1993

Synergistic Control of N-Body Computer Generated Robots.

Don Alan Iglehart
Louisiana State University and Agricultural & Mechanical College

Order Number 9401539

Synergistic control of N-body computer generated robots

Iglehart, Don Alan, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1993

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

SYNERGISTIC CONTROL OF N-BODY COMPUTER GENERATED ROBOTS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Don Iglehart
B.S., University of Southwestern Louisiana, 1972
M.S. Math, University of Southwestern Louisiana, 1973
May 1993

ACKNOWLEDGMENTS

I wish to express my appreciation the committee members for their forbearance and assistance: S. Sitharama Iyengar, Peter P. Chen, Donald H. Kraft, Doris L. Carver, J. Robert Dorroh, and Ahmed A. El-Amawy.

I thank my wife Susan for her loving kindness and support.

Finally, I thank Wisdom, who dwells with prudence and finds out knowledge of witty inventions [Proverbs 8:12].

TABLE OF CONTENTS

ABSTRACT

This project uses N-Body computer generated robots to investigate the fundamental problem of synergistic control of robots. This facility is important in an automated manufacturing environment and relates directly to the expanding field of study involving nonlinear systems. Experiments are performed utilizing this force controlled model to investigate nonlinear control problems using one of the techniques advanced.

The principal results of this work are summarized:

- A new simulation package is developed which uses Lagrange equations with multipliers to model two dimensional user defined robots. Three dimensionalization and parallelization are discussed and a dynamic optimization algorithm is introduced for regulating the size of the time increment.
- A control package shell is developed which allows the user to construct and test control methodologies against this robot model; it also allows response time delays to be incorporated into the simulation.
- A set of programs are developed as tools for investigating the synergism problem: neural networks, genetic algorithms, and a controller based on Newtonian physics.
- Modified versions of genetic algorithms are investigated and faster convergence rates are obtained.

- A reinterpretation of gradient decent is proffered and methods are found which speed backpropogation convergence in a realtime learning environment.
- A controller is developed, based on Newton's second law, which allows us to determine the "effective mass" and to quantity and compensate for external forces.
- Experiments similar to the broomstick and inverted pendulum are performed using the Newtonian control process.
- An experiment is performed on a human torso robot.

CHAPTER 1

INTRODUCTION

This dissertation uses N-Body computer generated robots to investigate the fundamental problem of synergistic control of robots. In recent years there has been a tremendous increase in interest in the mobile robot learning process. As advanced learning algorithms are developed they will have many applications, their impact on the manufacturing environment will, no doubt, be astounding. Another major application of such learning processes is the prospect of creating agile perambulating robots. These walking robots will be much more mobile than contemporary wheeled robots because much of our architecture has been designed for people who walk, and because most of the land mass of the earth is inaccessible to wheeled vehicles. In addition to the practical implications, the learning process holds an intrinsic fascination as well.

A child spends many months learning to walk and many more perfecting the process - a colt is walking within moments of birth. We attempt to explore the processes which are entailed in learning to manipulate such highly nonlinear systems.

A physical robot with a reasonable learning capability could conceivably be allowed to operate until it has learned

to walk - or until it has pummeled itself into a pile of junk. A simulated robot can be run without such physical consequences; moreover, it can be stopped in any configuration then be restarted (numerous times) from that exact same configuration. Additionally, using a simulated robot allows us to simulate the processing power which may be required (but unavailable) to control a real time robot; the simulated reality can be slowed down to match the processing speed available. Thus a simulated robot is a preferable tool for an initial research project.

Although it would be desirable, the simulated robot need not model reality precisely, as long as it presents the control system with the same type of nonlinearity that would be encountered in the real world. Our intent is to have a reasonably accurate model of reality in a two dimensional, or possibly three dimensional, world. To this end we have developed a two dimensional prototype robot model, written in C, which simulates realistic behavior of robots in response to changes in external forces. We can develop control programs based on such techniques as neural nets, genetic algorithms, genetic programming, conventional methods, etc.; then test these programs in conjunction with this simulator to determine their efficacy.

Typically dynamics simulators are meticulously precise because they are used to electronically prototype mechanisms for engineering purposes, two examples of this are the Newton

project at Purdue University and the DAMS (Dynamic Analysis of Multibody Systems) program of the Mechanical Engineering department at the University of Illinois at Chicago [38]. These are systems developed to model virtually any kind of robot with extreme accuracy over long periods of time. Our purpose does not necessarily require an extremely accurate model over extended time periods - we do require a fast model which is controlled by forces and reasonably credible for the class of mechanisms we will be investigating. If a control program technique, not based on physics, works well in an environment which behaves this realistically then it will work well in the real world.

Two separate sets of programs are required: the first is a model of the real world using Lagrangian dynamics - within small time increments it solves simultaneous differential equations to predict the behavior of a multiple rigid body system which has been defined by the user. The second program set is a control system employing neural nets, genetic algorithms, conventional control methods, or some other method, to control the actuators of the multiple rigid body system previously mentioned.

Communication between the control program and the robot simulation model is as follows: The control program receives configuration information from the model and responds with torque values for the body joints - the model receives these forces and applies them to the body over small time intervals

(which sum to the response time) then returns the new configuration information. If synchronized multitasking is available then the two processes should be run concurrently. If it is not available then concurrence should be simulated, this can be done with a response time delay if desired. The simulation model is a foreground process, providing a real time graphical display of the body (which can also be recorded and replayed).

The principal results of this work are summarized:

- A new simulation package is developed which uses Lagrange equations with multipliers to model two dimensional user defined robots. Three dimensionalization and parallelization are discussed and a dynamic optimization algorithm is introduced for regulating the size of the time increment.
- A control package shell is developed which allows the user to construct and test control methodologies against this robot model; it also allows response time delays to be incorporated into the simulation.
- A set of programs are developed as tools for investigating the synergism problem: neural networks, genetic algorithms, and a controller based on Newtonian physics.
- Modified versions of genetic algorithms are investigated and faster convergence rates are obtained.
- A reinterpretation of gradient decent is proffered and methods are found which speed backpropogation convergence in a realtime learning environment.

- A controller is developed, based on Newton's second law, which allows us to determine the "effective mass" and to quantity and compensate for external forces.
- Experiments similar to the broomstick and inverted pendulum are performed using the Newtonian control process.
- An experiment is performed on a human torso robot.

In this dissertation we will first describe some of the work others are currently doing in related research, then we will describe the robot model and a technique employing Lagrange equations with multipliers. A discussion of the control model interface, genetic algorithms, neural nets, and a Newtonian control procedure will follow. A particular robot will be investigated, a simple human torso, observations will be made and an experiment described. Finally, conclusions will be presented.

CHAPTER 2

BACKGROUND

A mechanism or robotic system may be considered to be comprised of interconnected ridged (and even deformable) bodies each of which may undergo large translational and rotational displacements. The dynamic equations which govern the motion of such systems are highly nonlinear.

Our intent is to model robots composed of rigid bodies with revolute joints. This class of robots will allow us to model a dog, horse, roach, spider, man, arm, hand, etc. The robot modeling approach we have taken is based primarily on information and techniques described in the book Dynamics of Multibody Systems [38].

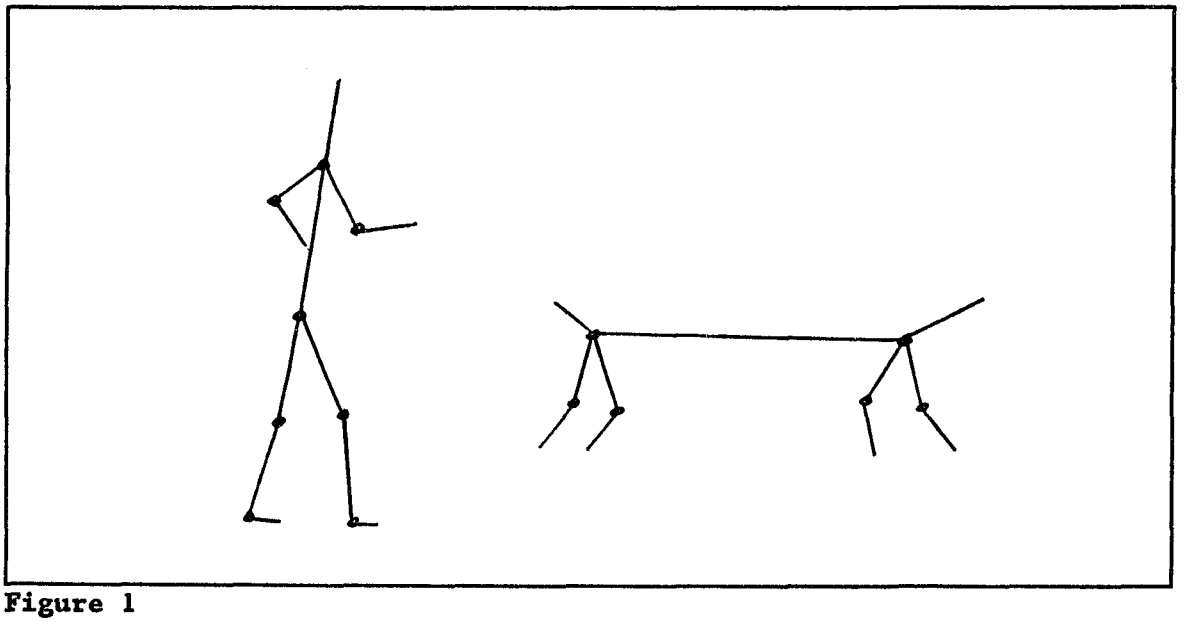

Figure 1

The robot model is a tool needed in order to carry out the research on the controller which is a more state-of-the-art subject. However, work is being published in regard to such modeling: an interactive robot simulation package [33] for the MacIntosh microcomputer was recently introduced. A Lagrangian dynamics model was used to produce realistic animation of a human walking [39]; the controller portion of the project might be of interest to animators as seen by the use of controller algorithms in a paper discussing the animation of dynamic legged locomotion [36], although animators would not require the realism of a response time delay and would have no compunction about using the model itself to determine forces. Likewise, engineers would use a model, such as our robot model, as a controller for an actual robot.

Numerous articles regarding control programs can be found in current journals. Most of these control systems are variations of either classical control theory or modern control theory. There are many books on these subjects (see for example [10,12,30,34]). The classical theory, called "frequency-domain method", uses transfer functions, from Laplace transforms, to represent processes; the interconnection of transfer functions in a block diagram permit visualization of the interaction of various subsystems within a more complex system. Modern theory, called "state-space method", characterizes system processes by coupled

differential equations - this is precisely how we modeled our robot as described in the first part of this project. However, as remarkable as a Louisiana roach may be, intuition tells us that it does not solve simultaneous differential equations to determine how it should move its legs in order to walk!

Increasingly, researchers have been addressing control by various renditions of neural nets [1,4,5,7,8,16,22,24], generally these attempt to model physical systems with very few links and use fairly standard back propagation techniques [40] attempting to attain an adaptive control capability. The problem of slow convergence of neural nets using back propagation has been addressed by using Radial Basis expansion [32,35], the cerebellar model articulation controller [2,3,27], variable learning and momentum coefficients [21], or dynamic insertion of neurons [6].

A novel approach [11] based on Sparse Distributed Memory [23] uses an associative memory instead of a neural network to learn to correlate input patterns with their corresponding output patterns.

By far the most impressive results were obtained though the use of "genetic programming" [9]. Here a pair of stick legs were self taught to walk in a lifelike manner in a two dimensional velocity controlled kinematic model. Most research in synergism is done on kinematic models because they are much easier to construct and run much faster, but

they do not provide the same level of nonlinearity found in reality.

Research in this area is interdisciplinary, for example, the Department of Psychology at Princeton [28] is doing similar work on a planar six link redundant manipulator. Their work differs in that their model is physical and uses antagonistic actuators, also they integrate knowledge based systems with the neural net control.

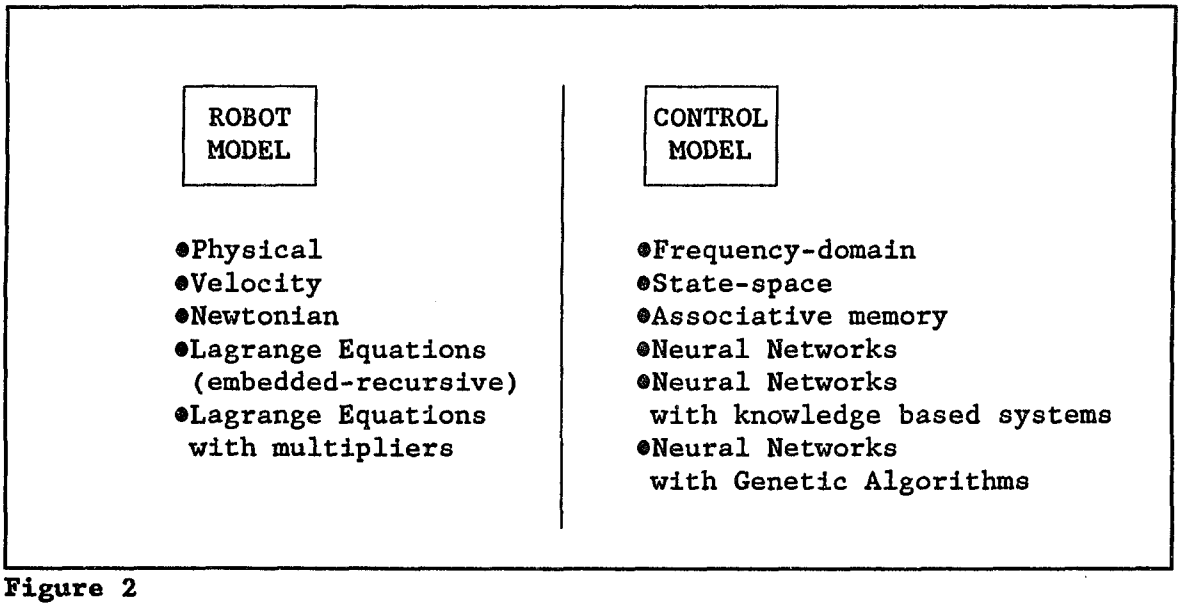

Figure 2

Current research with similarities generally has any of three purposes: graphics, engineering, and the study of synergism. In the study of synergism there are several models being used: physical robots, velocity controlled models, older force controlled techniques (such as embedding) and Lagrange equations with multipliers. The control methods

used employ: frequency-domain, state-space, knowledge-based, neural nets, associative memory and genetic algorithms.

CHAPTER 3

ROBOT MODEL

The primary motivation of this dissertation was to develop a robot model which will be available for continuing research into robot learning. The availability of robots is certainly lower than that of computers, so a sophisticated computer model is very desirable. We desire to deal with robots controlled by forces (rather than by velocities) so that our control methodologies will be tested against the same nonlinearities found in reality.

The description of this robot model begins with the three procedures of which the model is composed, followed by a description of the primary data structure. An overview of the methodology is followed by a derivation of the matrix of Lagrange equations with multipliers - this technique is the foundation of the model. Specific issues are addressed in some detail, a synopsis is provided to bring the process into clear focus.

Procedures

CREATE is a program which allows the user to graphically create a new robot or modify an existing one. New elements (links) may be added to either end of any existing element and joint limit constraints may be established. Each element

has an initial orientation, a mass and a length; either end of the link may be flagged so that this point will not be allowed beneath the ground line (y = 0). Note that each element will have at most one parent and one actuator associated with the joint to that parent. Numerous single key commands are provided which allow the user to build and modify the robot and to control the graphics.

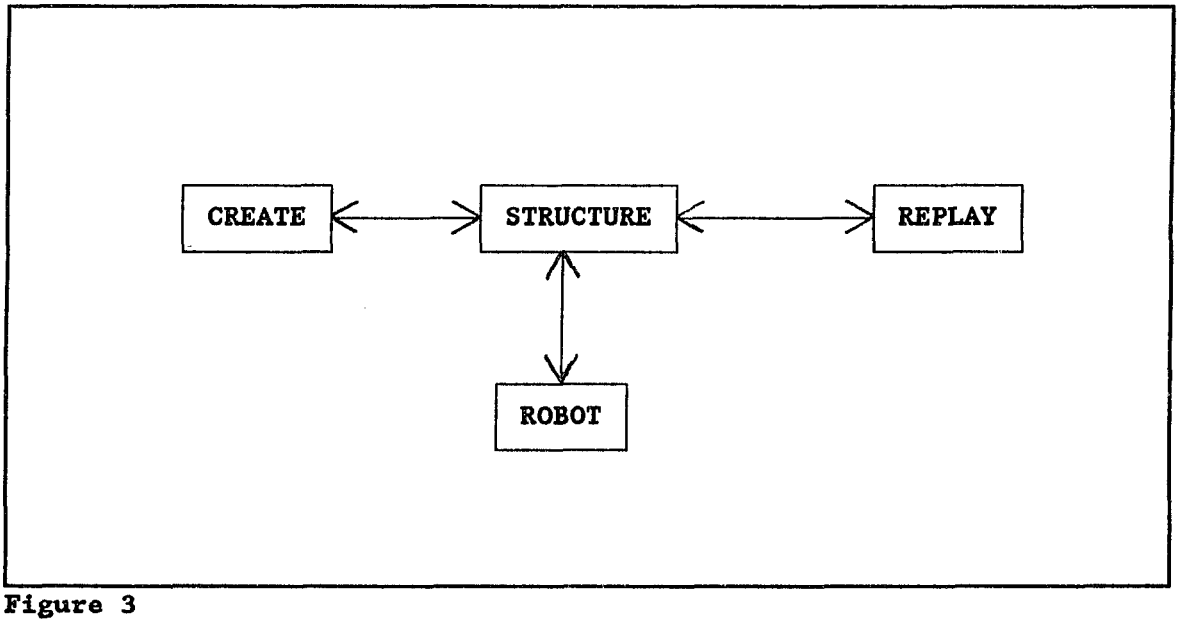

Figure 3

ROBOT is a program, it loads in a robot structure file - from the CREATE program or down loaded from ROBOT itself. It analyzes gravity, constraints, and any externally supplied torques (from a control program or direct user intervention) to produce appropriate changes in the configuration over time. Numerous single key commands are provided which allow the user to affect the process and to control the graphical output.

REPLAY is a graphical output program which can replay data produced and recorded by the ROBOT program; several single key commands are provided which allow the user to control the speed and the graphics.

Data Structure

The term n-body refers to two or more links (ridged bodies or components - we shall call these "elements") which are joined together to form a system, in our case a robot.

The data structure is given in table I.

Table I

```
struct {
   int parent,              parent number
     jointo,               which end of the parent
     groundtyp,            if ground limit, which end
     groundnum,            temporary membrane number
     torque,               externally supplied torque
     childtorques;         sum of children's torques
   double mass,             element mass
     halflength,           half of the element length
     childlimtorques,      joint limit constraints
     jointlimit1,          joint freedom limitation
     jointlimit2;          joint freedom limitation
    struct {
     double R1,             horizontal position
            R2,             vertical position
            Theta,          angle
            R1v,            horizontal velocity
            R2v,            vertical velocity
            Thetav;         angular velocity
    } n,o;                   at new and old time
}  Elem[El] ;
```

We decided to maintain the elements of the robot in a tree structure, this structure allows us immediate access to

parent information for joint analysis. It can also allow us to process the parents before the children or vice versa - in this regard their order in the array is sufficient.

It simplifies our task for each element to have only one parent and one actuator - which is associated with that joint.

Methodology

For the initial point in time, we create and solve a matrix of simultaneous differential and algebraic equations to determine the instantaneous accelerations. We integrate forward in time using simple approximating formulas for the new positions and velocities in terms of these accelerations. For the next time period we again create (using the new positions and velocities) and solve a similar matrix to determine the instantaneous accelerations. We average these accelerations with the initial accelerations and reintegrate; we repeat this procedure until the positions do not change substantially, then we use these accelerations as initial accelerations and repeat the process for the next time interval. The process is described in more detail below and followed by a summary of the discussion.

Matrix of Lagrange Equations

Consider three elements joined together as depicted in figure 4. We will begin to develop the matrix equations which will be the centerpiece of our robot model program; a matrix of simultaneous differential and algebraic equations which will define our instantaneous accelerations in terms of all other system variables.

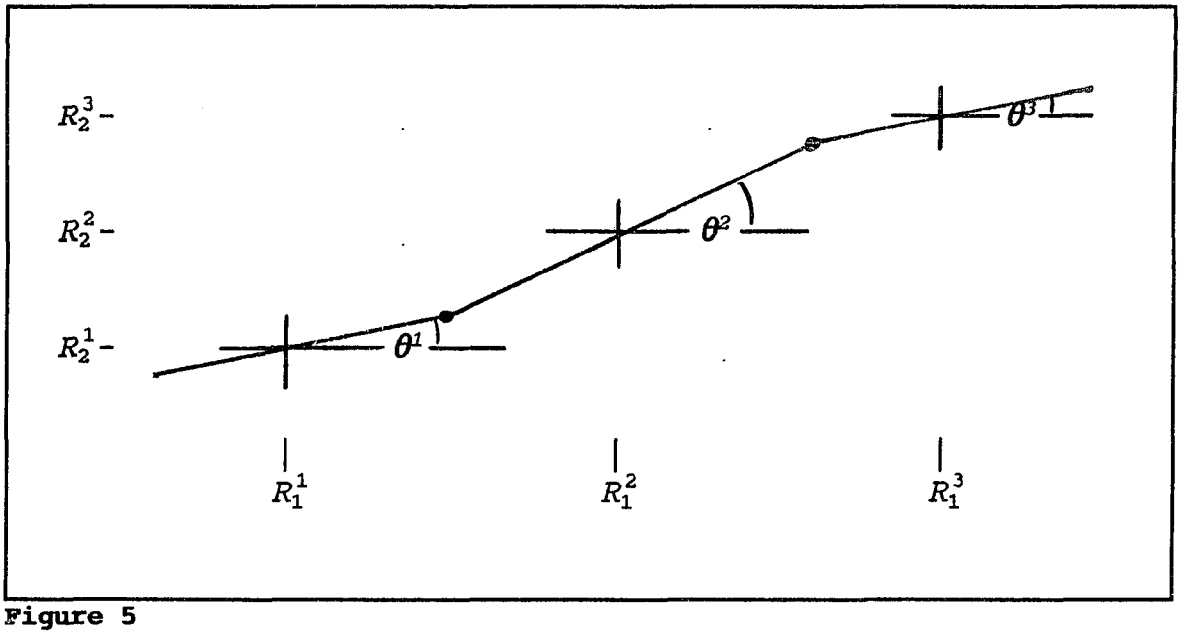

Figure 5

The positions may be expressed as generalized coordinates:

$$q = \begin{bmatrix} R_1^1 \\ R_2^1 \\ \theta^1 \\ R_1^2 \\ R_2^2 \\ \theta^2 \\ R_1^3 \\ R_2^3 \\ \theta^3 \end{bmatrix} = \begin{bmatrix} R_1^2 \\ R_2^1 \\ \theta^1 \\ R_1^1 + \frac{1}{2}\ell_1\cos\theta^1 + \frac{1}{2}\ell_2\cos\theta^2 \\ R_2^1 + \frac{1}{2}\ell_1\sin\theta^1 + \frac{1}{2}\ell_2\sin\theta^2 \\ \theta^2 \\ R_1^2 + \frac{1}{2}\ell_2\cos\theta^2 + \frac{1}{2}\ell_3\cos\theta^3 \\ R_2^2 + \frac{1}{2}\ell_2\sin\theta^2 + \frac{1}{2}\ell_3\sin\theta^3 \\ \theta^3 \end{bmatrix}$$

The velocity is the first derivative of the generalized coordinates:

$$\dot{q} = \begin{bmatrix} \dot{R}_1^1 \\ \dot{R}_2^1 \\ \theta^1 \\ \dot{R}_1^2 \\ \dot{R}_2^2 \\ \theta^2 \\ \dot{R}_1^3 \\ \dot{R}_2^3 \\ \theta^3 \end{bmatrix} = \begin{bmatrix} \dot{R}_1^2 \\ \dot{R}_2^1 \\ \theta^1 \\ \dot{R}_1^1 + \frac{1}{2}\ell_1\theta^1\cos\theta^1 + \frac{1}{2}\ell_2\theta^2\cos\theta^2 \\ \dot{R}_2^1 - \frac{1}{2}\ell_1\theta^1\sin\theta^1 - \frac{1}{2}\ell_2\theta^2\sin\theta^2 \\ \theta^2 \\ \dot{R}_1^2 + \frac{1}{2}\ell_2\theta^2\cos\theta^2 + \frac{1}{2}\ell_3\theta^3\cos\theta^3 \\ \dot{R}_2^2 - \frac{1}{2}\ell_2\theta^2\sin\theta^2 - \frac{1}{2}\ell_3\theta^3\sin\theta^3 \\ \theta^3 \end{bmatrix}$$

The system constraints are:

$$R_1^1 + \tfrac{1}{2}\ell_1\cos\theta^1 = R_1^2 - \tfrac{1}{2}\ell_2\cos\theta^2$$
$$R_2^1 + \tfrac{1}{2}\ell_1\sin\theta^1 = R_2^2 - \tfrac{1}{2}\ell_2\sin\theta^2$$

$$R_1^2 + \tfrac{1}{2}\ell_2\cos\theta^2 = R_1^3 - \tfrac{1}{2}\ell_3\cos\theta^3$$
$$R_2^2 + \tfrac{1}{2}\ell_2\sin\theta^2 = R_2^3 - \tfrac{1}{2}\ell_3\sin\theta^3$$

which we express as the vector of constraint functions:

$$C = \begin{bmatrix} R_1^1 + \frac{1}{2}\ell_1\cos\theta^1 - R_1^2 + \frac{1}{2}\ell_2\cos\theta^2 \\ R_2^1 + \frac{1}{2}\ell_1\sin\theta^1 - R_2^2 + \frac{1}{2}\ell_2\sin\theta^2 \\ R_1^2 + \frac{1}{2}\ell_2\cos\theta^2 - R_1^3 + \frac{1}{2}\ell_3\cos\theta^3 \\ R_2^2 + \frac{1}{2}\ell_2\sin\theta^2 - R_2^3 + \frac{1}{2}\ell_3\sin\theta^3 \end{bmatrix} = \mathbf{0}$$

In the Constraint Jacobian matrix each column has been differentiated with respect to one of the variables.

$$C_q = \begin{bmatrix} 1 & 0 & -\frac{1}{2}\ell_1\sin\theta^1 & -1 & 0 & -\frac{1}{2}\ell_2\sin\theta^2 & 0 & 0 & 0 \\ 0 & 1 & +\frac{1}{2}\ell_1\cos\theta^1 & 0 & -1 & +\frac{1}{2}\ell_2\cos\theta^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\frac{1}{2}\ell_2\sin\theta^2 & -1 & 0 & -\frac{1}{2}\ell_3\sin\theta^3 \\ 0 & 0 & 0 & 0 & 1 & +\frac{1}{2}\ell_2\cos\theta^2 & 0 & -1 & +\frac{1}{2}\ell_3\cos\theta^3 \end{bmatrix}$$

If δq is the virtual displacement then

$$C_q \, \delta q = 0$$

It follows trivially that

$$\lambda^T C_q \delta q = 0$$

The λ are known as multipliers, we will return to them later.

Now we consider D'Alembert-Lagrange's equation using vector notation; q is the generalized system coordinates so δq is the virtual displacement:

$$\left[\frac{d}{dt} (T_{\dot{q}}) - T_q - Q^T \right] \delta q = \Lambda \, \delta q = 0$$

T is the kinetic energy and Q is the vector of external forces:

$$Q_{R_1^i} = 0$$

$$Q_{R_2^i} = - m_i g$$

$$Q_{\theta^i} = torque_i - \sum_{children} torque_j$$

We may also adjust these forces with spring and damper forces:

$$c_s(q - q_0) + c_d \dot{q}$$

c_s is, of course, Hook's constant.

Consider a single coordinate q:

$$\Lambda = \frac{d}{dt}(T_{\dot{q}}) - T_q - Q$$

$$\Lambda = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{q}}(\tfrac{1}{2}m\dot{q}^2)\right) - \frac{\partial}{\partial q}(\tfrac{1}{2}M\dot{q}^2) - Q$$

$$\Lambda = \frac{d}{dt}(m\dot{q}) - m\dot{q}\frac{\partial \dot{q}}{\partial q} - Q$$

$$\Lambda = m\ddot{q} - \dot{m}\dot{q} - m\dot{q}\frac{\partial \dot{q}}{\partial q} - Q$$

The quadratic velocity vector becomes zero when the origin of the body is at the center of mass, therefore:

$$\Lambda = m\ddot{q} - Q$$

We now return to the D'Alembert-Lagrange's equation using **M** as the mass matrix.

$$\left[\frac{d}{dt}(\boldsymbol{T}_{\dot{q}}) - \boldsymbol{T}_q - \boldsymbol{Q}^T\right]\delta \boldsymbol{q} = [\boldsymbol{M}\ddot{\boldsymbol{q}} - \boldsymbol{Q}^T]\,\delta \boldsymbol{q} = \boldsymbol{0}$$

Recalling that

$$\boldsymbol{\lambda}^T \boldsymbol{C}_q \delta \boldsymbol{q} = \boldsymbol{0}$$

We obtain

$$\delta \boldsymbol{q}^T\left[\boldsymbol{M}\ddot{\boldsymbol{q}} - \boldsymbol{Q} + \boldsymbol{C}_q^T\boldsymbol{\lambda}\right] = \boldsymbol{0}$$

From this equation we can obtain a matrix of Lagrange equations with multipliers - a matrix of simultaneous differential and algebraic equations which will define our instantaneous accelerations in terms of all other system variables.

$$\begin{bmatrix} M & C_q^T \\ C_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q \\ Q_c \end{bmatrix}$$

The Q_c can be obtained from

$$Q_c = C_q \ddot{q} = D_t^2 C$$
$$= - C_{tt} - (C_q \dot{q})_q \dot{q} - 2 C_{qt} \dot{q}$$

Which, for our purposes, reduces to

$$Q_c = - (C_q \dot{q})_q \dot{q}$$

The matrix equation above can be solved for the instantaneous accelerations, $\ddot{q}$, corresponding to the masses in M, the positions in C_q, the velocities in Q_c, and the forces in Q, which taken together define the state space at a given time. The values of the multipliers, λ, are of no interest to us, the multipliers are simply the glue used to hold the simultaneous differential and algebraic equations together.

Integration of Acceleration

The matrix equation obtained in the previous section is composed of: an unknown vector of accelerations and multipliers; a known vector of external forces (torques, gravity, springs & dampers), constraints, positions (angles) and velocities; and a matrix of masses and mass moments of inertia, constraints and positions (angles). We rewrite this as a vector function of acceleration in terms of position, velocity and force:

$$\boldsymbol{a} = \boldsymbol{F}(\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{f})$$

Forget position, for a moment, and consider figure 5 which depicts acceleration as some function of velocity at two different times (the subscripts represent time).

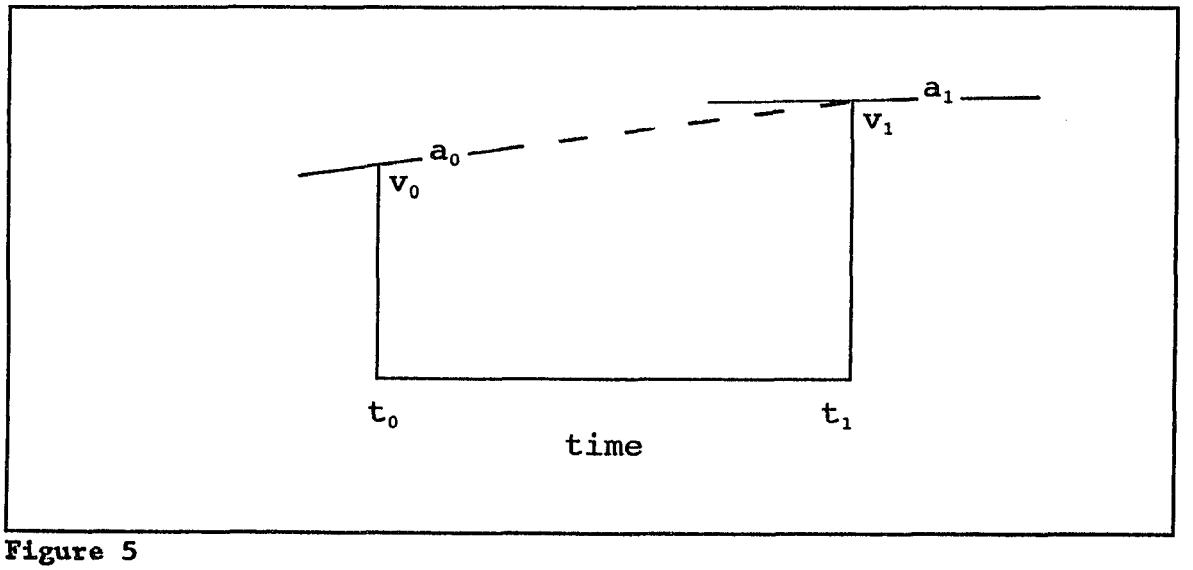

Figure 5

If we know v_0 and the function then we can calculate a_0 and use the two simultaneous equations,

$$a_1 = f(v_1)$$
$$v_1 = v_0 + \frac{(a_0 + a_1)}{2} t$$

to calculate v_1 and a_1. If the function is highly nonlinear, an iterative process must be used - this is a projection to v_1 using the v_0 and a_0 and a correction to the acceleration using a_1 followed by another projection, etc. From a mathematical perspective the time interval is being modeled with a quadratic equation using the beginning value, its slope and the slope of the ending value. The technique is known as a projection correction method - the projection here being the Euler Method - this specific version is called an iterative starting function. Engineering systems requiring great accuracy would use a multistep iterative process and would reproject, or narrow the time intervals, until the correction is within some delta of the projection; this delta is determined by the multistep function used. For our prototype we seek only a lifelike response and do not require great accuracy, and we may be changing forces and constraints so frequently as to diminish the effectiveness of a multistep process.

Now remembering and including the position, as well as the velocity, we obtain the following three simultaneous equations:

$$p_1 = p_0 + v_0 t + \frac{(a_0 + a_1)\, t^2}{4}$$

$$v_1 = - v_0 + \frac{2\,(p_1 - p_0)}{t}$$

$$a_1 = F(p_1, v_1, f)$$

Time Interval Optimization

Matrix evaluations are very expensive, especially on our SIMD implementation, so we will want to minimize the number of matrix evaluations. In the previous section we developed the procedure whereby, knowing $\{p_0,v_0,a_0\}$ we easily project $\{p_1,v_1\}$, do a matrix evaluation to determine a_1, and finally use $\{p_0,v_0,a_0,a_1\}$ to reproject $\{p_1,v_1\}$. If the second projection is sufficiently close to the first one then we have been successful and have only entailed one matrix evaluation. Otherwise we have two options: we can continue to reproject (requiring a matrix evaluation for each of these) or we can cut our losses and start over with a smaller time interval. Empirically, it is found that the conversion rate for reprojecting is rather slow - halving the time interval is a better bet (generally requiring 3, versus 5, evaluations for the original interval).

The problem at this point is to know what time increment to use in the first place - it should be large enough to require few ordinary evaluations, but small enough to avoid many halvings. The problem is exacerbated by the fact that

this value will change with circumstances, so using any constant would be a poor approach. Whatever the value, it should be continually reevaluated and tested in areas which are probably not optimal. For purposes of discussion we will refer to this default time interval as Δt, the largest allowed value for default time will be called the "maximum" time. The program will attempt to use the default value but will halve it as necessary to accomplish its ends. The default will initially be set to the maximum, the default will be recalculated at every passage of a time interval equal to maximum. To describing this simple algorithm we define "evaluations" as the number of matrix evaluations incurred during the maximum time period:

If evaluations > 2 * maximum / Δt then $\Delta t = \Delta t / 2$;

If evaluations < maximum / Δt + ϵ then $\Delta t = 2 * \Delta t$.

The logic being, if there are too many halvings then the default is too large - if half of the intervals were halved then evaluations would equal twice maximum/Δt and also equal to what evaluations would (probably) have been if Δt had been half its value, this is the break even point. If there are too few halvings then the default is too small - no halvings is certainly too few. Also, Δt is prevented from exceeding maximum.

Suppose we started with some interval and halved several levels down and achieved a successful evaluation of a subinterval, how should we handle the rest of the interval?

We could handle the remainder of the interval as a whole, or break it into intervals the size of the subinterval just done, or do the next portion of the current halving (one sub-interval the size of the one just done) then handle the rest of the interval as a whole. If we assume that the original interval was optimum in the region, but some anomalous situation (such as engaging a one-sided constraint) caused one point in the interval to require a smaller interval, then we note that point is either in the subinterval just evaluated or in the next subinterval of that same size. With this observation it is seen that it is better to do the next portion of the current halving (one sub-interval the size of the one just done) then handle the rest of the interval as a whole.

The value for maximum should not be chosen too large because we should reevaluate the efficacy of the default value frequently so that we can respond quickly to rapid changes in the environment.

The algorithm we have described should oscillate above and below the optimal value in order to continually test the waters. A dramatic speed up was noted when this code was introduced to the program, the program was set to print dots for matrix evaluations and h's for halvings, the improvement in the distribution of these symbols was visually dramatic.

No effort has been made to quantify or improve this simple algorithm, this is left for some later version of the

model. One caveat is that time intervals can become too small for the numerical accuracy available and this routine would fail.

Implementation of the Matrix

Matrix rows are reserved for: elements, ground membranes, and constraints. Ground membranes are elements (links) which are not explicitly in the tree structure, but are a phantom (very light) extension of the robot (to include the earth) when building the matrix - a ground membrane requires three rows in the matrix just as an ordinary element. Extra columns of the matrix array are reserved for accelerations. Figure 6 shows a sample problem with three elements, figure 7 shows the matrix setup, and figure 8 shows the solution to the problem. Gravity is set to 50 units.

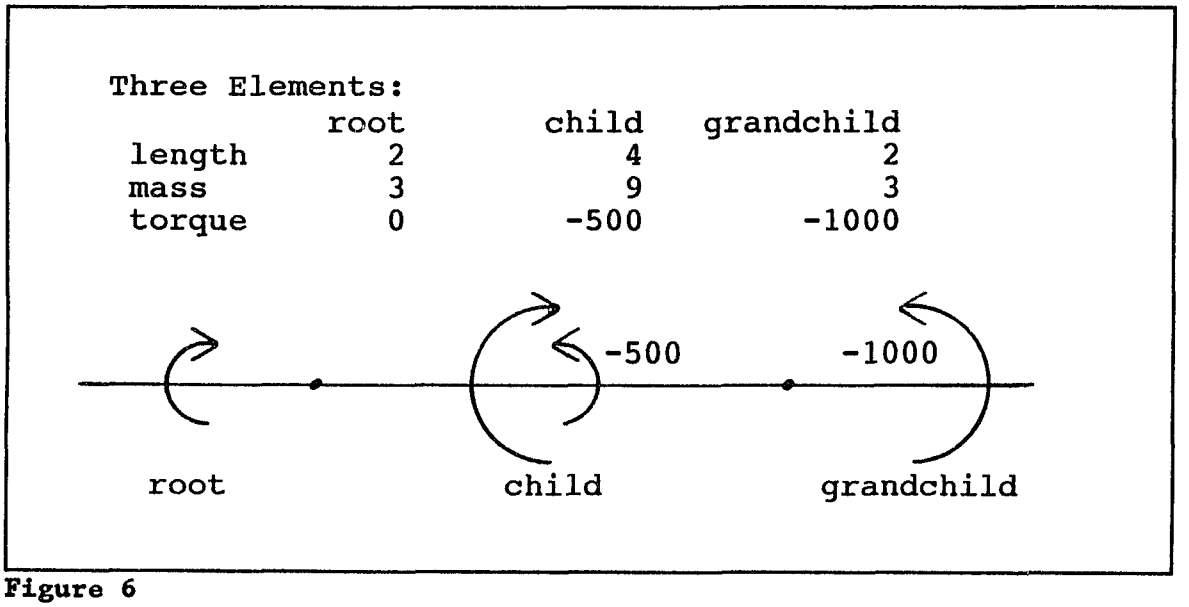

Figure 6

MATRIX:

3	0	0	0	0	0	0	0	0	0	0	1	0	= 0	
0	3	0	0	0	0	0	0	0	0	0	0	1	= -150	root
0	0	1	0	0	0	0	0	0	0	0	0	1	= 500	
0	0	0	9	0	0	0	0	0	1	0	-1	0	= 0	
0	0	0	0	9	0	0	0	0	0	1	0	-1	= -450	child
0	0	0	0	0	12	0	0	0	0	2	0	2	= 500	
0	0	0	0	0	0	3	0	0	-1	0	0	0	= 0	
0	0	0	0	0	0	0	3	0	0	-1	0	0	= -150	grandchild
0	0	0	0	0	0	0	0	1	0	1	0	0	= -1000	
0	0	0	1	0	0	-1	0	0	0	0	0	0	= 0	child vs
0	0	0	0	1	2	0	-1	1	0	0	0	0	= 0	grandchild
1	0	0	-1	0	0	0	0	0	0	0	0	0	= 0	root vs
0	1	1	0	-1	2	0	0	0	0	0	0	0	= 0	child

Figure 7

SOLUTION:

1	0	0	0	0	0	0	0	0	0	0	0	0	= 0	
0	1	0	0	0	0	0	0	0	0	0	0	0	= -182.936508	
0	0	1	0	0	0	0	0	0	0	0	0	0	= 101.190476	
0	0	0	1	0	0	0	0	0	0	0	0	0	= 0	
0	0	0	0	1	0	0	0	0	0	0	0	0	= 57.142857	
0	0	0	0	0	1	0	0	0	0	0	0	0	= 69.444444	
0	0	0	0	0	0	1	0	0	0	0	0	0	= 0	
0	0	0	0	0	0	0	1	0	0	0	0	0	= -238.492063	
0	0	0	0	0	0	0	0	1	0	0	0	0	= -434.523810	
0	0	0	0	0	0	0	0	0	1	0	0	0	= 0	
0	0	0	0	0	0	0	0	0	0	1	0	0	= -565.476190	
0	0	0	0	0	0	0	0	0	0	0	1	0	= 0	
0	0	0	0	0	0	0	0	0	0	0	0	1	= 398.809524	

Figure 8

Figure 7 is partitioned so that the mass matrix and Jacobian are easily identified. The values on the right side of the equations in figure 7 are forces while those in figure 8 are accelerations.

The matrix is built in a single pass of our data structure. Each child does have to send torque data to its immediate parent; except for this minor point the matrix can

be built instantaneously on a parallel machine. The matrix is solved using a Gaussian Elimination technique; partial pivoting can be use to increase the accuracy or sparse matrix procedures can be use to speed the process. We did neither, anticipating the next phase of the project which will be implementation on a parallel machine, where we can use the matrix LU Factorization technique parallelized by Lord, Kowalik and Kumar [29].

Constraints and Forces

If a constraint is always in place - such as the standard kinematic revolute joint constraints or joints fixed to a specific angle - then it is a workless constraint and may be handled by providing a line in the Jacobian of the Lagrange matrix. One-sided constraints such as joints with limits and ground limits, when they come into play, may have velocities which will cause work to occur. We address this problem by using springs and dampers instead of one-sided constraints, these forces are determined from the previous and projected positions and velocities - they will be zero when not in affect.

Several problems are solved by the use of spring and damper forces in place of one-sided constraints:

- One-sided constraints can be deployed only at the appropriate position and only when velocity and acceleration are both brought to zero,

- Which one-sided constraints are "in play" at a given moment can be a highly interactive proposition,
- With the forces (in the two dimensional case) our set of independent variables remains the same - one vertical, one horizontal and all nonfixed angles,
- With the forces our Jacobian structure does not change and our matrix should always be invertible.

We note that even if the Jacobian (within our matrix) were not linearly independent, if Gaussian Elimination is employed to solve our matrix then redundant rows (and corresponding columns) can be identified and removed during the process.

When spring and damper forces are in affect they will change the matrix (the function being modeled) from each tiny time interval to the next; this does not obviate the use of multistep methods, but if the function has changed, how much do previous points have to contribute to the knowledge of the current function? Indeed, they can change the function from one (re)projection to the next in the same time interval - changing the function as we try to model it! It is unlikely that this would jeopardize convergence, but as it happens, we have already decided to avoid reprojections and seek optimally small time intervals instead.

Dependent and Independent Variables

It is important to know which variables are dependent and which are independent, this information can be used to remove redundant constraints from the Jacobian and to do kinematic adjustments to the dependent variables in lieu of accepting the results of their integration.

The kinematic equations are represented in our matrix by their derivatives in the Jacobian, which contains the angles but not the horizontal and vertical coordinates; due to integration approximations, kinematic errors will accumulate and the elements of the body will tend to drift apart. We found that the accuracy of our model was adequate for some purposes without any employing kinematic adjustments.

To incorporate kinematic adjustments only positional adjustments need be added - recall that we calculate velocities from positions. The procedure is simple because, using spring and damper forces, our set of dependent and independent variables is always known. The set of independent variables includes one vertical coordinate, one horizontal coordinate and all angular coordinates except those fixed at a specific angle.

Our adjustment procedure is to find the vertical coordinate, y_1, of the lowest ground membrane then recalculate all horizontal and vertical coordinates coming down from the root element to the bottom of our tree. Again we find the vertical coordinate, y_2, of the lowest ground

membrane then adjust all of the vertical coordinates by adding $y_1 - y_2$. This approach prevents ground penetration from increasing or decreasing due to the adjustments.

The situation is somewhat more complex in the three dimensional case where, using Euler parameters, each element has an extra dependent variable. More general methods for determining dependent and independent variables are available from Wehage [41] or from Kim and Vanderploeg [25].

Extension to Three Dimensions

The two dimensional case which we have implemented is fairly simple, each element of the dynamical body has three coordinates: R_1 and R_2 defining the position of the center of the element and θ defining the orientation of the element. The two dimensional rotation matrix used to build the kinematic equations (which are used in the Jacobian, the velocity equations and the position adjustments) is

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

An element in the three dimensional case is not simply a line segment but three orthogonal segments crossing at their centers. Specifying the location and orientation is a more complex task. R_1, R_2 and R_3 are adequate to define the position of the center of an element but three variables are required to define its orientation. Unfortunately, any three

variable approach will lead to singularities where the orientation will not be uniquely defined. This problem can be avoided by using four variables, three of which will be independent and one of which will be dependent. To this end we will use Euler parameters as described below.

Let $\mathbf{v} = [v_1, v_2, v_3]$ be a unit vector in a three dimensional Cartesian space and let θ be the angle of rotation around that vector. Now let $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \theta_3]$, where

$$\theta_0 = \cos\frac{\theta}{2}$$
$$\theta_1 = v_1 \sin\frac{\theta}{2}$$
$$\theta_2 = v_2 \sin\frac{\theta}{2}$$
$$\theta_3 = v_3 \sin\frac{\theta}{2}$$

These are the four Euler parameters. Note that given θ_1, θ_2 and θ_3 one can determine $\mathbf{v}$, θ and θ_0. These parameters can be used to define a three by three rotation matrix

$$A = \begin{bmatrix} 1-2(\theta_2)^2-2(\theta_3)^2 & 2(\theta_1\theta_2-\theta_0\theta_3) & 2(\theta_1\theta_3+\theta_0\theta_2) \\ 2(\theta_1\theta_2+\theta_0\theta_3) & 1-2(\theta_1)^2-2(\theta_3)^2 & 2(\theta_2\theta_3-\theta_0\theta_1) \\ 2(\theta_1\theta_3-\theta_0\theta_2) & 2(\theta_2\theta_3+\theta_0\theta_1) & 1-2(\theta_1)^2-2(\theta_2)^2 \end{bmatrix}$$

Note that when $\mathbf{v} = [0,0,1]$ the matrix reduces to

$$A = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which is the three dimensional version of the standard two dimensional rotation matrix above.

Consider a joint between two elements in a two dimensional situation, we can determine the location of this point by applying the rotation matrix of each reference frame to this point on the corresponding element. We set the results equal to one another because the two points must coincide - this is how we developed the kinematic constraints for the two dimensional case. A similar analysis will yield three of the kinematic constraints for the three dimensional case, but we must also restrict the freedom of the joint. A revolute joint in three dimensions will have one degree of freedom, this freedom may be restricted by spring and damper forces but that will not affect the current analysis as the use of one-sided constraints would have.

Consider the parent element with no rotation, let the user define a unit vector parallel to the axis of rotation which will be allowed for the child element. If the parent is rotated then its rotation matrix operating on this vector must retain equality with the unit vector along the axis of rotation of the child. The remaining three of the six kinematic equations can be obtained from this equality. There is now one kinematic equation for each of the dependent variables.

In the two dimensional prototype we allowed no choice regarding which end of a child is joined to (either end of)

the parent. This reduced the number of relationships which had to be addressed from four to two with no loss of generality. A similar tact in the three dimensional case - allowing only a specific child end point (one of the six) to connect to any of the six parent end points - will reduce the number of cases from thirty six to six.

While the quadratic velocity vector disappears in our two dimensional case when the mass is taken at the center of the element, this is not true in the three dimensional case so this value must be calculated and added to our current forces. It is no longer particularly beneficial to take the mass as the center of the element, so more complex mass matrices should be allowed for greater generality.

Synopsis

The activity over a time interval t (from configurations subscripted 0 to those subscripted 1) is modeled by the following three simultaneous sets of equations:

$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + \boldsymbol{v}_0 t + \frac{(\boldsymbol{a}_0 + \boldsymbol{a}_1)\, t^2}{4}$$

$$\boldsymbol{v}_1 = -\boldsymbol{v}_0 + \frac{2(\boldsymbol{p}_1 - \boldsymbol{p}_0)}{t}$$

$$\boldsymbol{a}_1 = \boldsymbol{F}(\boldsymbol{p}_1, \boldsymbol{v}_1, \boldsymbol{f})$$

where F is the inverted Lagrange matrix operating on f, the set of forces associated with the ending state. Although each of these sets of equations can be readily solved alone,

together they must be solved iteratively, primarily because of the nonlinear trigonometric formulations of the Jacobian within the matrix. This matrix was seen to be:

$$\begin{bmatrix} M & C_q^T \\ C_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q \\ Q_d \end{bmatrix}$$

Since the thrust of this dissertation is control of highly nonlinear systems, we must point out that the matrix can be trivially manipulated to determine forces where the desired accelerations (or equivalently, positions or velocities) are known.

The prototype we have developed is readily parallelizable and is compatible with the use of Euler parameters for the purpose of three dimensionalization. Our simple algebraic integration formulas were derived from a starting function. They can easily be replaced with algebraic integration formulas derived from a multistep function, this would make the accuracy of our model comparable to that of prevailing engineering models.

CHAPTER 4

CONTROL MODEL

Our long term objective is to develop methods to obtain synergistic control of the robot model. To this end we develop a control package shell which interfaces the robot model and control programs we may develop. Attention must be given to the interface so that reality is duplicated in respect to the response time delays which will occur. We will discuss this feature as well as three techniques which can be used to explore the synergism problem: genetic algorithms, neural nets and a Newtonian control system. Finally we will make observations about a particular robot we will be investigating - a human torso.

Control Package Shell

Communication between the control program and the robot simulation model is accomplished through two ram disk files so that the two systems remain entirely distinct. The data flow is as follows: The control program receives the configuration information from the model and responds with the torque values for the actuators corresponding to the body joints - the robot model receives these forces and applies them to the body in small time intervals then returns the new configuration information to the control program.

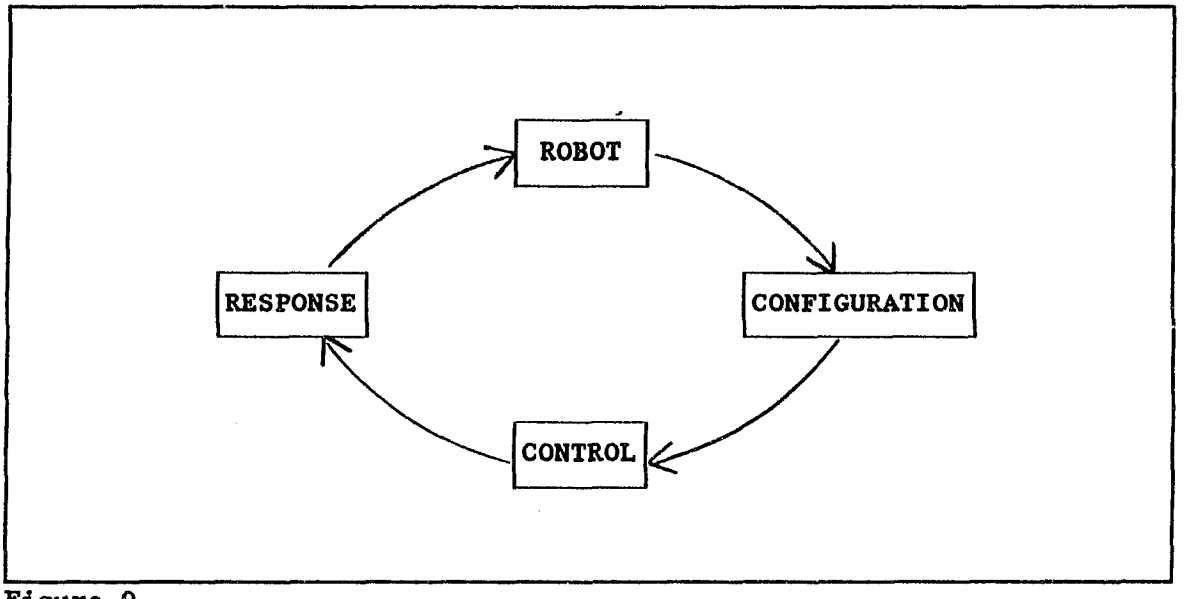

Figure 9

If synchronized multitasking is available then the two processes can be run concurrently. Our control package can simulate concurrency and can introduce a response time delay, as depicted in figure 10, using a loop with the following instruction sequence:

Robot: Reads & Writes & Processes;

Controller: Writes & Processes & Reads.

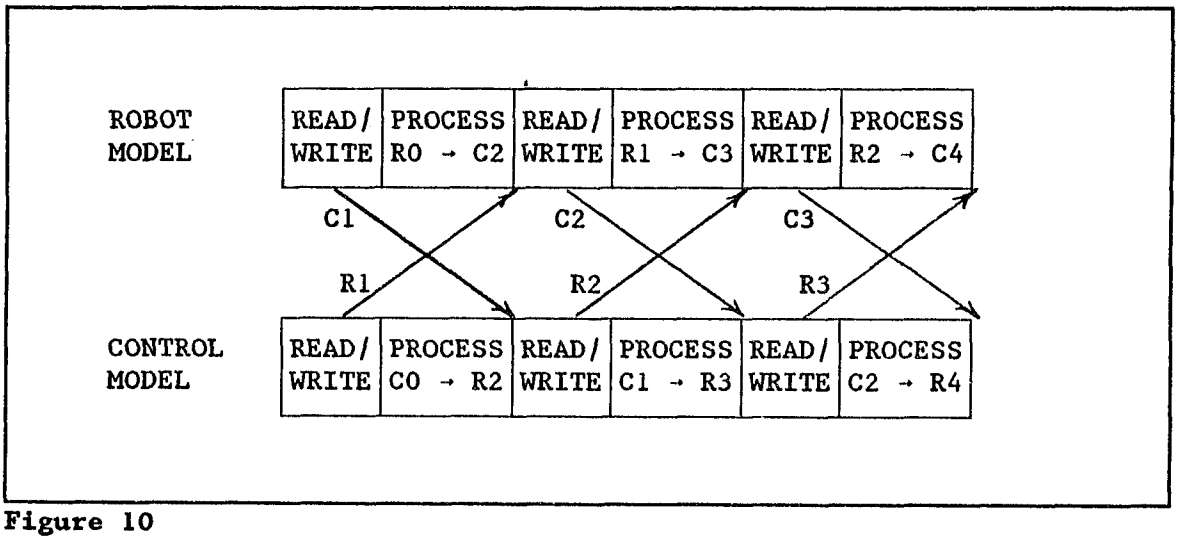

Figure 10

An alternative approach is to assume that the control process is virtually instantaneous, this is depicted in figure 11 and employs a loop with the following instruction sequence:

Robot: Writes & Reads & Processes;

Controller: Writes & Reads & Processes.

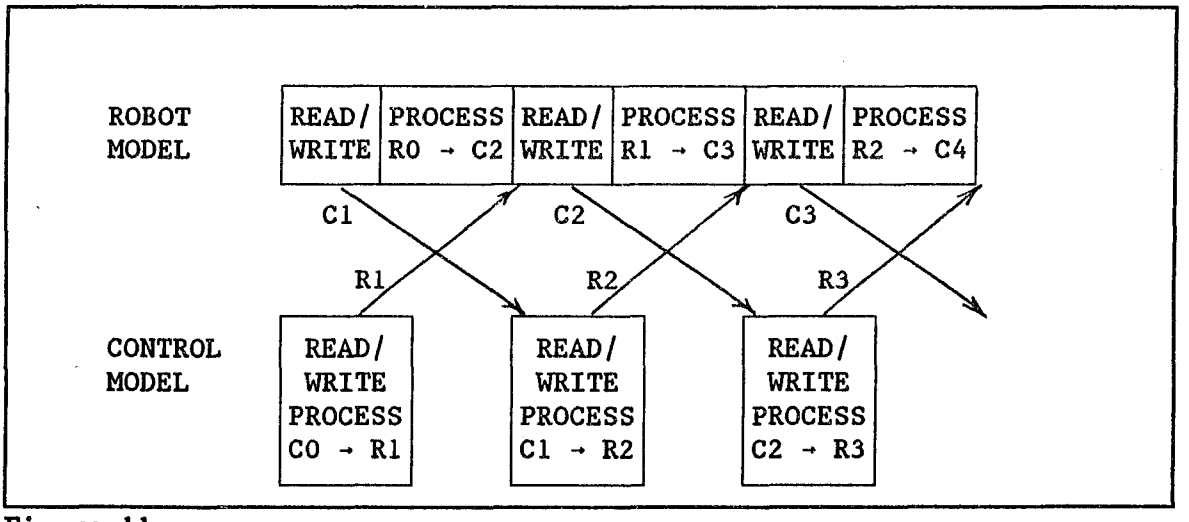

Figure 11

When we develop our Newtonian Controller we will assume the control process is virtually instantaneous, but the basic approach we will use is applicable to either assumption.

The configuration information sent by the robot includes joint angles, joint velocities, booleans indicating whether ground points are in contact with the ground, and an angle and angular velocity from one element which has been assigned the function of being the balance (analogous to the inner ear); the responses received by the robot are simply the torques to be applied to the joints. Additional nonlinearity

could be injected at this point if we were to scale actuator torques to some function of the current joint angle, we do not do this in our current experiments.

The robot simulation model is a foreground process, providing a real time graphical display of the body (this data can also be recorded and replayed). The control model is a background process, but some of the robot model single key commands can instruct the control model to display information for the user and to receive data from the user.

Genetic Algorithm

A human body is not simply connected elements with actuators stuck on them as are our robots. Sinews, bones, joints, muscles, nerves, networks of neurons, etc. were all designed and fitly put together for the purpose of walking. This design took place over aeons via a genetic algorithm process. In a similar way we might consider using genetic algorithms to answer such questions as: When do joint limits come into play and what should be the values of the spring and damper coefficients? We could attempt to build a linear controller for our robots and use genetic algorithms to find the best coefficients for those linear equations.

The success of DeGrais [9] in using genetic algorithms to determine weights in neural nets is another compelling reason for considering them in our work - although, he did

not use crossover and genetic algorithms without crossover are hardly worthy of the name.

A genetic algorithm is provided as a tool for the development of control methodologies. Goldberg [14] wrote the definitive book on genetic algorithms (see also Iyengar [20]) which are a biologically inspired paradigm useful in solving nonlinear problems. One starts with a gene pool - a diverse set of strings - all strings are evaluated with respect to an objective function representing how well the string would solve the given problem. A subsequent generation of strings is created by mating strings from the former generation - the selection of strings to be mated is random, but weighted in favor of strings with good objective function values. The mating of two strings, or crossover, is simply a matter of taking the first part of one string, cutting it off at some random point, and filling in the rest from the latter part of the second string. An infrequent random mutation of a string is beneficial to the process.

Thus the three features of a genetic algorithm are:

- Selection,
- Crossover,
- Mutation.

How genetic algorithms work is no longer a mystery, a rather involved theoretical foundation has been laid out and may be found in Goldbergs's book. At this point we note that genetic algorithms work with a coding of a parameter set, not

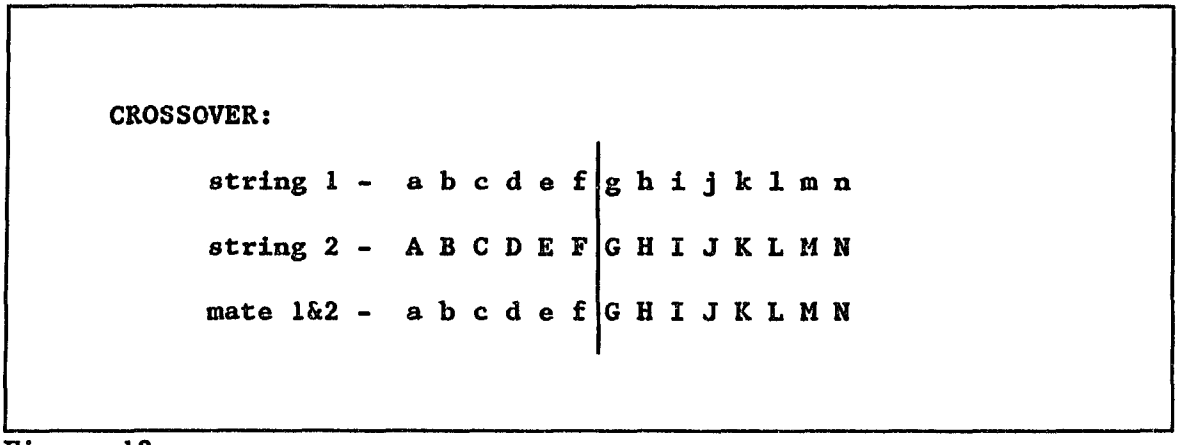

Figure 12

with the parameters themselves; they work with a population of strings, not a single point; they use payoff (the objective function) information, not derivatives or other auxiliary information; and they use probabilistic transition rules, not deterministic rules.

A chromosome is a string of bits representing the encoding of a set of parameters; for instance a string of five 16 bit binary unsigned integers contains 80 bits which could represent 40 parameters each ranging from 0 through 3 (2 bits each), it could also represent 10 parameters each ranging from 0 through 255 (8 bits each), etc.

An optional approach which one might consider is to leave the string in a decoded form - for example a string of 40 integers each allowed to take on values 0 through 3. Crossover would require encoding the two parameters at which the crossover occurs and performing a local crossover on them

then decoding the resulting parameter. Using this structure would require more space but would substantially reduce the number of decodings required.

The standard approach for programming a genetic algorithm is to initialize the first generation of N strings with random values, compute the value of the objective function for each string, then build a weighted "roulette wheel" from these values. Two strings are randomly selected from the wheel and mated using the crossover described. This selection and mating is done until a new generation has been built. The process is repeated by computing the value of the objective function for each string, etc.

Our approach is to initialize an array of 2*N strings with random values, compute the value of the objective function for each string, collect the best N strings into the first half of the array, reinitialize the second half of the array with random values, compute the value of the objective function for each of these strings, again collect the best N strings into the first half of the array, again reinitialize the second half of the array with random values, etc. We may do this several times in order to obtain a better survey of the terrain for the first generation.

Gathering the strings with the best values into the first part of the array (0,N-1) is accomplished by filling an ancillary array with 0 through 2*N-1, using quick sort to sort the first half of this ancillary array on the objective

function values (from bad to good) of the corresponding strings (0 through N-1); then using quick sort to sort the second half of this ancillary array on the objective function values (from bad to good) of the corresponding strings (N through 2*N-1). Starting at each end of the ancillary array we are able to replace the strings with the worst values in the first part of the array with the strings with the best values in the second part of the array. We continue replacing strings until the objective function values from the first part are better than those from the second part.

Building a new generation in the second half of the array from the strings in the first half is done as follows: assign the objective function value of the first string to the first string, assign the objective function value of the second string plus the preceding value to the second string, assign the objective function value of the third string plus the preceding sum to the third string, etc. The last string (N-1) will, therefore, be assigned the sum of all of the objective function values. Selecting a weighted random string is now a simple matter of picking a random number scaled to the sum of the objective function values and doing a binary search on the construct we just described. Direct selection from a wheel would be faster than the binary search, but this would probably be offset by the time and space that would be required to construct such a structure -

this depending on the units of the objective function value and the accuracy required.

The standard genetic algorithm retains good strings from generation to generation because they mate with themselves - causing more and more duplications of these strings. This process is, in fact, helpful to the selection feature previously mentioned.

One of the great strengths of genetic algorithms it their ability to tolerate fuzzy objective functions. We note, however, that it is often the case that the objective function is well defined and its value for any given string will not change. It seems reasonable in such cases to discourage the duplication previously described. Without duplication, however, we could lose better results in previous generations as well as diminish the effectiveness of the selection feature.

We will explore the idea of avoiding, instead of encouraging, duplication. To avoid losing good strings we will employ the feature already described whereby we can collect the best strings into the first half of the array - to combine the best of the current and previous generation - from this combination strings can be selected and mated for the next generation.

We have implemented duplication avoidance in two ways. The first approach prevents duplicates from being gathered into the first part of the array. During the initialization

process these strings must be compared to the previous strings, from this point on duplicates can be culled when the two halves of the (ancillary) array are sorted by objective function value. Not every two strings with a matching value is the same, but every two strings that are the same will have a matching value. With this approach duplicates are eliminated after being evaluated; but if we preclude any string from ever being mated with itself, the likelihood of duplication is fairly low.

The second approach is to use a hash table (accessed by hashing a string) which contains the index numbers for all currently existing strings. When a new string is created it is passed to the hash table routine which compares it to all strings with the same hashing, if is not a duplicate it is added. If it is a duplicate, the hash table routine responds that it cannot be added, it may then be mutated or remated as desired, and passed again to the hash table routine. This approach to avoiding duplicates does so before evaluating the objective function - allowing us to make the string unique before evaluating it.

Both of these methods are able to avoid duplicates between two generations, but neither eliminates duplicates over all generations.

Neural Net

Neural networks are considered in this dissertation because they are use by nature for the purpose of controlling the dynamic systems we want to model - the bodies of animals. Wasserman [40] wrote a fairly comprehensive book on neural nets (see also Gulati, Barhen and Iyengar [15]); this is a biologically inspired paradigm which is useful in modeling nonlinear problems. Modeled on the individual neuron and its interconnection with other neurons, a neural net is an array of artificial neurons. We will develop the single artificial neuron in a somewhat unusual way by considering a first-order liner model relating input variables x_1, x_2, x_3 to the output variable y:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \delta$$

Instead of using the familiar linear regression to determine the coefficient, we will use the delta rule (a simple gradient decent technique similar to Hebbian learning) this rule states that for each presentation of a training pair (input and output vectors), the error (the deviation from y) will be distributed to the coefficients in proportion to the corresponding input value.

$$w_0 = w_0 + \eta\delta$$
$$w_1 = w_1 + \eta\delta x_1$$
$$w_2 = w_2 + \eta\delta x_2$$
$$w_3 = w_3 + \eta\delta x_3$$

The term δ is the error and η is a learning rate. As it stands this neuron is not well suited for handling both large and small output values, this deficiency can be solved by applying a squashing function - forcing all output values into some small range such as (0,1):

$$\hat{y} = f(y) = \frac{1}{1 + e^{-y}}$$

$$\hat{y} - \epsilon = f(w_0 + w_1x_1 + w_2x_2 + w_3x_3) = f(net) = out$$

Rumlehart [37] has generalized the delta rule for this situation:

$$w_i = w_i + \eta\delta x_i$$

$$\delta = f'(net)\epsilon$$

For the squashing function chosen

$$f'(net) = f(net)(1-f(net)) = out(1-out)$$

We now have an artificial neuron and a method of training. This type of neuron is called an Adaline and handles nondiscrete input and output - the discrete corollary is called a percepton, it inputs and outputs only zeros and ones and uses a threshold function instead of a squashing function.

An Adaline is implemented by associating a weight with each input and an additional weight (neuron bias, which we

labeled w_0), sum the neuron bias and the products of the weights and their inputs to obtain net, then apply the squashing function. In practice a momentum term, α, is added to the formula allowing a larger value of η to be used:

$$\Delta w_i = \eta \delta x_i + \alpha \Delta w_{i,previous}$$
$$w_i = w_i + \Delta w_i$$

Since we used a linear model to begin with, we would expect the neuron to be limited in it modeling capabilities, this is in fact the case. The problem is solved by networking the neurons to form a neural net, where the outputs of neurons are use as inputs to other neurons. A simple conceptualization of a neural net is a set of nodes (neurons) connected to inputs and to one another with weights; but the most common configurations are layers of neurons. The terminology differs from one author to another, often the inputs are referred to as "input neurons" (they are not really neurons as described above - at most they may simply squash the input), or as layer zero. Layer one is the first layer of actual neurons, layer two inputs the outputs from layer one, layer three inputs the outputs from layer two, etc. All layers preceding the last (output) layer are called hidden layers. This configuration is called a (multilayer) feed forward network.

The squashing function is important in regard to combining neurons, because without these intervening

nonlinear functions the cascading of neurons - matrix multiplications - would mathematically degenerate into the equivalent of a single matrix and nothing would be gained. As we intimated a one layer neural net can model only a limited number of functions, two or more layers are required to model an arbitrary function [17]. The problem is known as linear separability [31] - the ability to train multilayer networks is important because of this.

Rumlehart [37] extended his generalized delta rule to handle multilayer networks, the method is called backpropogation. The output layer is trained as previously described, but the hidden layers do not have an explicit value for ϵ so the errors in the higher layer must be brought back through the appropriate weights of that layer:

$$w_i = w_i + \eta \delta x_i$$

$$\delta = f'(net)\epsilon = f'(net)\sum w\delta_{higherlayer}$$

Of course there are other network configurations and other training methods, but we concentrate on this procedure because it is more commonly used by the control community.

We should note that neural nets are sensitive to: the sequence of input (presentation of one set at a time seems to be much preferable, though less realistic); the type of function (XOR, OR, AND, etc.); the weight initialization range (and probably the type of distribution within that range, and probably even by neuron level); the momentum

parameter; the learning rate; and the neural net architecture.

We have several observations:

If we train several nets concurrently then we can tolerate a higher rate of divergence in exchange for lowering the convergence rates on the low end of the distribution. Let us define divergence as requiring more than 5000 presentations of various patterns. If we have an average of 50% divergence then this would, in most situations be unacceptable, but if we are training six nets concurrently our probability of divergence drops to 1.5%, which in many situations is perfectly acceptable.

Carrying on with the assumption of using six nets, and assuming we have a histogram of 100 convergence figures - the 50th value moves from a 50% possibility to a 98.5% probability, the 32nd value moves from 32% to 90% and the 11th value moves from 11% to 50%. This analysis is not precisely correct, but we will apply the necessary exactness and use these medians through out this paper.

Now we note that $f'(net)*\epsilon$ is not, as many authors might lead us to infer, an equivalent error - it is counter intuitive to such a notion - an intuitive equivalent error would use the reciprocal of $f'(net)$. $f'(net)$ is a learning inhibitor and we could question the efficacy of applying this inhibition to an output neuron which we know must eventually

attain the given training value - remember that the mathematics of the gradient descent has no regard for speed.

Since the intuitive equivalent error would effectively cancel $f'(net)$, we eliminate $f'(net)$ on output neurons - this looks like the simple delta rule, but it is not equivalent to eliminating the squashing function from the final layer - which would slightly diminish the ability to handle both large and small outputs.

A second option we will consider is to eliminate the squashing function from the final layer, as long as the target output is restricted (scaled or squashed) to (0,1) we should have less need to squash the actual output. If squashed, the interpretation of output values outside the range (0,1) would simply be that more training is required.

Another similar issue is the thought that any change in weights should improve the final error. This idea may be flawed, the underlying structure (hidden layers) may need to be changed first in a way which diminishes the final result in early stages. We will, however, proceed on the assumption that every change should improve the final error; we go even further and attempt to obtain a specific level of improvement in this error. Although η is still constant for all neurons in the network, it changes from one presentation to another. For each presentation we can change η dynamically to obtain better convergence. In particular, for each presentation we train the neural net, then multiply η by 2 if new error has

changed signs, or divide it by 3 if it is greater than one-third of the original error. We do the same thing again (to the original neural net) with this new η, if this level of improvement is not attainable in, say 10 operations, then we take what is achieved and move on to another presentation. This method will be refereed to as the 2/3 procedure.

An alternative approach is to change η dynamically using a secant method to seek an optimal value. Both methods are similar in nature, they may do multiple temporary training sessions for a given pattern using different values of η, then instantiate the final session. This second method will be referred to as the secant procedure.

Changing η dynamically requires more processing time than ordinary training, but requires fewer training sessions; it milks each pattern before going on to the next. This approach is useful in the situation where no scratch pad is available to remember training vectors and where training vectors are expensive to develop, in other words, a realistic situation. It should also provide a better recollection of the most recently presented test pair.

The 2/3 and secant procedures find and optimal η for each presentation so that the problem of using $f'(net)*\epsilon$ in the output layer is, perhaps, solved. On the other hand, a different η for the output layer could still prove useful; but solving for two η's for each presentation will obviously require some guideline. To this end we will combine the 2/3

procedure with the removal of f´ from the final layer. In a second approach we will combine the secant procedure with the removal of the squashing function from the final layer. We leave the reverse combinations for subsequent research.

All of these methods we have discussed can probably be generalized to multiple output networks by training each output individually - we have not tested this hypothesis and leave this issue for later study. Another candidate for subsequent research is the possibility of dynamically adding neurons whenever an adequate η cannot be found.

The program we developed to study neural nets allows the used to set the number of networks, the size of the input vector, and the number of neurons in the first layer, the second layer (if applicable), and in the output layer. The user may also set parameters specifying the use of standard backpropogation, the 2/3 procedure or the secant procedure; with f´, without f´, or without f (i.e., f is the identity function); sequential presentation, random presentation or random but no contiguous duplication. Several other parameters may be set, some of these are: the learning coefficient, the momentum coefficient, the initialization range, and the error reduction (for the secant and 2/3 procedures) coefficient. Matrix multiplication is used to evaluate the neural nets.

Neural Net Training

Usually neural net training is done with predetermined training vectors. This is not a reasonable situation for a realtime learning environment. There are several approaches to developing such training data for Neural Networks. One method for adaptive control uses a network to control the plant and another network to model the plant dynamics (or inverse dynamics). An alternate method uses a critic signal (indicating performance) to train the control network.

We note that in some simple cases it will be possible to employ a secant method adjustment procedure whereby the last response and result pair will be compared to this response and result pair to determine a response adjustment which should improve the result - the neural net is trained for the response plus this adjustment. This approach assumes near linearity from one response time to the next - hopefully, the neural net training would smooth out errors from this assumption. Figure 13 shows the sequence of events, the response error is calculated as follows:

$$\Delta Response = (Response_2 - Response_1) \frac{Desired - Result_2}{Result_2 - Result_1}$$

The Newtonian controller developed in the next section might be enhanced with a neural network which can adjust the mathematical "knee jerk" reactions with a modicum of intelligence. Such a neural net is a good candidate for this

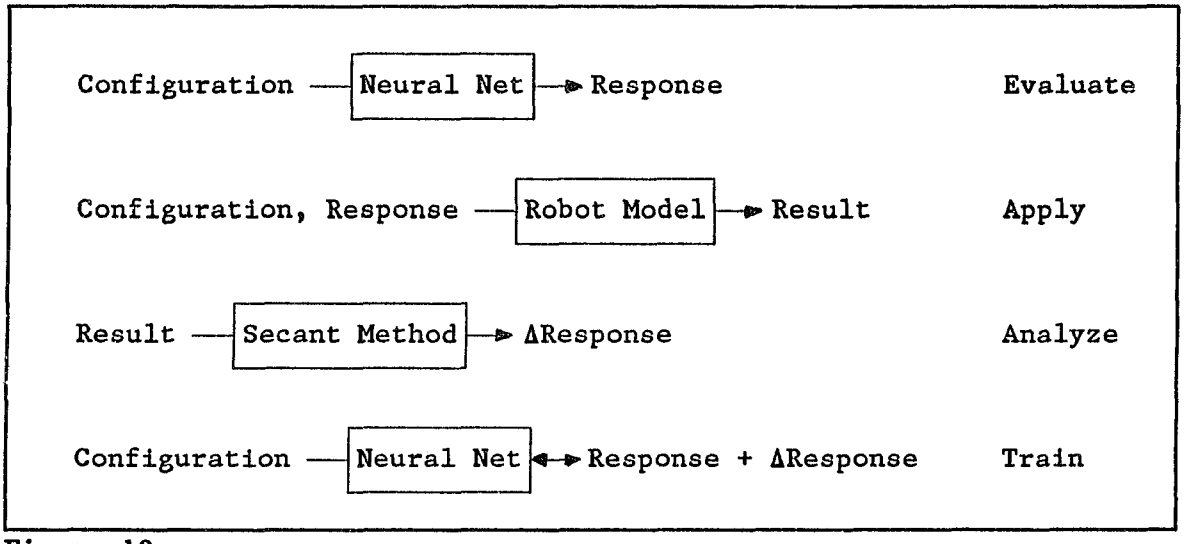

Figure 13

technique. There are several possible ways to setup such a scheme; one might be to define the configuration to be the current anticipated response as well as the last several responses - this configuration can easily be reconstructed for retraining purposes when the accurate results become available. The result could be in terms of displacement error or in terms of the subsequent force required, which should be diminished. There will be some false training when arbitrary external forces produce reactions which cannot be anticipated, but in general one hopes that with this neural network the controller could recognize when it is over (or under) reacting and learn to compensate in an appropriate manner.

Newtonian Controller

A controller base on Newton's laws is developed because it will be useful in validating our robot model, and because it will be useful in maintaining the balance angle for the robots we investigate.

A system can be defined in terms of its state, for example position and velocity, and its control, perhaps a torque which can be exerted. The dynamics of the system involve how the state changes under the influence of the controls. Employing feedback (this is called a closed loop system as opposed to an open loop system) one can allow the state of the system to automatically set the control.

Control theory requires that the relationship between the controller and the item to be controlled be linear in the following sense: if x_1 produces y_1 and x_2 produces y_2 then x_1+x_2 produces y_1+y_2; and for any constant C, $C*x_1$ produces $C*y_1$. Note that neither $y = x^2$ nor $y = mx + b$ are linear in this sense. Laplace transforms can be used to produce transfer functions which simulate this linearity for the higher order differential equations often found governing relationships. An approximation of linearity is often sufficient; indeed, all systems are nonlinear when variables are increased beyond some limits.

Although we will pose our problem in terms of linear coordinates, the analysis will be equally applicable to angular coordinates.

We will have need of an ability to control positions using torques - for instance, we may need to maintain a specific angle for the robot element designated as the balance element. This might appear to be a second order control problem, but we will apply Newton's second law (force equals mass times acceleration) and analyze the situation algebraically. We define the "effective mass" of an element as the acceleration divided by the force that caused it. This value is not constant but changes over time as the elements change their positions with respect to one another; we will restrict our attention to a neighborhood of time small enough that the effective mass is nearly constant. Figure 14 shows the relation between the response forces and the positions due to the response time delay; the response R_2 must be determined from position p_1 and applied to p_3 to obtain the desired position p_4.

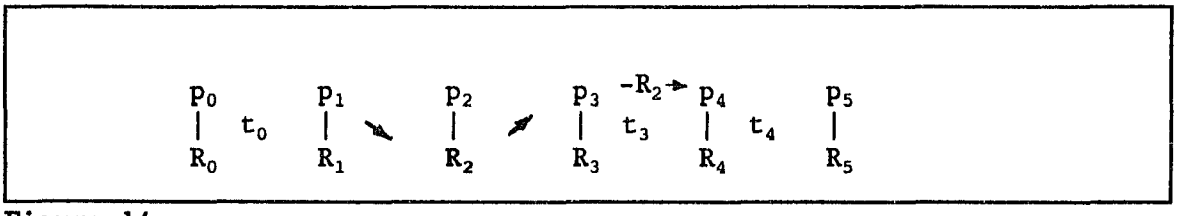

Figure 14

Figure 15 depicts our subscripting convention. The subscripts for positions and velocities indicate the time at which they occurred, while the subscripts for the response force and the commensurate acceleration indicate the time at which the response force was determined.

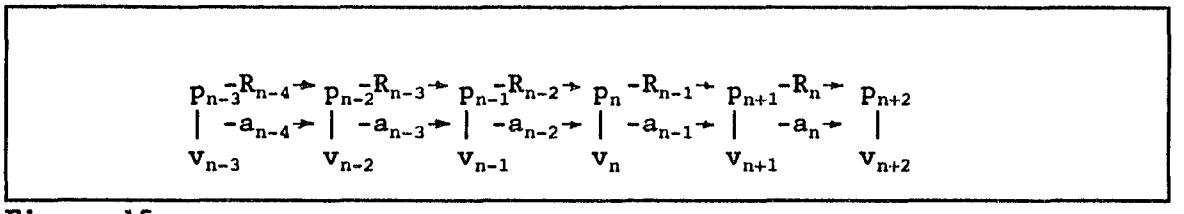

Figure 15

Now suppose that we are to determine R_n, we will have just obtained p_{n-1} and v_{n-1} from the sensors (robot model), we know any of the previous values as well. If we assume that the effective mass and the external forces operating on the element do not change too rapidly compared to the response intervals, then the ratio of the output to the input will be nearly constant. It can be calculated from the response for which all information is now available:

$$K = \frac{V_{n-1} - V_{n-2}}{R_{n-3}}$$

For convenience the time interval is assumed to be unity. We can project positions and velocities based on the above value - the most current information available.

$$V_n = K\,R_{n-2} + V_{n-1}$$

$$p_n = \frac{V_n + V_{n-1}}{2} + p_{n-1}$$

$$V_{n+1} = K\,R_{n-1} + V_n$$

$$p_{n+1} = \frac{V_{n+1} + V_n}{2} + p_n$$

since

$$p_{n+2} - p_{n+1} = \frac{v_{n+2} + v_{n+1}}{2}$$

p_{n+2} being the desired position, then

$$R_n = \frac{v_{n+2} - v_{n+1}}{K}$$

$$R_n = \frac{2\,(p_{n+2} - p_{n+1} - v_{n+1})}{K}$$

This approach is, however, a greedy and short sighted; seeking the desired position in this manner will cause us to attain the position with an undesired momentum which may lead to large oscillating response forces. We modify our approach by using two response intervals to reach a desired velocity, v_d, and the desired position, which we call p_d. Refer to figure 16 and note that p_{n+2} is now an intermediate position.

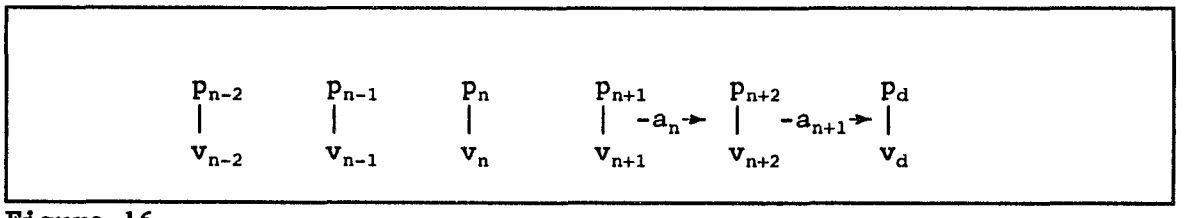

Figure 16

The following two equations

$$p_{n+2} - p_{n+1} = \frac{v_{n+1} + v_{n+2}}{2}$$

$$p_d - p_{n+2} = \frac{v_{n+2} + v_d}{2}$$

can be combined to obtain v_{n+2}

$$v_{n+2} = p_d - p_{n+1} - \frac{v_d + v_{n+1}}{2}$$

$$a_n = v_{n+2} - v_{n+1}$$

$$a_{n+1} = v_d - v_{n+2}$$

These relations lead to:

$$R_n = \frac{a_n}{K} = \frac{v_{n+2} - v_{n+1}}{K}$$

All things being perfect and $v_d = 0$ then a subsequent call to the routine with these same values (p_d and v_d) will yield a new a_n equal to the previous a_{n+1} and the new $a_{n+1} = 0$. Calls subsequent to this will yield $a_n = a_{n+1} = 0$. The question of stability, a bounded input producing a bounded output, is fairly straight forward at this point - under the stated conditions the process will obtain the desired velocity and position in just two steps.

Of course in a highly nonlinear environment there will be significant background forces to contend with. To accommodate these extraneous forces we replace R_n with $R_n + F$ in our previous equations, and assuming K and F are reasonably constant in this neighborhood we have:

$$\frac{R_{n-4} + F}{v_{n-2} - v_{n-3}} = \frac{R_{n-3} + F}{v_{n-1} - v_{n-2}}$$

$$F = \frac{R_{n-4}(v_{n-1} - v_{n-2}) - R_{n-3}(v_{n-2} - v_{n-3})}{2v_{n-2} - v_{n-3} - v_{n-1}}$$

If a severe change occurs in background forces there will be several intervals before it can be determined and corrected, ordinarily it can be corrected because the correcting forces are of the same order of magnitude as the offending forces; the problem concerns what may have been done just prior to learning of these forces and what may be done in response to them when, unknown to the routine, they may have quickly disappeared. A force that comes quickly and disappears is not much of a problem, but rapid drastic changes will certainly overcome the routine. A failure of this routine can be overcome by reducing the size of the response time interval to accommodate the assumption that background forces do not change too often in the course of several intervals. Oscillations will not occur due to an inherent overreaction by the routine, but will occur due to the unanticipated changes in its environment.

This procedure deviates significantly from standard control methods; and as it stands now, the routine has no sensitivity to what is realistically attainable - this deficiency can be ameliorated by adding an input parameter telling the routine how many time intervals it should use to attain the desired position and velocity. One simple trajectory would be to move from p_{n+1} to p_d in i response

intervals at a constant velocity during all but the first and last intervals (where we must transition to, or from, that constant velocity), finally stopping at p_d with the desired velocity as shown below:

$$p_d - p_{n+1} = \frac{v_{n+1} + v_{n+2}}{2} + \frac{v_{n+2} + v_{n+3}}{2} + \ldots + \frac{v_{n+i} + v_d}{2}$$

$$v_{n+2} = v_{n+3} = \ldots = v_{n+i}$$

$$v_{n+2} = \frac{p_d - p_{n+1} - \frac{v_d + v_{n+1}}{2}}{i-1}$$

For i = 2 this equation is the same one we derived before, so in our routine we simply divide v_{n+2} by i-1 if i is greater than 2. A parabolic trajectory could have been constructed in a similar manner.

Robot Torso

This dissertation would not be complete without an experiment on a relatively complex multibody robot of the genre for which our robot model was developed.

The current work is being performed on a six link robot simulating the torso and legs of a human - a stick man. This configuration is, of course, overly ambitious - it would be like taking someone who has never walked, putting their head in a neck brace, strapping their arms to their sides, cutting off their feet and hoping they could to learn to walk. None the less, it has been walked through several paces manually;

human intuition being used to assign torques at each response interval. This approach has been useful in that is allows the user to develop, first hand, a "feel" for what each forces is doing...it may be somewhat helpful to have such a familiarity with a process before trying to model it.

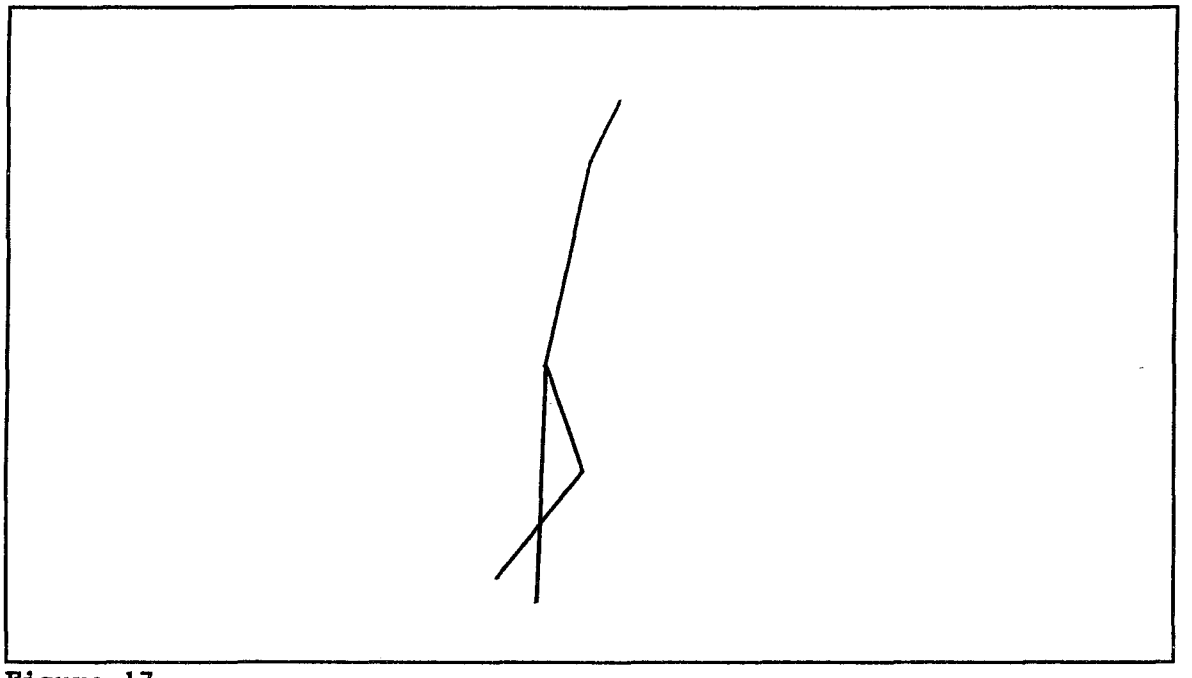

Figure 17

Theoretically, if we start with the same configuration and apply the same forces we will obtain the same result; we might hope to find a starting configuration and some corresponding torques which would lead us back to this same configuration. The problems with this hope are the expected variations and errors in the sensors and actuators, as well as the fact that this information is passed in discrete response time intervals. There is even the possibility that there is no set of initial configurations and torques which

will return to the same initial configuration in a single cycle, but rather must pass through several sets of configurations and torques before returning to the original configuration. If this were the case our problem would be much more difficult - we will presume we can find such a set. Then if we could learn to compensate for relatively minor deviations, each step could correct for the errors of its predecessor. There is even the possibility that the deviations could be in such a narrow range that these adjustments could be modeled linearly with enough accuracy to remain within this narrow range. This noted, our next task is to attempt to reduce the complexity (inputs and outputs) of the problem.

The controller has been modified to halve the complexity of the problem by determining when a cycle begins (a foot hits ground) and switching the inputs and outputs to use the same control process regardless of which leg is leading. We will call the grounded leg the balance leg and the leg moving forward the stride leg.

In dealing manually with the robot we noticed that restricting the range of motion of the knees allows us to be concerned with qualitative knee torques with little regard to the quantitative value. This thought is in keeping with the observation that the configuration constraints help reduce the complexity of the task - imagine how difficult it would be to learn to walk if your knees were truly double jointed.

The knee of the balance leg must remain straight throughout the step, this can be accomplished by setting it to the greater of: some small positive constant, or the balance leg's hip torque plus that constant. The knee of the stride leg is initially set to some small negative constant to lift the foot; by causing it to reverse the torque direction and increase its magnitude when the upper legs have past a certain angle (near parallel) with respect to one another, the knee sensors and controls need not be considered by the higher level controller.

With the above observation the control output is reduced to the two hip torques. It would be easy to adjust the robot model matrix so that the balance angle is held fast and the balance leg torque required to accomplish this would be computed and could be used to train a balance neural net, but this might be cheating. Instead we will use a Newtonian controller (previously described) to maintain the balance using the balance leg's hip torque.

A cycle begins when the foot hits the ground. At this time the angle of the two hip joints with respect to one another (we call this the stride) and the angle of the upper body (the balance) are adequate to completely describe the position configuration of the body. If the balance is well maintained then the upper body velocity and the balance can remain relatively constant. Given these observations, we can at the beginning of a step analyze three inputs - balance

angle, stride angle, and velocity (or response intervals in the step just completed) - to determine these same variables for the next step.

The balance angle will define a center of gravity, the optimal stride will fall on this point, a longer stride will decrease the velocity while a shorter stride will increase velocity. If one of these angles could be held constant, the other could be played against the velocity, reducing the complexity of the relationship. The secant method adjustment process previously discussed would be a reasonable approach for training a small neural net in realtime; substantial time would be required to accomplish this on our SIMD model.

CHAPTER 5

RESULTS AND DISCUSSION

Testing the Robot Model

The robot model was initially tested against simple symmetrical configurations to quickly catch most program or logic errors. The model was further verified by the success of the Newtonian control system.

Genetic Algorithm Convergence

The following experiment is a simplified version of a real life problem being considered at the time of this research. We have 40 users each requiring 0 to 100 units from any of 4 suppliers. Our string contains 40 parameters of 2 bits each, these parameters point to one of the four suppliers (0 through 3), indicating the supplier who must furnish the units. We seek a string for which the suppliers will furnish percentages of the total in a predetermined ratio: supplier 0 will furnish 30%, supplier 1 will furnish 10%, supplier 2 will furnish 5%, supplier 3 will furnish 55%.

One optimization function would be :

$$opt = \sum |pct_{actual} - pct_{desired}|$$

The perfect value is zero the worst is 200; but we want a function that maximizes...

$$obj = 200 - \sum |pct_{actual} - pct_{desired}|$$

The best will be 200, the worst can approach zero. In order to obtain a raw comparison of the various approaches we will apply no scaling to this objective function.

We will have an initial generation of 200 with no duplicates and subsequent generations of 100. Table II shows the results of performing twenty tests on each of four approaches: a standard genetic algorithm; deleting duplicates after evaluation; remating duplicates identified by a hash routine; and mutating duplicates identified by a hash routine. The average number of generations required to converged to obj > 198 is depicted for each approach.

Table II

	Mean	Std.	Median
Standard with duplication	919.5	1008.3	491.5
Dups eliminated after eval	13.2	3.3	13.5
Hash without mutation	12.2	2.0	12.0
Hash with mutation	11.1	2.2	11.5

It turns out that most of the improvement can be attributed to having obtained the best of both generations rather than to the elimination of duplicates. In table III the best of the new and old generations are combined as previously described with no effort to discourage duplicates.

Goldberg discusses overlapping generations as a method of increasing storage efficiency noting "We could maintain a

Table III

	Mean	Std.	Median
Best of two generations	13.9	4.0	13.0

single overlapping population and pay more careful attention to who replaces whom in successive populations." We have not found a discussion about obtaining an order of magnitude increase in convergence by combining two generations. What is being done is analogous to sending poor quality animals to market while breeding the better quality animals - even well beyond their life span using today's technology.

The applicability of this process to handle fuzzy objective functions has a reasonable chance of success because duplicates can be allowed. The standard algorithm in the above tests did not have the benefit of conventional scaling techniques - but neither did the proposed procedures. In table IV we have applied a linear fitness scaling, with the conventional factor of 2, to two of the algorithms, in order to compare the obtainable convergence rates. Additional parameter manipulation could very well bring convergence rates down still lower, but the interesting aspect of this new procedure is that convergence rates were low with no parametric manipulation at all - no a priori assumptions about the proper setting for parameters! It will be necessary to study these procedures as applied to the conventional genetic algorithm test functions.

Table IV

Scaled	Mean	Std.	Median
Standard with duplication	32.4	11.2	31.0
Best of two generations	11.2	2.1	10.5

A rapid convergence rate may well be at the expense of missing global maxima - premature convergence; if so, our routine can be tempered in several ways: gathering the best strings using a probabilistic scheme, or restricting the portion of old string which may be retained, etc.

Although our purpose was to investigate the affect of avoiding the reevaluation of previously evaluated strings, we appear to have serendipitously stumbled upon a more general technique.

Finally in table V we compare the best of two generations technique starting with an initial generation of 200, as we have been doing, versus the same procedure starting with an initial generation of 100. An equitable comparison of the two requires adding a generation to the first set (the 200); even so, it appears that increasing the initial generation survey does produce some tangible benefits.

Table V

Scaled	Mean	Std.	Median
Best of two generations (200)	11.2	2.1	10.5
Best of two generations (100)	13.6	7.5	12.0

Neural Net Convergence

The medians through out this analysis are pulled from histograms which are be derived by averaging 5 sets of 100 convergence attempts each. The value taken from the distribution, the 50th value for example, will be smoothed with a weighted average among adjacent values:

#50 = (#48 + 2*#49 + 3*#50 + 3*#51 + 2*#52 + #53)/12

The table column heading MED50 refers to the median (the 50th value obtain as described), likewise MED32 and MED11 refer to medians at the 32nd and 11th value. These medians will provide insight into the efficacy of employing parallel networks.

Perhaps greater accuracy could be obtained by averaging 10 sets of 50 (or 20 sets of 25, etc) and interpolating from the distribution, we chose not to do this in order to give more clarity to the low end of the distribution and avoid making any assumptions regarding the type of distribution this may be. This approach proves more than adequate in light of the disparities in the tests to be compared and provides the reader a means of considering the effect of using (six) concurrent networks.

Another measure may be employed when 500 evaluations would be too burdensome; 50 evaluations are used, the mean of the lowest half of the distribution is obtained and labeled AVG50 while the mean of the lowest 32 percent is

labeled AVG32. A similar measure was used when searching for parameter optimization.

We will use the term "presentation" to mean the training done on any given test vector before proceeding to another, as opposed to meaning sweeps through the set of all patterns; we will continue to define divergence as requiring more than 5000 presentations.

Refer to the test observations shown in table VI which represent convergence results from a neural net with two inputs, two neurons in a hidden layer and one output. The input was random except that no pattern was allowed to be input consecutively - training is slower for this than it would be for sweeps through the patterns. An output of .1 was used in place 0 and .9 in place of 1; the accuracy requirement was set to .01 for all patterns. The parameters are: η the learning coefficient, α the momentum, ρ range of weight initialization $(-\rho,+\rho)$, and υ the error reduction coefficient (the level of improvement sought after for the error delta). The three sets of data in table VI are for the XOR function, the OR function and the AND function. The XOR experiments were done first to compare optimized results for each method; the OR and AND were done later with the essentially the same parameterization to test the general usefulness of the methods and their parameters.

Table VI

XOR

	η	α	ρ	υ	MED50	MED32	MED11	DIV
1a standard	.25	.9	1.5		3128	2817	2544	15%
1b standard	1.00	.9	1.0		1067	919	765	16%
1c standard	2.00	.9	3.0		---	3882	928	63%
2 without f´	.25	.9	1.5		828	677	524	20%
3 without f	.20	.9	0.5		895	760	635	5%
4 2/3		.35	2.0	.3	324	256	210	21%
5 sec		.35	1.0	.3	396	345	308	19%
6 2/3 w/o f´		.4	2.0	.3	278	236	198	11%
7 sec w/o f		.9	0.5	.3	313	270	226	4%

OR

	η	α	ρ	υ	MED50	MED32	MED11	DIV
1a standard	.25	.9	1.5		---	---	4114	72%
1b standard	2.00	.9	3.0		692	467	301	6%
1c standard	2.00	.9	3.0		692	467	301	6%
2 without f´	.25	.9	1.5		735	549	429	0%
3 without f	.20	.9	0.5		1546	1256	801	3%
4 2/3		.35	2.0	.3	301	243	199	0%
5 sec		.35	1.0	.3	457	373	322	0%
6 2/3 w/o f´		.4	2.0	.3	271	227	197	0%
7 sec w/o f		.9	0.5	.3	510	470	420	3%

AND

	η	α	ρ	υ	MED50	MED32	MED11	DIV
1a standard	.25	.9	1.5		---	---	---	100%
1b standard	3.00	.9	4.0		859	525	221	6%
1c standard	2.00	.9	3.0		1899	1292	652	11%
2 without f´	.25	.9	1.5		2319	2063	1556	5%
3 without f	.20	.9	0.5		2507	1595	1058	8%
4 2/3		.35	2.0	.3	853	633	418	0%
5 sec		.35	1.0	.3	1174	931	754	0%
6 2/3 w/o f´		.4	2.0	.3	676	555	346	1%
7 sec w/o f		.9	0.5	.3	1059	935	702	2%

Tests 1a, 1b and 1c employ standard backpropogation:

(1a) the parameterization suggested by Rumlehart - choosing an initialization range suitable for XOR;

(1b) an optimal set of parameters for the function being tested;

(1c) a set of parameters which seemed close to an average of the optimal parameters for XOR, OR, and

AND - this was the optimal parameterization for OR.

(2) the f´(NET) is eliminated from the error delta of the final layer, the same parameters are used as in 1a.

(3) the squashing function is eliminated from the final layer, the parameters must be adjusted to obtain an improvement.

(4) we use the 2/3 procedure (discussed in a previous section) to retest a presentation several times seeking a η which improves the error delta by a ratio between zero and υ.

(5) we use the secant procedure (discussed in a previous section) to retest a presentation several times seeking a η which improves the error delta by a ratio between zero and υ.

(6) we use the 2/3 procedure and the f´(NET) is eliminated from the last layer.

(7) we use the secant procedure and the squashing function is eliminated from the final layer.

A reasonable effort was made to optimize parameters, although these may be local maxima, they probably do reflect the results one would expect to obtain.

The results for the XOR function are in the first data set of table VI. The specialized methods all out performed

even the optimized standard method. The 2/3 procedure, with or without f´, showed the best performance at the low end where six parallel neural nets would provide nearly a 50 percent increase in performance - the divergence rates are not acceptable for a single network but are good enough if multiple networks are employed. The secant method without f is the obvious choice if only one network is to be employed because of its relatively low divergence.

From the OR and AND data of table VI we determine that for the standard procedure, parameters with relatively good convergence rates for one function have poor rates for another. The specialized procedures were competitive with the optimized standard method even though they were not optimized for these functions. Again the 2/3 procedure, with or without f´, showed the best performance at the low end providing a 100 percent increase in performance for the AND function. The secant method without f is still a tolerable option for a single network situation.

A real life problem would probably encounter input vector values between 0 and 1. Again consider the methods and parameters used on the XOR problem in table VI, the first data set in table VII is the result of narrowing the inputs (0,1) to (.4,.5); that is, by an order of magnitude. Note that this first data set has the divergence point set to 100,000 presentations rather than the usual 5,000. It is obvious that narrowing the input slows convergence. Again

Table VII

XOR - narrow input (0.4,0.5) *divergence = 100,000

	η	α	ρ	υ	AVG50	AVG32	DIV*
1a standard	.25	.9	1.5		52049	50366	22%
1b standard	1.00	.9	1.0		33300	31926	30%
1c standard	2.00	.9	3.0		86054	78211	78%
2 without f´	.25	.9	1.5		23912	23043	10%
3 without f	.20	.9	0.5		37294	35697	6%
4 2/3		.35	2.0	.3	14191	9712	40%
5 sec		.35	1.0	.3	47345	17727	70%
6 2/3 w/o f´		.4	2.0	.3	17669	7316	52%
7 sec w/o f		.9	0.5	.3	12167	11126	2%

XOR - standard input (0,1)

	η	α	ρ	υ	AVG50	AVG32	DIV
1a standard	.25	.9	1.5		2875	2658	10%
1b standard	1.00	.9	1.0		863	793	16%
1c standard	2.00	.9	3.0		2613	1392	63%
2 without f´	.25	.9	1.5		622	551	20%
3 without f	.20	.9	0.5		715	656	5%
4 2/3		.35	2.0	.3	242	219	21%
5 sec		.35	1.0	.3	333	315	19%
6 2/3 w/o f´		.4	2.0	.3	221	204	11%
7 sec w/o f		.9	0.5	.3	255	235	4%

XOR - wide input (0,2)

	η	α	ρ	υ	AVG50	AVG32	DIV
1a standard	.25	.9	1.5		2457	1982	54%
1b standard	1.00	.9	1.0		993	582	52%
1c standard	2.00	.9	3.0		4257	3840	90%
2 without f´	.25	.9	1.5		1255	445	52%
3 without f	.20	.9	0.5		2492	1082	72%
4 2/3		.35	2.0	.3	1634	224	62%
5 sec		.35	1.0	.3	241	212	38%
6 2/3 w/o f´		.4	2.0	.3	302	204	42%
7 sec w/o f		.9	0.5	.3	196	171	32%

XOR - wide input (-1,1)

	η	α	ρ	υ	AVG50	AVG32	DIV
1a standard	.25	.9	1.5		1757	1709	16%
1b standard	1.00	.9	1.0		479	431	22%
1c standard	2.00	.9	3.0		524	301	40%
2 without f´	.25	.9	1.5		374	284	38%
3 without f	.20	.9	0.5		369	339	38%
4 2/3		.35	2.0	.3	118	101	32%
5 sec		.35	1.0	.3	194	173	30%
6 2/3 w/o f´		.4	2.0	.3	124	100	32%
7 sec w/o f		.9	0.5	.3	206	185	26%

the 2/3 procedure, with or without f´, showed the best performance at the low end, but the divergence rates are high enough to begin to jeopardize the use of these methods even with multiple networks. The secant method without f is a very viable option even for a single network situation.

This deleterious effect of narrowing inputs might be somewhat ameliorated if we could scale the input to a wider range without adverse effects. The second data set in table VII is the benchmark with the input value at 0 and 1. The third data sets is for the inputs (0,1) replaced with (0,2); that is, doubled. The fourth data set is for the inputs (0,1) replaced with (-1,1); that is, doubled then shifted to become centered at zero. It is seen that widening the inputs increases divergence, but this is not as pronounced when widened to (-1,1); here the low end values are brought down by twofold.

The 2/3 procedure showed the best performance at the low end, twice as good when the inputs are widened to (-1,1), and the divergence rates are acceptable for a multiple network situation. There is no discernable advantage to removing f´ (that is, to scaling the η of the final layer) when employing this procedure. The secant method without f and with input at (0,1) is a good choice if only one network is to be employed. The stability of this procedure is noteworthy, but the reason for this stability is not known. These procedures are less sensitive to parameterization the standard method.

We will now consider a larger neural net, we choose a two input neural net with six neurons in a single layer. This configuration is chosen because it has seven neurons and twenty-five weights - this is the same as that for a neural net with three neurons in the first layer and three neurons in the second, or a neural net with four neurons in the first layer and two neurons in the second. Because of this similarity these three structures can later be compared to one another in terms of the power of various methodologies on different structures.

Table VIII shows test data for this larger network, (1a) represents the standard backpropogation with a conservative learning coefficient. We see the expected improvement over the similar test in table VI. (1b) represents an attempt at optimizing MED11 - the penalty was high divergence. The optimum parameters for the secant procedure without a squashing function in the final layer (2) are an order of magnitude better than the standard method. The OR and AND values were also obtained for these same parameters and are very good. The convergence at the low end indicates that using parallel networks would produce improvements of 50 to 150 percent.

Similar levels of disparity are found dealing with sequential (sweeps) input as well as with totally random input, and again when dealing with more complex network architectures.

Table VIII

XOR

	η	α	ρ	υ	MED50	MED32	MED11	DIV
1a standard	0.25	.9	2.0		2101	1964	1773	0%
1b standard	1.00	.9	2.0		---	1913	772	53%
2 sec w/o f		.9	3.0	.3	245	201	162	1%

OR

	η	α	ρ	υ	MED50	MED32	MED11	DIV
2 sec w/o f		.9	3.0	.3	542	370	209	3%

AND

	η	α	ρ	υ	MED50	MED32	MED11	DIV
2 sec w/o f		.9	3.0	.3	313	245	159	1%

Newtonian Controller Experiment

The Newtonian controller was tested on a two element body where the element whose angle was to be controlled by an actuator's torques was dwarfed by the mass of the other element, this was too easy because no significant nonlinearities were introduced - making the other element lighter made the experiment more challenging. To develop the required history the first several response torques were +1000,-2000,+3000, and -4000; this also introduced motion into the system.

Experimentation has shown that the introduction of the velocity control feature reduced the magnitude of response force oscillation by an order of magnitude. The remaining errors may be attributable to the purposeful truncation of the robot sensor data and to the laxness of the robot model position requirements.

The introduction of the background force feature, in the absence of any background forces, approximately doubled the errors - but they were quite low to begin with. Impinging a tremendous constant background force had virtually no affect once the controller became aware of the situation - it accurately identified the magnitude of the force and compensated immediately.

The controller was able to ride out a sine wave of tremendous magnitude and wavelength of 20 intervals, dipping perceptibly when the maxima and minima passed. The background forces were not always very well identified, but the trend was apparent and the result was effective.

Experiments were performed in the absence of gravity and in the presence of gravity - the latter experiments bare some similarity to the familiar broomstick and inverted pendulum problems.

Robot Torso Experiment

This experiment employed a six link robot simulating the torso and legs of a human - a stick man.

Theoretically, if we start with the same configuration and apply the same forces we will obtain the same result; we might hope to find a starting configuration and some corresponding torques which would lead us back to this same configuration. Then if we can compensate for relatively

minor deviations, each step could correct for the errors of its predecessor.

The controller halves the complexity of the problem by determining when a cycle begins (a foot hits ground) and switching the inputs and outputs to use the same control process regardless of which leg is leading. The knee of the balance leg must remain straight throughout the step, this can be accomplished by setting it to the greater of: some small positive constant, or the balance leg's hip torque plus that constant. The knee of the stride leg is initially set to some small negative constant to lift the foot - its range has been restricted so it can rise only a few degrees; by causing it to reverse the torque direction and increase its magnitude when the upper legs have past a certain angle (near parallel) with respect to one another, the knee sensors and controls need not be considered by the higher level controller.

With the above observation the control output is reduced to the two hip torques. In the following cases a Newtonian controller is employed to maintain the balance using the balance leg's hip torque:

Applying a constant manual torque to the stride leg the designated balance angle was attained and maintained.

Applying variable manual torques to the stride leg the designated balance angle was attained and maintained except

for brief dips due to the controller's inability to anticipate changes in the stride leg torque.

Now the balance leg torque returned from the controller has the stride leg's variable torque subtracted from it before it is used (this is invisible to the Newtonian controller) so that the opposing forces will tend to cancel out and the controller will not have to determine these extraneous forces and suffer the consequences of a delay; the designated balance angle was attained and maintained quite well.

Leaving the balance controlled as above, a Newtonian controller is now also employed to control the stride leg. This new controller keys on the stride angle and angular velocity and is to attain a stride angle of twenty radians in sixteen response intervals. Although the balance was well maintained, the stride torques produced by the new controller were erratic, particularly when the stride leg's knee torques changed.

Repeating the above experiment, but with no stride leg knee torques we obtain improved, but still inadequate results. The conclusion is that the stride angle is not dependant enough on the stride torque alone, also the balance torque changes may overwhelm the new controller.

Other robots can also be investigated; adding feet and arms might be helpful. A plethora of experiments can be devised with just a little imagination.

CHAPTER 6

SUMMARY AND CONCLUSIONS

Defining the robot linkages and actuators in a tree structure significantly simplifies our problem. This structure used in conjunction with springs and dampers, rather than one-sided constraints, allows us to trivially identify a set of independent variables. At the same time this structure is adequate to handle the class of robots which are of interest to us.

The robot model prototype is completed, the next step is to put it on a parallel computer. The preponderance of the program's processing is spent building and solving the large Lagrange matrix - it can be built from our data structure in parallel and we know how to parallelize matrix evaluation by LU Factorization.

A concurrent task will be to extend the current and parallel versions to three dimensions. The information currently furnished to the Lagrange matrix from an element is obtained from a two by two rotation matrix, the information would now be obtained from a three by three rotation matrix. The quadratic velocity vector disappears in our two dimensional case when the mass is taken at the center of the element - this is not true in the three dimensional case so this value must be calculated and added to our current

forces. The use of the Lagrange matrix with multipliers has made the process readily extensible to three dimensions and even to elastic as well as rigid bodies.

If desired, the simple iterative starting function we employed can easily be replaced by a multistep procedure to make our process as exacting as contemporary engineering models.

A control package shell allows programs based on various control techniques to be interfaced with the robot model. Response time delays may be introduced into the interface to reflect reality.

A genetic algorithm routine was written for use with the controller. A modified algorithm was investigated, where poor solutions in the new generation were replaced by good solutions from the old generation. This method enhanced convergence on the problem to which it was applied and appears to be less sensitive to parameterization. Several additional modifications were tested and appeared to have some benefit. Although genetic algorithms may have influenced the makeup of the brain, it seems unlikely that animal learning itself could operate in this manner - that does not, however, diminish the potential of an artificial learning process using genetic algorithms.

Neural network backpropogation convergence can be enhanced, in terms of the number of presentations, with the use of any of several procedures. The 2/3 procedure showed

the best performance at the low end, twice as good when the inputs are widened to (-1,1), and the divergence rates are acceptable for a multiple network situation. There was no discernable advantage to removing f´ (that is, to scaling the υ of the final layer) when using this procedure. The secant method without f and with input at (0,1) is a good choice if only one network is to be employed. The stability of this procedure is noteworthy, but the reason for this stability is not known. This procedure also shows promise for a multiple network situation for more complex network configurations. An order of magnitude improvement in convergence rate are found dealing with sequential (sweeps) input as well as with totally random input, and again when dealing with more complex network architectures. These procedures are also less sensitive to parameterization than the standard method. Further investigation of these and similar routines is warranted.

An interesting experiment to be done in the future would be to combine our genetic algorithm routine with our parameter driven neural net routine to seek optimal methodologies.

We have produced a control procedure which uses the most current data available to compensate for response delays in order to obtain a desired position and velocity in optimal time. A failure of this routine can be overcome by reducing the size of the response time interval to accommodate the

assumption that background forces do not change too often in the course of several intervals. This is analogous to a reflex system providing quick local communication, as opposed to a distant higher level process. An interesting experiment would be to enhance the Newtonian controller with a neural net to adjust the responses - learning from "miscalculations" of the past. This experiment would be an ideal application for the neural net training procedure which was described in the neural net training section of this dissertation.

Our SIMD Lagrangian robot model is, perhaps, too slow for most robotics research; aside from the difficulty of implementation, this is probably the reason most control research is done on kinematic models. There has been a great deal of work done on these velocity controlled models - despite their diminished realism. Perhaps our Newtonian controller can take advantage of some of this work, because through it our realistic robot model may appear to be a velocity model.

The current work involves using the models and tools described to investigate the mysteries of perambulation. Our initial robot torso experiment attempted to solve the problem by reducing its complexity - to an extent we eliminated the synergism rather than addressing it - this approach is obviously inappropriate for larger problems. Although this approach has allowed us to look at the trees, the general solution will lie in an understanding of the forrest -

probably in some as yet unknown neural process. We now have several tools with which to pursue this elusive goal.

REFERENCES

[1] ABDALLAH, C., HORE, B., AND HUSH, D., "Model Following Using Multilayer Perceptrons", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.3, p. 1730, 1990

[2] ALBUS, J., "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)", *Trans. ASME - J. Dyn. Syst. Meas. Control*, vol. 97,pp. 220-227, 1975

[3] ALBUS, J., "Brain, Behavior and Robotics", *BYTE Books*, 1981

[4] ANTSAKLIS, P.J., "Neural Networks in Control Systems", *IEEE Control Systems*, pp. 3-5, April 1990

[5] ANTSAKLIS, P.J., "Neural Networks in Control Systems", *IEEE Control Systems*, pp. 8-10, April 1992

[6] ASH, T., "Dynamic Node Creation", Tech. Rep. ICS 8901, University of California - San Diego, 1989

[7] CHAND,S., AND LAN M.S., "Neural Network Augmented Control For Nonlinear Systems", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.3, pp. 1732-4, 1990

[8] CHEN, FUCHANG, AND KHALIL, H.K., "Adaptive Control of Nonlinear Systems Using Neural Network", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.3, pp. 1707-12, 1990

[9] DE GARIS, HUGO., "Genetic Programming: Building Nanobrains with Genetically Programmed Neural Network Modules", *IJCNN International Joint Conference on Neural Nets*, vol. 3, pp. 511-16, 1990

[10] DORF, RICHARD C., "Modern Control systems", *Addison-Wesley*, Philippines, 1980

[11] FERNANDEZ, B., PARLOS, A., AND TSAI, W.K., "A Novel Associative Memory For High Level Control Functions", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.4, pp. 2374-9, 1990

[12] FRIEDLAND, BERNARD., "Control System Design", *McGraw-Hill*, New York, 1986

[13] GLEICK, J., "CHAOS", *Viking Penguin*, 1987

[14] GOLDBERG, D.E., "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley*, Reading, Mass., 1989

[15] GULATI, S., BARHEN, J., AND IYENGAR, S.S., "Neurocomputing Formalisms for Computational Learning and Machine Intelligence", *Advances in Computers*, Academic Press Publications, vol.33, pp. 172-233, 1990

[16] HOSKINS, D.A., AND VAGNERS, J., "A Neural Network Based Explicit Model Reference Adaptive Controller", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.3, pp. 1725-9, 1990

[17] IRIE, B., AND MIYAKE, S., "Capabilities of Three-Layered Perceptons", *Proc. IEEE Int. Conf. Neural Networks*, pp. I-641, 1988

[18] IYENGAR, S.S., AND ELFES A., "Autonomous Mobile Robots: Perception, Mapping and Navigation", *IEEE Computer Society Press*, vol. 1, 1991.

[19] IYENGAR, S.S., AND ELFES A., "Autonomous Mobile Robots: Perception, Mapping and Navigation", *IEEE Computer Society Press*, vol. 2, 1991.

[20] IYENGAR, S.S., PRASAD, L., AND MORTON, D., "Structure of Genetic Algorithms in Biological Systems", *Structuring Biological Systems*, Ed. S.S Iyengar, pp 1-16, 1991.

[21] JACOBS, R., "Increased rates of convergence through learning rate adaptation", *Neural Networks*, vol. 1, no. 3, pp. 295-307, 1988

[22] JOHNSON, W.A.,LEAHY, AND M.B. JR., "Adaptive Model-Based Neural Network Control", *Proceedings 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, OH, USA, (Los Alamitos, CA, USA: IEEE Comput. Soc. Press 1990), vol. 3, pp.1704-9, 1990

[23] KANERVA, PENTTI., "Sparse Distributed memory", *MIT Press*, Cambridge, Mass., 1988

[24] KHORASANI, K., AND PATAL, R. V., "A Neural Network Architecture, For Model Reference Adaptive Control", *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, HI, USA, vol.5, pp. 2768-9, 1990

[25] KIM, S.S., AND VANDERPLOEG, M.J. "QR Decomposition for State Space Representations of Constrained Dynamic Systems", *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 108, pp. 183-188, June 1986

[26] KOSSLYN, S.M., AND KOENIG, O., "Wet Mind: the new cognitive neuroscience", *The Free Press Macmillan Inc.*, New York, 1992

[27] LANE, STEVEN H., HANDELMAN, DAVID A., AND GELFAND, JACK J., "Development of adaptive B-splines using CMAC Neural Nets", *Proc. Int. Joint Conf. Neural Networks*, Wash. D.C., 1989

[28] LANE, STEVEN H., HANDELMAN, DAVID A., AND GELFAND, JACK J., "Can Robots Learn Like People Do?", *Proceedings of the SPIE Conference on Applications of Artificial Neural Networks*, Orlando, Florida, 1990

[29] LORD, R.E., KOWALIK, J.S., AND KUMAR, S.P., "Solving linear algebraic equations on an MIMD computer", *Journal of the ACM*, vol. 1, pp. 103-117, 1983

[30] MACKI, JACK, AND STRAUSS, AARON, "Introduction to Optimal Control Theory", *Springer-Verlag*, New York, 1982

[31] MINSKY, K.L., AND PAPERT, S., "Perceptons", *MIT Press*, Cambridge, Mass., 1960

[32] MOODY, J., AND DARKEN C., "Learning with localized receptive fields", *Proc. 1988 Connectionist Model Summer School*, Eds. D. Touretzky, G. Hinton, and T.Sejnowski, Morgan Kaufman, 1988

[33] PARKIN, R.E., "An Interactive Robot Simulation Package", *Simulation* (USA), vol. 56, no. 5, pp.337-45, May 1991

[34] PITMAN, R.J.G, "Automatic Control Systems Explained", *St Martin's Press*, New York, 1966

[35] POGGIO T., AND GIROSI F., "Networks for approximation and learning", *Proc. IEEE*, vol. 78, no. 9, pp. 1481-1496, 1990

[36] RAIBERT, MARC H. AND HODGINS JESSICA K., "Animation of Dynamic Legged Locomotion", *Computer Graphics*, vol. 25, no. 4, pp. 349-358, July 1991.

[37] RUMLEHART, D.E., HINTON, G.E., AND WILLIAMS, R.J., "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing* (Eds. D.E. Rumlehart and J.L.McClelland), vol. 1, chap. 8, *MIT Press*, Cambridge, Mass., 1986

[38] SHABANA, AHMED A., "Dynamics of Multibody Systems", *Wiley*, New York, 1989

[39] VASILONIKOLIDAKIS, N., AND CLAPWORTHY, G.J., "Animated Human Walking Using Lagrangian Dynamics", *Computer Graphics 90. Proceedings of the Conference*, London, UK, (London, UK:Blenheim Online 1990), pp. 189-203, 1990.

[40] WASSERMAN, PHILIP D., "Neural Computing: theory and practice", *Van Nostrand Reinhold*, New York, 1989

[41] WEHAGE, R.A., "Generalized Coordinate Partitioning in Dynamic Analysis of Mechanical Systems", Ph.D. Thesis, University of Iowa, December 1980

VITA

Don Iglehart was born 11-27-45 in Jennings, Louisiana and graduated from high school in 1963, then attended college for several unrewarding semesters. He joined the Marine Corps. Reserve in 1965 and, after active duty, worked as a roughneck in the oil fields of Louisiana. He returned to the University of Southwestern Louisiana in 1968 and supported himself through college - earning a B.S. and an M.S. in mathematics with minors in physics - and was selected to be the marshal of the graduate class in 1973. Most of the ensuing years were spent as a programmer and systems analyst consulting in the development of Management Information Systems for both private and public entities. As a consultant to the State of Louisiana he computerized the criminal history system, designed and directed a rewrite of the Louisiana Mineral Information System (part of which is a royalty accounting system auditing an annual revenue of $500,000,000), provided the litigation support for the 8G suit against the federal government (an initial settlement sum of $600,000,000), and wrote the quantification model for Ernst & Young's audit of Texaco royalty payments (depending on the legal scenario the model determined Texaco and LL&E each underpaid by approximately $500,000,000). He resumed his formal education in 1989, while continuing his profession as a consultant, and is currently a candidate for a Doctoral

degree in Computer Science - specializing in robotics and databases - from Louisiana State University.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Don Iglehart

Major Field: Computer Science

Title of Dissertation: Synergistic Control of N-Body Computer Generated Robots

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

March 18, 1993