped response of Figure F.4(c).

An impedance controller should be able to function both in free space motion and during contact with the environment. The controller parameters required depend on whether or not the robot is in contact with the environment and if contact exists, the characteristics of the contacted surface. The most difficult situation for the controller to handle is the transition from free space motion to contact with a hard surface. In this situation, the controller will tend to bounce on the contact surface, often in an unstable fashion. This bounce effect is shown in Figure F.5(a) where the controller parameters have been set to $m = 20$ kg., $k = 200$ N/m and $b = 500$ kg m/s. There is a large initial force spike when the end effector contacts the stiff surface. Subsequent bouncing is indicated by regions where the contact force becomes zero due to the end effector leaving the table surface. This bouncing characteristic may be damped out by using a larger damping parameter. Figure F.5(b) shows the effect of increasing the damping to $b = 1000$ kg m/s. The end effector still bounces a couple of times but eventually surface contact is maintained. Bounce can be eliminated by increasing the damping further, but this will result in free space motion that in prohibitively sluggish.

A second method of improving the contact force response, without slowing the response in free space motion, is to provide some passive compliance in the end effector. If a one centimeter thick piece of packing foam is applied to the end effector and the task repeated the force response is shown in Figure F.6. Bounce is eliminated and steady surface contact is achieved. This technique works in two ways. First the foam material provides passive damping and absorbs some of the energy of the initial impact. Second, the materials compliance lessens the contact stiffness making the programmed damping more effective. There are problems with the use of passive compliance. To change the character-

Figure F.3 Response to a Step Change in Force

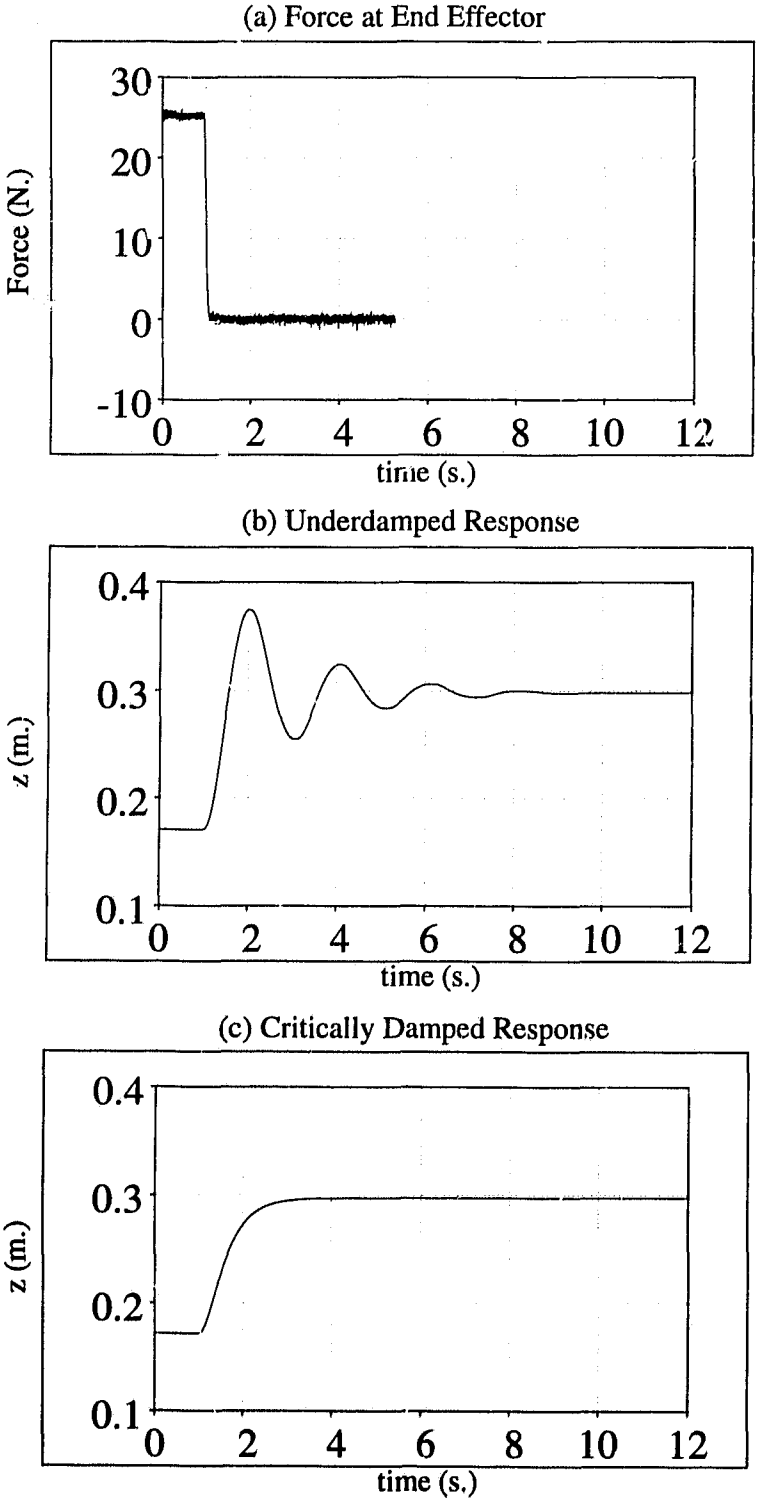
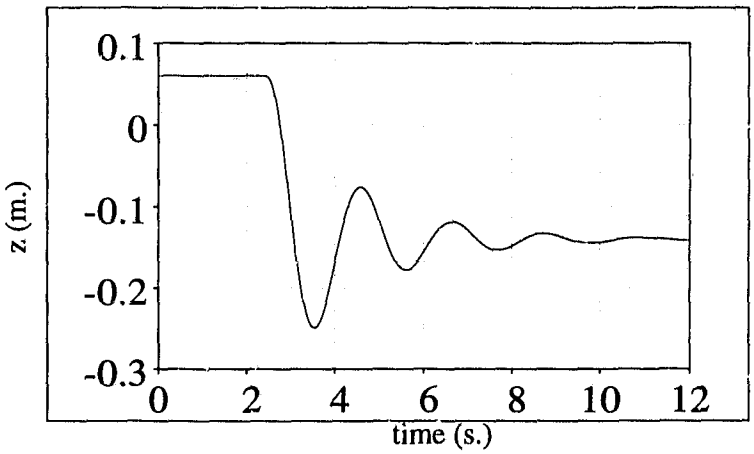
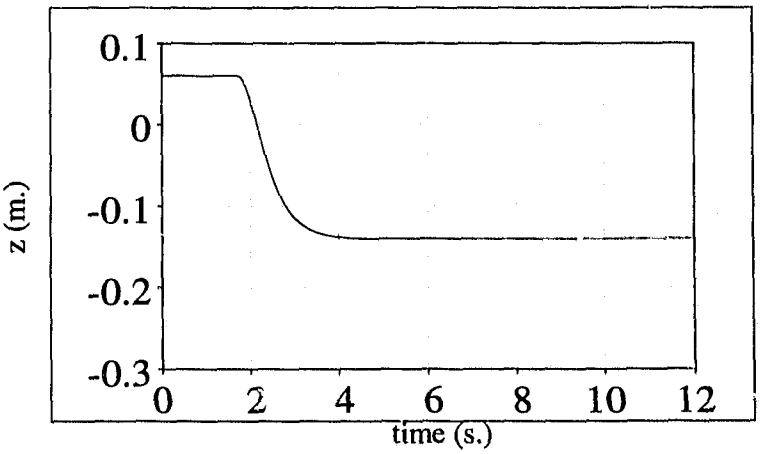


Figure F.4 Response to a Step Change in Position

(a) Underdamped Response



(b) Critically Damped Response



(c) Overdamped Response

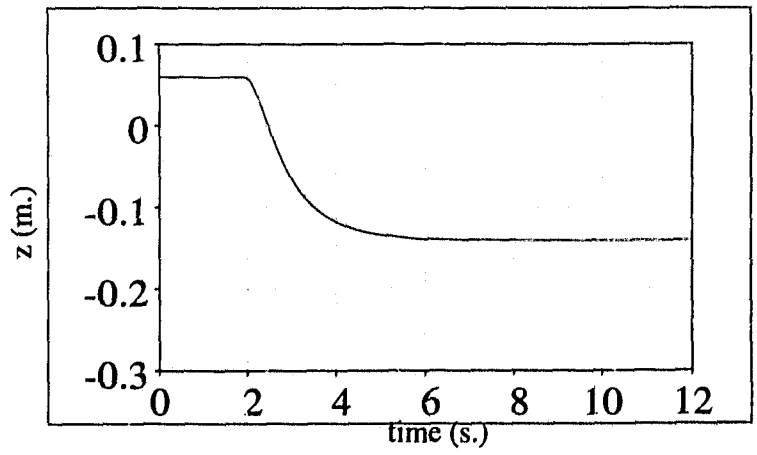
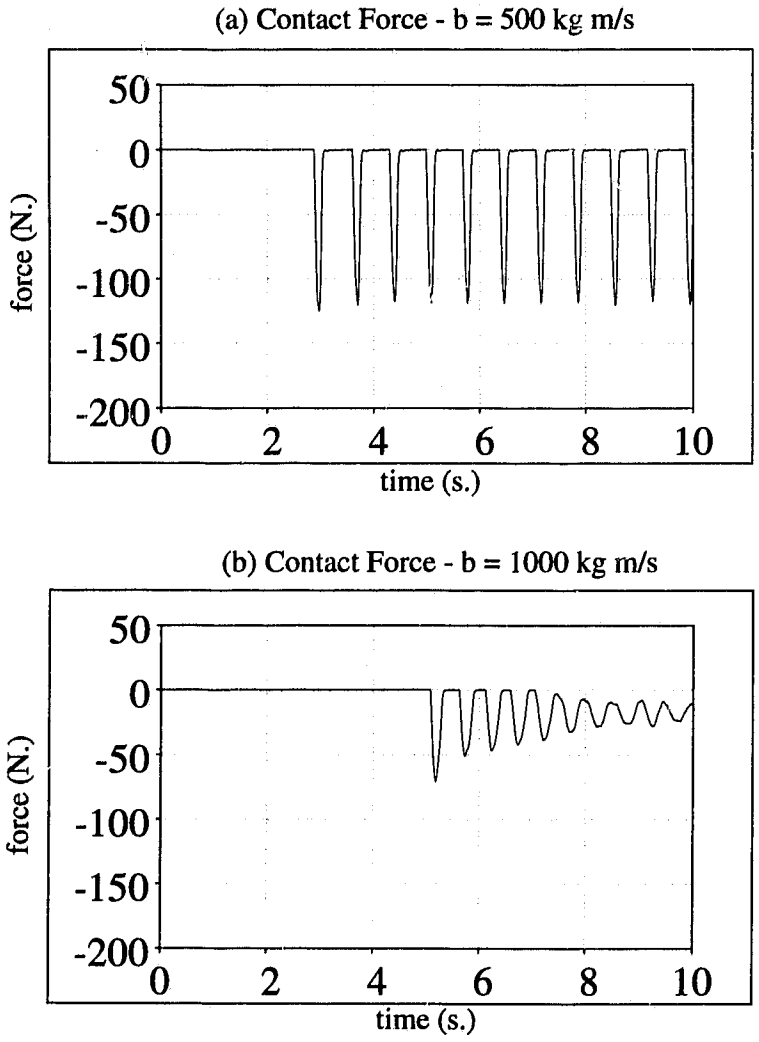


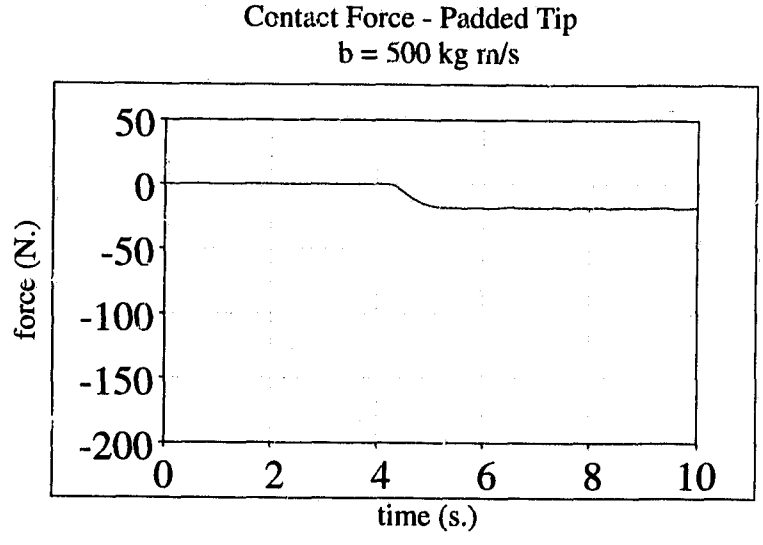
Figure F.5 Impact Response Without Model Adaptation



istics the material must be physically changed. It limits the effective stiffness of the manipulator during contact and therefore the forces which may be applied. If the passive compliance is uninstrumented, precise knowledge of the position of the object in contact with the manipulator is lost.

An alternative to passive compliance is to allow the parameters of the impedance model to change depending on the characteristics of the task at hand. For example, con-

Figure F.6 Impact Response With a Padded Tip



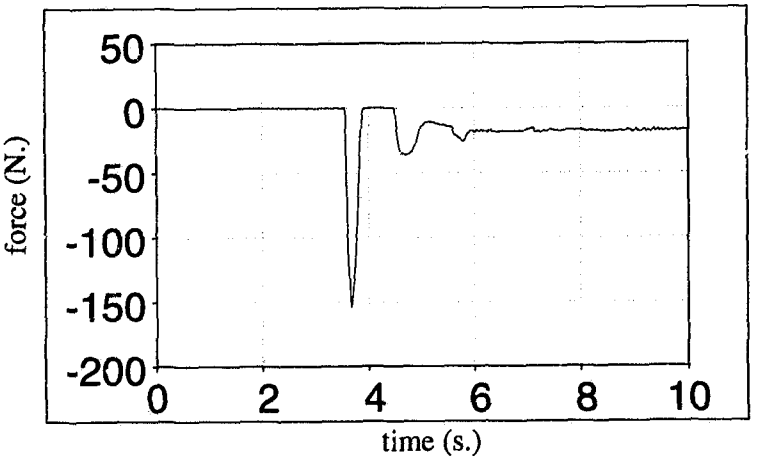
sider Figure F.7(a). Here the damping was increased to 3000 kg m/s when the force sensor reaches a threshold force of 10 N, indicating that contact has occurred. The damping is then allowed to decay linearly with time to a value suitable for maintaining contact. Note that the initial excitation due to the impact is quickly damped out. Since the controller does not adjust its parameters until contact has already occurred there is still a large undesirable initial impact force. If proximity sensors or an approximate world model are available, the parameters may be adjusted before impact. Figure F.7(b) shows that the addition of a world model enables bounce to be eliminated completely and significantly reduces the initial impact force.

F.5 Implementation on the I.S.E. S.P.D.M.

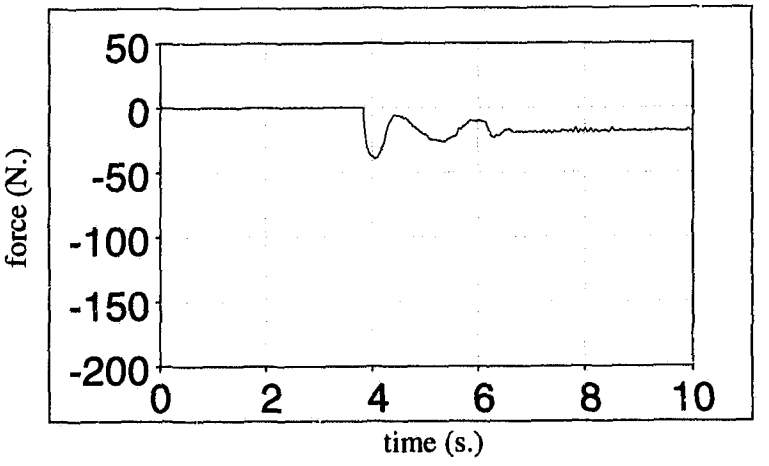
This impedance controller has also been implemented on the Special Purpose Dextrous Manipulator (S.P.D.M.) built by International Submarine Engineering (I.S.E.) of Port Coquitlam, British Columbia. The S.P.D.M. was built as a mock-up of a proposed

Figure F.7 Impact Response With Model Adaptation

(a) Contact Force - Parameter Adaptation with Force Sensor



(b) Contact Force - Parameter Adaptation with World Model



arm design for the remote servicing system for N.A.S.A's proposed space station. The S.P.D.M., shown if Figure F.8, is a seven degree of freedom arm actuated by DC servo motors through harmonic drive gear reducers. The system is controlled by a PC mounted DSP board running I.S.E.'s in house real time operating system. Despite the hardware and operating system difference between I.S.E's system and TEST, the simple structure of the

Figure F.8 International Submarine Engineering - S.P.D.M.

model reference impedance controller approach allowed its implementation on the S.P.D.M. to be straight forward.

F.6 Future Work

As verified by experiment, a model reference impedance controller has successfully been implemented on a six degree of freedom industrial arm. No model of the manipulator dynamics was required; just an underlying controller capable of tracking a specified path in joint space. The bandwidth of the impedance control will be limited by the bandwidth of the manipulator and its underlying position controller. Future work with this impedance controller should include a study of how to best to assign the impedance based on the task at hand. This should involve mechanisms to adaptively modify impedance controller parameters based on sensory or model information. In this paper an intuitive adaptation of only the damping parameter has led to some improvements of the impact

response. Furthers improvements should be achievable by allowing for adaptation of all impedance model parameters. Stability analysis of controllers of this type is an ongoing topic of research.

VITA

Surname: Field Given Names: Glen Arthur

Place of Birth: Pembroke, Ontario, Canada

Educational Institutions Attended:

Malaspina College	1980 to 1981
University of Waterloo	1981 to 1988
University of Victoria	1988 to 1995

Degrees Awarded:

B.A.Sc. (1st Class Honours)	University of Waterloo	1986
M.A.Sc.	University of Waterloo	1988

Honours and Awards:

B.C. Government Scholarship	1980
NSERC PGS1, PGS2 Scholarships	1986-1988
NSERC PGS3, PGS4 Scholarships	1988-1990
University of Victoria Presidents Scholarship	1988, 1989
University of Victoria Teaching Fellowship	1991, 1992, 1993, 1994

Publications:

Field, Glen, Optimal Trajectory Planning for Robotic Manipulators with Minimum Energy Cost Criteria, *M.A.Sc. Thesis*, University of Waterloo, 1988.

Field, Glen, Energetically Optimal Trajectory Planning for Robotic Manipulators, *in Proceedings of the 20th Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA., May, 1989, pp. 1927-1932.

Field, Glen and Yury Stepanenko, Robot Impedance Control Without a Dynamic Model: A Model Reference Approach, *in Proceedings of the IASTD International Conference on Control and Robotics*, Vancouver, Canada, August, 1992, pp. 110-113.

Field, Glen and Yury Stepanenko, A New Model Reference Impedance Controller for Telerobotics, *in Proceedings of the SPIE Telemanipulator Technology Conference*, Boston, Ma., November, 1992, pp. 69-79.

Field, Glen and Yury Stepanenko, Modelling Energy Consumption in Robotic Systems with Bond Graphs, *in Proceedings of the IASTED International Conference on Applied Modelling and Simulation*, Vancouver, Canada, July, 1993, pp. 41-45.

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my dissertation to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on behalf of one of its users. I further agree that permission for extensive copying of this dissertation for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this dissertation for financial gain shall not be allowed without my written permission.

Title of Dissertation:

Minimum Energy Trajectory Planning for Robotic Manipulators

Author: _____

Glen Arthur Field
June 23, 1995