NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

The University of Alberta

A Hybrid Numerical/Knowledge Based System for Locomotion Control of a Multi-Legged Articulated Robot

by

Ahmed S. Mohamed

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

Department of Computing Science

Edmonton, Alberta
Spring 1989.

ISBN 0-315-52925-3

Canada

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Ahmed S. Mohamed

TITLE OF THESIS: A Hybrid Numerical/Knowledge Based System for Locomotion Control of a Multi-Legged Articulated Robot.

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1989

(Signed) .....Mohamed.....

Permanent Address:
142 Omar Loutfi Street
Sporting - Apt. 2
Alexandria - EGYPT

Dated 04/21/89

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **A Hybrid Numerical/Knowledge Based System for Locomotion Control of a Multi-Legged Articulated Robot** submitted by **Ahmed S. Mohamed** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy.**

....

Supervisor

....

....

....

....

....

Date ....

Abstract

Goal-directed interaction of articulated bodies (such as humans, high animal species, and robots) with the environment has long been a problem area for control engineers, mechanical engineers, biomechanical scientists, and behavioral psychologists. Nowadays it is a major problem area in the fields of robotics and graphical animation, where researchers seek to mimic purposeful movements and motion.

However, none of these fields yet offers any satisfactory solution to the problem of understanding, synthesizing, and learning coordinated movements. There are partial solutions in psychological studies, graphical animation systems, and robotics simulations. It seems that lack of a well-defined methodological framework for analysis of movements has been a major impediment to both experimental simulations and theoretical studies.

This thesis proposes a model for understanding, synthesizing, and learning coordinated movements within a cohesive framework which combines the mathematical rigor of approaches taken in robotics, biomechanics, and artificial intelligence literature with the behavioral relevance of psychological approaches. The model is demonstrated in a locomotion control system for a four-legged articulated robot.

One of the primary objectives of this research is to examine the effectiveness of merging the concepts of computer numerical control (mathematical rigor of approaches taken in robotics, and biomechanics) with the concepts of motor learning (knowledge-based processing as studied in artificial intelligence and behavioral psychology). The expected result is that both methodologies can act together to provide an autonomous motion control system that is capable of synthesizing and learning coordinated locomotive movements and improving its performance as a result of practice and experience.

The proposed locomotion control system currently navigates a four-legged articulated robot through a simulated environment containing obstacles, holes, inclines, rough terrain, etc. The equations of motion of the robot are solved by a recursive method, implemented on an IRIS-2400 graphics workstation. The robot must show some prudence in choosing the most appropriate locomotive skill at any point during its navigation: to slow down while turning a corner, to prefer the paths it has traveled on before, to reduce its total energy consumption in executing a mission, etc.

The emphasis in this research is two-fold: 1)acquiring high agility through a learning process, as might be required to quickly move from one point to another on a structure under construction, or needing repairs in an emergency; 2)enabling the robot to solve, 'on-board' and in real-time, complex problems of manipulator dynamics required for locomotion control.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

TABLE OF SYMBOLS

(X_I, Y_I, Z_I)	Inertial Frame: A frame fixed in the nonrotating earth.
(X, Y, Z)	Body fixed coordinate frame: A frame that has its origin fixed at the center of gravity of the robot body.
(X_b, Y_b, Z_b)	The components of the position vector of the center of gravity of the robot body relative to the inertial frame.
$(\dot{X}_b, \dot{Y}_b, \dot{Z}_b)$	The components of the translational velocity of the center of gravity of the robot body relative to the body fixed frame.
(ϕ, θ, ψ)	Body Orientation Bryant angles (also known as Cardan angles). These angles are defined such that when the body angles ϕ, θ, ψ are all simultaneously reduced to zero, the (X,Y,Z) axes are parallel to the (X_I, Y_I, Z_I) axes of the inertial frame. The rotation from (X_I, Y_I, Z_I) system to the (X,Y,Z) is accomplished by first rotating an angle ψ about the Z-axis, then an angle θ about the rotated Y-axis, and finally an angle ϕ about the rotated X-axis.
$(\dot{\phi}, \dot{\theta}, \dot{\psi})$	Body rotation rates expressed in the body coordinates.
$(\omega_x, \omega_y, \omega_z)$	Body angular velocity expressed in the body coordinate frame.
$(X_h, Y_h, Z_h)_i$	Leg i hip local fixed coordinate frame. Its origin is at the hip of leg i. The Z_{hi}-axis is parallel to the body Z-axis and directed downward. The Y_{hi}-axis is perpendicular to the plane of the leg segments, and the X_{hi}-axis is uniquely determined by maintaining a right hand coordinate system.
$(X', Y', Z')_i$	Leg i hip local non-fixed coordinate frame. This frame has its origin at the hip of leg i and rotates with the upper limb segment of the leg. The X'_i-axis is directed along the upper limb segment. The Y'_i-axis is perpendicular to the plane of the leg segments, and the Z'_i-axis is uniquely determined by maintaining a right hand coordinate system.
$(X'', Y'', Z'')_i$	Leg i knee local non-fixed coordinate frame. This frame has its origin at the knee of leg i and rotates with the lower limb segment of the leg. The Z''_i-axis is directed along the lower limb segment into the supporting surface. The Y''_i-axis is parallel to the Y_{hi}- and Y'_i-axes, and the X''_i-axis is uniquely determined by maintaining a right hand coordinate system.
T_{Ib}	3×3 transformation matrix between the body fixed coordinate frame (X,Y,Z) and the inertial frame (X_I, Y_I, Z_I).
$(\psi_i, \theta_{hi}, \theta_{ki})$	Leg i joint angles. ψ_i is the azimuth angle. It is the rotation angle around the Z_{hi}-axis (i.e. it is the angle between the leg plane and the body longitudinal axis, X). θ_{hi} is the hip elevation angle. It is the rotation angle around the Y'_i-axis (i.e. it is the angle between the body plane, XY, and the upper limb segment). θ_{ki} is the knee elevation angle. It is the rotation angle around the Y"-axis (i.e. it is the angle between the lower limb and a perpendicular to the upper limb).
$(\dot{\psi}_i, \dot{\theta}_{hi}, \dot{\theta}_{ki})$	Leg i joint angle rates.

λ_x	Longitudinal Stride Length. This is the longitudinal distance by which the robot body is translated in one complete locomotion cycle.
λ_y	Lateral Stride Length. This is the lateral distance by which the robot is translated in one complete locomotion cycle.
λ_ψ	Angular Displacement. This is the amount by which the robot body is rotated in one complete locomotion cycle.
τ	Gait Period. This is the time required for one complete locomotion cycle of a gait.
β_i	Duty factor for leg i. This is the relative amount of time spent on the ground by leg i during one locomotion cycle (fraction of the locomotion cycle).
ϕ_i	Relative phase for leg i. This is the amount of time by which leg i lags behind that of leg 1, expressed as a fraction of the time required to complete one locomotion cycle.
$(X_E, Y_E, Z_E)_i$	The components of the position vector of the foot of leg i, expressed in the inertial frame.
T_{bh}	3 × 3 transformation matrix from the body fixed coordinate frame (X,Y,Z) to the hip local coordinate frame $(X',Y',Z')_i$.
(a_i, b_i, c_i)	The coordinates of the hip socket for leg i as expressed in the body fixed frame (X,Y,Z).
l_1	The length of the upper limb segment.
l_2	The length of the lower limb segment.
$(\dot{X}_d, \dot{Y}_d, \dot{Z}_d)$	Components of the desired translational velocity of the center of gravity of the robot body.
L_i	An indicator of foot i position status, such that L_i=1 means that foot i is on the ground, L_i=0 means that foot i is off the ground.
$\hat{Z}_{iE}$	The predicted value of Z_{iE} for a given (X_{iE}, Y_{iE}). This is the regression surface value of Z_{iE} on variables X_{iE} and Y_{iE} (least square regression is used).
$(\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3, \hat{a}_4, \hat{a}_5)$	Least Square regression parameters.
$\vec{N}$	A unit vector normal to the desired body orientation, and expressed in the inertial coordinate frame.
f_k^{n+1}	is the reaction force from the ground to the tip of leg k expressed in the inertial coordinate frame. Its components are $(f_{kx}^{n+1} f_{ky}^{n+1} f_{kz}^{n+1})^T$. There are four 3 × 1 such vectors.
g_k^{n+1}	is the reaction torque from the ground to the tip of leg k expressed in the inertial coordinate frame. Its components are $(g_{kx}^{n+1}, g_{ky}^{n+1}, g_{kz}^{n+1})^T$. There are four 3 × 1 such vectors.
w^k	is the reaction force and torque from the ground to the tip of leg k. It consists of two components, the force and torque (f_k^{n+1}andg_k^{n+1}). There are four 6 × 1 such vectors.

F^0	is the net force acting on the robot body, expressed in the inertial frame. Its components are $(F_x^0, F_y^0, F_z^0)^T$. There is one 3×1 such vector.
G^0	is the net torque acting on the robot body, expressed in the inertial frame. Its components are $(G_x^0, G_y^0, G_z^0)^T$. There is one 3×1 such vector.
w^0	is the net force and torque acting on the body by the legs. It consists of the two components F^0 and G^0. There is one 6×1 such vector.
F_k^i	is the net force acting on link i of leg k. There are eight 3×1 such vectors.
G_k^i	is the net torque acting on link i of leg k. There are eight 3×1 such vectors.
m_r	is the mass of link r.
a_G	is the acceleration of gravity, expressed in the inertial frame.
f_E^r	is an external force, expressed in the inertial frame and acts on link r at point P_E^r from the origin of the proximal hinge of link r frame.
P_E^r	is the vector from the proximal hinge of link r to the point of application of the external force f_E^r to link r.
g_E^r	is an external torque expressed in the inertial frame and acts on link r.
a^r	is the linear acceleration of the proximal hinge of link r, expressed in the parent link coordinate frame.
r^r	is the linear acceleration of the center of mass of link r.
ω^r	is the angular velocity of link r.
$\dot{\omega}^r$	is the time derivative of ω^r (The derivative does not result in the components of the angular acceleration vector, since the derivative is applied to the components of the angular velocity represented in a moving frame.)
c^r	is the vector from the proximal hinge to the center of mass of link r.
f^r	is the force which link r exerts on its parent at the proximal hinge.
g^r	is the torque which link r exerts on its parent at the proximal hinge.
l^s	is the vector from the proximal hinge of link r to the proximal hinge of child s of link r.
R^r	is the transformation matrix that converts vector representations in the frame of link r to representations in the frame of the parent link.
R_I^r	is the transformation matrix that converts vector representations in the frame of link r to representations in the inertial frame.
J^r	is the 3×3 moment of inertia matrix of link r about its proximal hinge.
S_r	is the set of all links having link r as parent.
K^r, M^r	3×3 Armstrong recursive coefficient matrices (associated with link r). They are slowly varying variables used in the solution of the direct dynamics problem.
d^r, f''^r	3×1 Armstrong recursive coefficient vectors (associated with link r). They are used in the solution of the direct dynamics problem.
V^s	is the linear velocity of the center of mass of link s with respect to the base coordinate (r^s is its acceleration).

V^s	is the linear velocity of the frame of link s with respect to the base coordinate (a^r is its acceleration).
N^r	is the applied torque from various sources on link r expressed in its local frame.
F^r	is the net force acting on link r, expressed in its local frame.
z_r	is joint r axis of rotation and is located between link r and link s.
q_s	is the angle between link s and its parent (link r) measured about the axis z_r in the right sense. It is also known as "Joint coordinate" of link s.
$\dot{q}_s$	is the rate of rotation change between link r and s (i.e. the velocity of link s with respect to link r).
$\ddot{q}_s$	is the acceleration of link s with respect to its parent (link r).
g	is the driving torque vector (n × 1) for n DOFs system.
$H(q)$	is the variable inertial matrix expressed in the inertial frame.
$e(q,\dot{q})$	is the centrifugal and Coriolis forces. It is an n × n matrix that is dependent on joint positions and internal velocities.
$G(q)$	is the gravitational force vector.
$C(q)$	is 6 × n Jacobian matrix, specifying the torques (forces) created at each joint due to external forces and moments exerted on link n (the tip). $C(q)^T$ indicates the transpose.
W	is 6 × 1 vector of external moments and forces exerted on link n (the tip). The first three components are g^{n+1} and the last three components are f^{n+1}.
A_k	is a 6 × 6 coefficient matrix for the kth leg and is function of the present position and orientation of the leg.
B_k	is a 6 × 1 coefficient vector for the kth leg and accounts for the inertial and gravitational acceleration forces of the members of the leg.
m	is the number of legs that are in contact with the ground.
a^0	is the initial linear acceleration of the robot body expressed in the inertial frame.
$(J_{xx0}, J_{yy0}, J_{zz0})$	are the principal moments of inertia of the body, expressed in a coordinate system aligned with the principal axes of the body.
τ_{ik}	is the ith joint actuator torque in leg k.
C_k	is a 3 × 6 coefficient matrix for the kth leg and is function of the present leg position and orientation.
D_k	is a 3 × 1 coefficient vector for the kth leg. It accounts for the inertial and gravitational acceleration forces of the members of the leg.
f_k^N	is the normal component of the reaction force of the kth leg onto the ground.
$\dot{\omega}_i$	is a diagonal 3 × 3 matrix of the rate of change of leg i's angles $(\dot{\psi}_i, \dot{\theta}_{hi}, \dot{\theta}_{ki})$.

ξ	is a weighting coefficient.
$J_i(k)$	is the discrete state of joint i. k=1,2,....,l means that joint i can take l different discrete configurations.
S	Situation Space.
A	Action Space.
S^E, S^R	the situation space of the human expert and Robot skill description.
A^E, A^R	the action space of the human expert and Robot skill description.
F_S^E, F_S^R	Set of features defining the situation spaces S^E, S^R.
F_A^E, F_A^R	Set of features defining the action spaces A^E, A^R.

CHAPTER 1

INTRODUCTION

1.1. GENERAL BACKGROUND

Skill acquisition is one form of learning. Cognitive psychologists have pointed out that long after people are taught how to do a motion task, such as dance, play music, or do sport activities, their performance on that task continues to improve through practice [Nor80]. It appears that, although people can easily understand verbal instructions on how to perform a task, much work remains to be done to turn that verbal knowledge into *efficient* mental or muscular operations. Researchers in artificial intelligence and cognitive psychology have sought to understand the kinds of knowledge that are needed to perform skillfully. The processes by which people acquire this knowledge through practice are little understood [CoF81]. For example, how do highly skilled performers in dance, music, or sport make their actions appear to be so simple and easy with incredible smoothness, style, and grace? How is it that these performers can achieve such mastery accomplishments, while beginners in a similar task, are clumsy, inept, and highly unskilled?

A great deal about such phenomena can be revealed by an important aspect of motor control, the study of how movements are learned. Thus, in addition to the study of how movements are produced (or controlled) by the motor control system (as is done in control and mechanical engineering research), one should also study how movements are produced differently as a result of practice and experience. Understanding how motor skills are learned is the major concern of a field of study in psychology called motor learning.

Psychologists believe that motor skill development in children takes place over months, paralleled with their trials to control their limbs properly. During such development, errors are corrected iteratively in

the next instances of the movements. Through the ability to remember previous experiences of solving a specific or analogous problem, the child's motor system has the ability to learn and improve its performance. Subsequently, such mastered movements are called upon as motor skills. Learning the grasping of objects provides a good example of such a skill acquisition process. The whole hand has over two dozen degrees of freedom, yet the grasping motion, which involves many small muscles, is performed rapidly and without any apparent conscious intervention by adults. It takes an effort of will by a child to do the same grasping. This has led to the conclusion that child motor control systems are inherently goal-directed learners which acquire motor skills as they solve problems involving the coordination and control of their complex articulated bodies. In other words, both motor control and motor learning are parallel activities in child motor control systems [SZH79] .

For years, researchers of articulated bodies have dealt with motor control and motor learning issues separately. Motor control researchers are involved in setting up models for motion control (e.g. Open-loop, Closed-loop, Mass-spring, and Impulse-time) and solving mathematical equations for specifying motion using kinematics and dynamics. Such groups of researchers include animators in computer graphics, robotics researchers in control and mechanical engineering, and biomechanists in biomechanical sciences.

Motor learning researchers claim that humans do not solve mathematical equations in their motor systems when they assume control over their limbs during movements. Such researchers are using strategies of planning and learning instead of solving the mathematical equations of motion. They claim that humans and high animal species are able to gain control and maintain it by using both stored experience and feedback. Such groups of researchers include researchers of motor skills in behavioral psychology and biological scientists.

It is one cornerstone of the present work not to separate the study of motor control from the study of motor learning, as this artificial separation inhibits the understanding of both. This thesis proposes a locomotion control system for an autonomous four-legged articulated robot that draws a clear interfacing line between motor control and motor learning, similar to the one in child motor control systems [SZH79]. In the proposed system, the control of motor skills is parallel to the development of an intelligent body of

locomotion knowledge gained through practice and experience (the development of motor capabilities). This involves merging techniques from the important field of machine learning in artificial intelligence with efficient motion dynamics computations. On one hand, a knowledge-based motor control component would deal with symbolic data, but it is not called upon to perform efficient dynamics numerical computations. On the other hand, a dynamics-based motor control component will face the need to deal with large volumes of numerical computations but will lack symbolic reasoning mechanisms. Such a merge will combine decision making, judgement and reasoning with efficient numerical calculations.

Initially the articulated robot uses its dynamics-based component to control its movements. Simultaneously the development of motor skills in the form of production rules takes place through the calibration of the knowledge-based component by the numerical-based component. Then the knowledge-based component gradually goes through a natural "growing" process from being apprentice to the numerical component to being assistant and finally to being partner to it (i.e, it can take over the control of the robot actions without relying on the numerical component). This take-over of control will take place gradually and incrementally. Consequently, the robot would be able to control and coordinate its motion using the knowledge-based component instead of solving the mathematical equations of motion. Such a gradually growing synthetic "nervous system" seems to be essential for future highly adaptable articulated robots.

The work in this thesis is concerned with the modeling of a four-legged robot and with the development and implementation of algorithms for both its numerical and knowledge-based control components. The contributions of this thesis may be broken down into three general areas of the control problem:

(1) Effective control of the four-legged robot implies the need for algorithms which permit the synthesis of gaits suitable for realization of the desired robot trajectory, taking into account the constraints imposed by terrain conditions. Such algorithms have been developed and implemented for automatic leg positioning and automatic body height, pitch, and roll regulation over undulating terrain. These algorithms are called the joint coordination control algorithms and involve several kinematic control

problems.

(2) Robot locomotion involves more than just solving the joint coordination control problem kinematically, it also involves the control of forces and torques at the joints (the dynamics of motion). Failure to incorporate the dynamics of the robot motion may result in the unrealistic appearance of movements and/or the "binding effect" of its legs (one leg dragging another). To incorporate dynamics, the robot links are modeled as masses connected by joints and acting under the influence of external and internal forces and torques such as gravity, bendings, and flexings. The relationship between these forces and torques and the robot's motion is expressed as the "dynamic equations of motion". By using these equations one need not specify movements at each degree of freedom or worry about interdependence of body links because a force or torque on one link will affect other links to which it is attached. Consequently, one can control the robot movements by controlling such forces and torques. The development of algorithms for the numerical dynamics-based control component involved setting and solving a closed chain inverse dynamics problem, a linear programming problem, and a direct dynamics problem.

(3) Because of the complexity of the dynamics equations, the numerical dynamics-based control component has two problems. First, dynamic analysis of motion is computationally expensive and can not provide real-time robot motion control. Second, the dynamics equations are solved by using numerical methods, and when many degrees of freedom are involved numerical instability problems can arise. The knowledge-based control component solved these problems by developing production rules that regulated the robot's locomotion behavior. These rules go through modifications and enhancements as a result of practice and experience (through what we call "dynamic regression"). This idea of developing motor skills in the form of rules is drawn from machine learning research in artificial intelligence. The knowledge-based control component starts with general inference rules and learning techniques and gradually acquires complex skills through interaction with a human teacher.[1] The human supervises the knowledge-based component when it takes charge of the control

[1]We view our first version of this hybrid numerical/Knowledge-based control system as human-machine mix where artificial intelligence and tele-operated robotics are integrated into a highly reliable system. This is in order to minimize the technological risks

function (the teacher decides when). The role of the human is to perceive and redefine continually what needs to be done on the basis of what the knowledge-based component has learned, to take advantage of opportunities, to solve unforeseen problems and to save a mission (occasionally).

In summary, the objective of this thesis, then, is to design and implement a hybrid numerical/knowledge-based locomotion control system for a four-legged articulated robot that is capable of synthesizing and learning coordinated movements and improving its performance as a result of practice and experience. The produced prototype would allow us to examine the effectiveness of merging methodologies of computer numerical control (mathematical rigor of approaches taken in robotics, and biomechanics) with methodologies of motor learning (knowledge-based processing as studied in artificial intelligence and behavioral psychology). The expected result is that both methodologies can act together to provide an autonomous[2] locomotion control system that can improve its performance, perhaps through what might be called "motion understanding".

As possible areas of application for the proposed robot, it is envisioned that in the ensuing years, such articulated robots will perform tasks such as data collection in the forbidding regions of distant planets, mobile exploration on the ocean floor, duties in the arctic, fire fighting and explosive ordinance disposal. They might also be useful in constructing and maintaining structures in space such as solar arrays for space stations, large multibeam antennas, and space factories where the robot's legs could function as arms for object manipulation [MoA88a] [MoA88b].[3] These tasks will be open to the proposed robot because of its unique terrain adaptability, judgement and decision making. The emphasis in all these applications is two-fold: 1) acquiring high agility through a learning process, as might be required to quickly move from one point to another on a structure under construction, or needing repairs in an emergency; 2) enabling the robot to solve, "on-board" and in real-time, complex problems of manipulator dynamics and locomotion control.

---

of having, for example, a completely autonomous robot, that is learning by doing, on a space station. Meanwhile, a more autonomous control system in which the teacher has been replaced by experimentations is described in Section 6.5 (this is also related to the issue of supervised and non-supervised learning).

[2]See previous footnote.

[3]On a space structure one can't have the robot bumbling around while it learns by doing how to move. In [MoA88b] a ground simulated robot executes missions a few on-board camera images ahead of the space realistic robot. Accordingly, the simulated robot is used as an experimental planner by the realistic robot, to detect what would have happened if the realistic robot actually executed a mission using its knowledge-based controller.

It is important to distinguish here between animation and the kind of animated simulations that the experimental work of this dissertation describes. What we are dealing with here is what is called "task level simulation". This is a term that is shared with the robotics world, where the problem is to control robotic agents by specifying a set of events and some constraints on the behavior of the agent, and letting that agent fill in the details as necessary. So, we are not interested in constraining simulated agents to do what we want them to do. Rather, we want to let them interact with the environment and adapt their goal-directed motion accordingly.

1.2. ORGANIZATION

Chapter 1 of this thesis introduces the proposed locomotion control system with the various control problems encountered. Chapter 2 is a review of some of the work in computer graphics, robotics, motor psychology, and biomechanics, whose themes share common ground with the research in this thesis. Also the aims of these projects and their achievements are compared to those of the proposed locomotion control system. Chapter 3 presents an overview of the concepts and ideas of the experimental system. The system architecture is described, then the kinematics of motion and its algorithms are developed. The dynamics-based control component (simply called the numerical controller) is described in Chapter 4, followed by a description of the knowledge-based control component (simply called the learning controller) in Chapter 5. Chapter 6 gives some details for the implemented system. The results of the system's performance evaluation and extensions for further work are also included. Finally, a list of references and an appendix appear at the end of the thesis.

CHAPTER 2

RELATIONSHIP TO OTHER WORK

2.1. INTRODUCTION

The locomotion of articulated bodies has been the area of interest of at least four groups of researchers:

(1) Animators in computer graphics: Computer animators have simulated locomotive movements of articulated figures through the rapid display of successive images. Each image represents a sequential moment in time of the movement. When these images are displayed fast enough, the human eye interprets them as continuous motion. The main objective of computer animators is to produce realistic movements on the screen for the purpose of communicating ideas and thoughts.

(2) Legged motion technologists in control and mechanical engineering: A specific area in robotic research which deals with "legged technology" is motivated by the need for vehicles that can travel in difficult terrain. Legs promise improved vehicular mobility where the ground is soft, steep, or slippery (such as in the Arctic regions and in jungles). Other reasons for building legged machines are for defense and outer space applications.

(3) Motor skill researchers in behavioral psychology: The motivation of this group is to understand how animals, humans, and insects use their legs for locomotion. These psychologists are trying to understand how motor skills are stored in memory, how they are retrieved, and how they are produced differently as a result of practice and experience.

(4) Biomechanists in biomechanical research: Using mechanical principles, these researchers study animal bodies in motion and at rest. Their goal is a model which, given nervous signals, would provide muscular forces and, given muscular and other internal and external forces, would provide the positions, velocities, and accelerations of each limb.

In this chapter, some of the work done by each of these groups, with special emphasis on locomotion projects as a particular type of articulated body movements, will be reviewed. The review is meant to clarify how researchers studying the locomotion of articulated bodies have dealt with motor control and motor learning as separate issues (see chapter 1).

2.2. APPROACHES TO ANIMATING ARTICULATED BODY LOCOMOTION

In computer graphics there has been interest in the problem of animating articulated bodies since at least the late 1970's ([BuW76], [BaB78], [CCP80], [Doo82], [GiM85], [WiB85], [ArG85], [IsC87]). Since then, the dominant approach to describing articulated body animation has been kinematics (based on positions and angles varying over time). More recently, however, researchers have been shifting to dynamics (objects with mass moving under the influence of forces and torques) to improve the power of their animation tools.

Currently, dynamic analysis has been used in the animation of articulated bodies in very few research projects. The mechanical ants from the New York Institute of Technology by Dick Lundin included some dynamic analysis [Lun86]. The walking creature modeled in the PODA animation system by Girard and Maciejewski at Ohio State University [GiM85] included simple dynamics of the trunk. Jane Patricia Wilhelms [WiB85], at the University of California, and Armstrong and Green [ArG85] at the University of Alberta have also independently modeled articulated bodies using more complete versions of dynamic analysis. Recently, Paul Isaacs and Michael Cohen [IsC87] at Cornell University implemented DYNAMO (DYNAmic MOtion system), a system for the dynamic simulation of linked figures. Jane Wilhelms and David Forsey of Waterloo [WiF88] have implemented MANIKIN for interactive manipulation of articulated figures using dynamics. Also Armin Bruderlin and Thomas Calvert of Simon Fraser [BrC89] have implemented KLAW, a goal-directed system for the dynamic animation of Bipedal locomotion.

However, these dynamic-based systems have dealt only with experimental movements in which the articulated figures are passive elements in the environment. For example, in [WiB85] simple motion like letting a hand fall under the influence of gravity, raising the arm using controlling forces and torques, and a

few floor exercises for a 24-degree-of-freedom (dof) human figure, such as raising the knee up to the chest, etc. were tried. Similarly, in [ArG85], animating a finger to tap on a table and a twelve link human figure in a diving motion (its motion was under the influence of external downward and forward forces on the head and arms) were tried. Also, in [IsC87] five experimental movements were conducted: a series of hanging chains, a tree blown by wind, a whip, a person on a swing, and an arm catching and throwing a ball.

The common ground among these experimental movements is the testing of the influence of the joints' forces and torques on the mechanism's motion. However, none of these systems have dealt with any real coordinated task to perform a rhythmic action such as locomotion (except for Bruderlin's work). Locomotion is the act of moving from place to place. It is obvious that the application of a force to a stationary rigid body produces motion by overcoming inertia and by overcoming such restraining forces as friction and viscosity of the environment. Without the application of a force or forces, no voluntary movement, in this case locomotion, can take place. In articulated bodies, however, the applied forces cause relative motion between links of the body, which then produces reaction forces and torques from the ground. To achieve stable locomotion in legged figures, there must be a component of the reaction forces that supports the body against the pull of gravity. The inertia of the body resists movement and requires a force proportional to the mass to produce a change in the velocity of each particle. The horizontal component of the ground reaction is the force that produces the articulated body acceleration change during locomotion (see Figure 2.1).

In order to clarify the terminology here, one should define the difference between open/closed kinematic chain systems, and direct/inverse dynamic problems. In open chain kinematic systems, one end of the linkage structure chain in fixed and the rest of the chain is free to move. In closed chain, more than one endpoint is fixed in the linkage structure. The problem of determining the motion of a linkage system from a set of applied forces and torques is called the "direct dynamics problem", whereas the problem of determining the forces and torques required to produce a prescribed motion is called the "inverse dynamics problem".

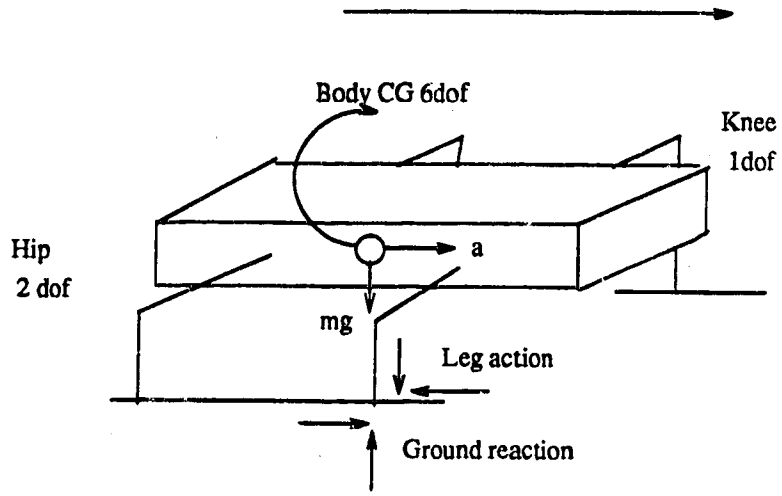

Figure 2.1 Ground Reaction Forces

Animation of legged figure locomotion using dynamics is a very difficult problem. One of its principal difficulties is the lack of an entirely satisfactory solution to the limb coordination control problem. In such problems, the force and torque assignment equations relating joint forces and torques to the desired motion trajectory result in fewer equations than unknown torque and force variables. Because of the underspecified nature of the problem, an infinite number of solutions are possible, and some method must be used to choose among these [SeA75]. Consequently, none of the existing animation systems has dealt with articulated body locomotion dynamically.

Historically, however, the difficulty of legged figure locomotion was alleviated by animating the movements kinematically. In kinematic locomotive animation ([BaB78], [Her78], [CCP80], [Doo82], [Fet82], [GiM82], [KoB82], [KBB82], [Zel82], [Stu84], [StB85], [GiM85], [Kro86]) the motion of the links of the articulated body are planned out directly by the animator, and the computer animation system plays no part in ensuring that the action is reasonable.

Girard and Maciejewski in their PODA animation system [GiM85] utilized a kinematic model of a four-legged creature which allowed animators to control the complex relationships between the motion of

the body and the coordination of its legs. Legs were moved along prespecified trajectories placed exactly at desired footholds and held in place as the body passed over them. A pseudo-inverse Jacobian matrix which describes how small changes in local joint positions are caused by small changes in the tip segment's world space position and orientation was used in computing the joint angles from the positions of the ends of the legs (this is called the inverse kinematic problem). This has helped to relieve the tedious job of specifying each degree of freedom separately.

Girard and Maciejewski included simple dynamics of the trunk but did not incorporate the dynamics of the motion of the legs. The problem that might result from ignoring the leg force control is that it is possible that the legs will tend to "fight" each other, while their net action is the desired body trajectory. For example, unless the leg forces are controlled, the legs on each side of the body can develop equal and opposing lateral forces such that the body, in equilibrium, will remain in equilibrium with an unnecessary expenditure of energy. Also, when one of these legs is lifted from the ground, undesirable vibration of the body may occur. This might be called the "opposing actuator" problem. Moreover, ignoring the force control issue might also result in one leg dragging another across the supporting surface ("binding effect").

While Girard and Maciejewski used linear approximation to update the inverse Jacobian matrices, Korein and Badler [KoB82] proposed a much faster method called "reach hierarchy" to solve the inverse kinematic problem. The method applies the rule "move each joint the smallest amount between keyframes that will allow the end effector to reach the goal", by assigning angle values to the joints outwardly from the inner links to the end effector. The method is faster than calculating the inverse Jacobian; however, it does not account for human or animal constraints of coordinated movement or habits, and so may be difficult to apply to realistic locomotive tasks.

Another locomotive kinematic model was proposed by Zeltzer [Zel82]. The model was based on the anatomy of the human body and characteristics of its locomotion. Motion was achieved by a hierarchy of motor programs. The low level motor programs controlled the joint angles for a fixed set of joints. These motor programs were controlled by middle level motor programs. The middle level motor programs would start and stop the low level programs based on the current state of the model (joint angles, center of mass,

support, etc.). Both the low level and middle level motor programs were modeled as finite automata. At the top level of the hierarchy complete skill programs used finite automata to activate sequences of the middle level motor programs.

Norman Badler's group at the University of Pennsylvania [BMW87] have developed a system for positioning articulated bodies using kinematic constraints. This refers to the ability to restrain part of the body relative to worldspace allowing the animator to position the figure without, for example, worrying if the feet will penetrate the floor. The animator is allowed to designate multiple constraints and rate their importance, a useful feature when they cannot all be resolved. Problems with kinematic constraints involve choosing a desirable solution from among many possible body positions.

Don Herbison-Evans of Sydney University in Australia has developed a movement simulation program called NUDES (Numerical Utility Displaying Ellipsoid Solids) [Her78] [Her80]. He used a dance notation called Benesh (a notation particularly appropriate for description of the stylized movement of classical ballet). Human-like figures were represented as "poly-ellipsoids", the limbs being interpenetrating ellipsoids, in order to provide real-time displays of dance.

Savage and Officer of the University of Waterloo have developed CHOREO, another interactive computer program for display of dance [SaO78]. Interactive graphics editors were available for input of the dance using either of two dance notations: CHOREO uses Massine notation and CHOREO-L uses Labanotation. The interesting thing about CHOREO is that its dance notation editor uses some sophisticated features of the "vi" and "emacs" text editors to create a convenient environment.

Another system that uses Labanotation in the kinematic specifications of dancer animation is the dance system developed at Simon Fraser University [CCP82]. An interactive graphics editor allows a set of Labanotation symbols to produce the scores. The body was defined in terms of limbs, joints, and connectivity. The program included several possible types of input, a hierarchical control model, and choice of output.

Another popular method of specifying locomotion kinematically is to take live data from, for example, a walker or a runner and to use the data to drive a graphics display. This is called "rotoscoping"

and uses two-dimensional data from a motion picture film or three-dimensional data from goniometers or video scanning equipment (e.g, [GiM82]). Perfectly life-like motion can be obtained this way, but at the cost of poor flexibility after the data is collected.

Another technique for describing the motion kinematically uses some form of keyframing (e.g, [Stu84]). Here the animator interactively positions the model on the screen by modifying 3D transformation matrices located at the joints. These are used to control the positions of the body below the joints. The BBOP system at NYIT [Stu84] uses a three axis joy stick to position a human figure by changing the joints' transformation matrices. A human-like figure was made to walk up or down the stairs by building one frame to show the figure's foot in the air and another to show the foot on the stair. Then the program uses cubic interpolation to fill in all the frames in between.

According to this last technique, a keyframe is necessary wherever any link of the body has a key action. This leads to a large number of keyframes. A better keyframe technique, called event-driven keyframe animation [Gom84], was introduced by Julian Gomez et al. at Ohio State University. The Ohio group introduced the TWIXT animation system that thinks of each link in the body as pursuing its own course of action. Therefore, the animator creates tracks of action and places links on them. He/she specifies what the link is doing at various points in time along those tracks. TWIXT, for example, defines several tracks: a position track, rotation track, velocity track, and attachment track. For intermediate times, the system mathematically interpolates those track parameters to construct in-between frames. To build a frame, the system evaluates the activity on every track of every link. Once the display parameters for a link (position, rotation, velocity, etc.) have been determined, its transformation matrix is built. Thus the frame can be considered as the union of activity in all tracks.

Gomez's approach is sometimes called "layering", whereby the animator is allowed to specify different motions independently and have the system take care of putting together these layers of motion. Analogies can be drawn from cartoon animation where a frame is built up of a number of cels lying on top of each other. In layering, however, the layers are not pieces of picture, but pieces of motion.

In fact, manipulating articulated bodies using keyframing is a laborious task because of the many degrees of freedom, the interaction between segments, and the difficulty of finding and/or constraining local joint positions.

Although all of these previous systems animate the locomotion of articulated bodies, they do not take into account the mass and inertia of the body in motion (the dynamics of motion). Thus, animations which are produced often have an unrealistic appearance. The links do not seem to have weight or mass, and thus speeds of movements are inaccurately represented. Furthermore, most of these systems can react to the environment only in restricted ways. For example, in most of the previous systems the articulated figures could walk over uneven terrain but could not respond to someone pushing them and could not even conserve the simplest environmental constraints (such as "foot should not go through the ground", "leg joints cannot go beyond their natural limits", etc.). Another problem is that kinematic control is typically quite low-level; users deal with each degree of freedom independently despite the fact that, even kinematically, they are interdependent. Therefore the amount of information the user must provide to produce a specific motion is enormous. The study of biological literature is required in order to acquire such information. Finally, restricting the choice of positions and velocities to those actually achievable by a real articulated body is very difficult to accomplish kinematically.

For all of these reasons, Sturman [Stu87] claims that all the kinematically based articulated figures animation systems (including his) should be classified for entertainment because they are not concerned with realistic simulations. Their goal is communication: if a motion sequence is communicated by ignoring realistic simulation, then animators have no hesitation in using it. Not so for the scientific and engineering community. Realism is important, purposefulness and expressivity less so. In scientific simulation, the purpose is to describe what happens, whatever happens [Stu87].

Dynamics takes into account mass and inertia as well as the various forces acting on the body. Animation, therefore, comes out realistically at the expense of much (slow) computation per frame. Links move correctly and appear to have weight and substance. In dynamics-based locomotion, only the ground reaction forces and torques on the feet need to be computed for the movements to be automatically

produced according to the laws of physics and mechanics. This allows the dynamics-based animation systems to predict the effects of such important influences as gravity, collisions, and applied external forces. Consequently dynamics-animations are "natural" in that they mimic how objects move in the real world, and constrain motion to be realistic (for the reality being modeled). Kinematic motion specifications have no way of incorporating such effects.

However, dynamics-based locomotion methods do have their own problems too. One is the amount of computer time required to compute the motion per frame. Another is that the dynamic equations are solved by using numerical methods, and when many degrees of freedom are involved numerical instability problems can arise. Moreover, in such systems the control of the figures' motion is shifted from the animator to the underlying physics of the environment. If one were to return control to the animators, they would be left with a whole new problem of determining the torques and forces required to produce a particular motion sequence. These problems are addressed in Chapter 4.

2.3. CONTROL AND MECHANICAL ENGINEERING RESEARCH ON WALKING MACHINES

The motivation for building walking machines comes primarily from the need for vehicles that can travel in difficult terrain. Legs also promise improved vehicular mobility where the ground is soft, steep, or slippery (such as in Arctic regions and jungles). The United States army showed interest in utilizing a hopping tank that fires while it hops. It is not an easy target and has great mobility in difficult terrain battle fields [Rai86]. NASA also showed interest in an insectlike robot that can construct large structures in space such as solar space stations, large multibeam antennas such as reflectors and lenses, and space factories [NAS78]. According to NASA, legged robots are expected to have an important role in the construction, inspection, and maintenance of these structures in view of the alternative astronomical cost of supporting men as construction workers in space [NAS78] [Hee73]. Another motivation for building walking machines is to understand better how animals, humans, and insects use their legs; of particular value is the opportunity to formulate and test precise locomotion theories (see section 2.4) that can guide biological

research [KaS70].

Currently, more than twenty legged systems have been built in the United States and Japan. Table 2.1 shows some of the milestones in legged systems technology since early 1960's [Rai86].

Early research in building walking machines originated at General Electric [LiM68]. Motion control of the GE walking machine proved to be very difficult, as the operator had to control simultaneously all four legs with his four limbs (master/slave control). This showed that even when the main motion of the machine was man-controlled, the intelligence of interacting limbs in this redundant system was definitely required for normal operation. The operator was aided by force reflecting servomechanisms which provided him with an indication of the interaction of the machine with the supporting terrain. Whenever the operator caused a machine leg to push on an obstacle, force feedback let the operator feel the obstacle as though it were his or her arm or leg doing the pushing. Despite its dependence on a well-trained human for control, this walking machine was a landmark in legged technology. This machine first walked in 1968 and later exhibited a significant ability to climb obstacles and to traverse difficult terrain.

An alternative to human control of legged machines became feasible in the 1970's: the use of computer for locomotion control. Robert McGhee's group at Ohio State University was the first to use this approach successfully in 1977 [McG83]. They built an insectlike hexapod that could walk with a number of standard gaits, turn, crab, and negotiate simple obstacles. The computer's primary task was to solve kinematic equations in order to coordinate the eighteen electric motors driving the legs. This coordination ensured that the machine's center of mass stayed over the polygon of support provided by the feet while allowing the legs to cycle through a gait. The machine traveled quite slowly, covering several meters per minute. Force and visual sensing provided a measure of terrain accommodation in later developments [McG80] [OTM84]. The hexapod provided McGhee with an excellent opportunity to pursue his earlier theoretical findings on the combinations and selection of gait [McK72]. The group at Ohio is currently building a much larger hexapod, about 3 tons, that is intended to operate on rough terrain with a high

Mosher68	GE quadruped machine under control of human driver- 4 legs- hydraulic actuators- 1968.
McGhee77	A Computer Coordinated Hexapod Walking Machine- 6 legs- electronic actuators- 1974.
Gurfinkel77	Hybrid Computer (digital/ analog) controls a Hexapod- 6 legs- electronic actuators- 1977.
McMahon&Greene77	Human runners set new speed records on tuned track at Harvard. Its compliance was adjusted to match the mechanics of the human leg.
Hirose80	Quadruped machine climbs stairs and over obstacles using simple sensors. The leg mechanism simplifies control - 4 legs- hydraulic actuators.
Kato80	Hydraulic biped walks with quasi-dynamic gait.
Matsuoka80	Machine that balanced in the plane while hopping on one leg.
Miura81	Walking biped balances actively in three dimensional space- 2 legs- electronic actuators.
Raibert82	One-legged hopping machine balancing in 3D. Quadruped runs with two-legged gaits and can change gait while running.
Sutherland83	Hexapod carries human rider- Computer, hydraulics, and human share computing task.
Odetics83	Self-contained hexapod- "lives" and moves in the back of a pickup truck.

Table 2.1 Characteristics and Capabilities of Existing Legged Machines

degree of autonomy [WVP84].

Gurfinkel and his co-workers in the USSR built a machine with characteristics and performance quite similar to McGhee's at about the same time [GGS81]. It used a hybrid computer for control, with heavy use of analog computations for low level functions.

Okhotsimski, Platonov et al. [OkP73] [OGA74] , also in the USSR, have investigated the movement of a walking machine over uneven terrain. The control computer of a six-legged vehicle was supplied with

a set of "standpoints" (foot placement points) that would support the vehicle in locomotion. The task of the computer was to control the timing of the liftings and placements (tracking schedule) from these standpoints while maximizing the stability reserve (margin) of the moving vehicle.

To develop the algorithm for synthesizing the tracking schedule with a given movement of the body, two models were used. One included a two-degree of freedom leg and the other, a three-degree of freedom leg. Several algorithms were investigated for synthesizing the tracking schedule. One algorithm assumed that the vehicle had to move by the tripod gait. A calculation of the optimum transfer time from one tripod of supporting legs to the other tripod of supporting legs was made. Another algorithm was based on the relative timing between the transfer waves for the right and left row of legs. The final algorithms developed were used during the movement of the machine over surfaces with complex relief. In further work, Okhotsimski [OGA74] enhanced the algorithms to control the walking machine in climbing over obstacles. Algorithms for generation of standpoints were included along with consideration of the necessary terrain measurement system. Planning algorithms were designed which were able to generate standpoint sequences for a route comprising an arbitrary curve on a surface with small-scale roughness. Also, algorithms were designed for generating special irregular standpoint sequences in the case of overcoming obstacles.

All the previous machines have used kinematic control techniques without regard for the forces which produce the motion. The dynamics of legged locomotion has been incorporated into legged vehicle projects only recently. In the following subsections, two such state-of-the-art projects will be investigated in some detail. The first is an American project at Carnegie-Mellon University, and the second is a Japanese biped project at the University of Tokyo. The objective here is to demonstrate the fact that despite the goal of improved vehicular mobility in difficult terrain, and the incorporation of the dynamics of motion, legged vehicles have not yet proved themselves by moving out of the laboratory and into the field. These state-of-the-art machines still enjoy the smooth, flat ground of the laboratories.

2.3.1. THE HOPPING MACHINE AT CARNEGIE-MELLON UNIVERSITY

It was to study running in its simplest form that Marc Raibert and the group at CMU built a running machine that had just one leg ([Rai81], [Rai84], [RBM84], [Rai85]). It ran by hopping like a kangaroo, using a series of leaps. With one leg this machine drew attention to active balance and dynamics while postponing the difficult problems of coordinating the behavior of many legs.

The machine had two main parts, a body and a leg. The body provided the main structure that carried the actuators and instrumentation needed for the machine's operation. The leg could telescope to change length and was springy along the telescoping axis. Sensors measured the pitch angle of the body, the angle of the hip, the length of the leg, the tension in the leg spring, and contact with the ground. The first machine was constrained to operate in a plane, so it could move only up and down and fore and aft and rotate in the plane [Rai81]. For this machine, running and hopping are the same. Its motion control system considered separately the hopping motion, forward travel, and orientation ("posture") of the body. A brief description of each part of the control system is given below. The details of the individual control algorithms are not as important as the framework provided by the decomposition.

(a) The hopping motion:

This part of the control system excited the cyclic hopping motion that underlay running while regulating how high the machine hopped. The hopping motion was an oscillation governed by the mass of the body, the springiness of the leg, and gravity. During support, the body bounced on the springy leg, and during flight, the system traveled a ballistic trajectory. The control system delivered a vertical thrust with the leg during each support period to sustain the oscillation and to regulate its amplitude. Some of the energy needed for each hop was recovered by the leg spring from the previous hop.

(b) The forward motion:

This part of the control system regulated the forward running speed and acceleration. This was done by moving the leg to an appropriate forward position with respect to the body during the flight portion of each cycle. The position of the foot with respect to the body when landing had a strong influence on the behavior during the support period that followed. Depending on where the control system placed the foot,

the body continued to travel with the same forward speed, accelerated, or slowed down. To calculate a suitable forward position for the foot, the control system took account of the actual forward speed, the desired speed, and a simple model of the legged system's dynamics. It considered four different cases: when the machine is hopping in place, accelerating to run, running at a constant speed, and slowing to a stationary hop.

(c) The posture control:

The third part of the control system stabilized the pitch angle of the body to keep the body upright. Torques exerted between the body and leg about the hip accelerated the body about its pitch axis, provided that there was good traction between the foot and the ground. During the support period there was traction because the leg supported the load of the body. A linear servo operated on the hip actuator during each support period to restore the body to an upright posture.

The utility of this three-part control system was first constrained to machines that operated in a plane, but it was later generalized for controlling a three-dimensional, one-legged machine, a planar two-legged machine, and a quadruped [RaS83] [RBM84] [Rai85].

The generalization to three-dimensional machines was easy. The mechanisms needed to control the remaining extraplanar degrees of freedom were cast in a form that fitted into the original three-part framework. For instance, the algorithm for placing the foot to control forward speed became a vector calculation. One component of foot placement determined forward speed in the plane of motion, whereas the other component caused the plane to rotate about a vertical axis, permitting the control system to steer. A similar extension applied to body posture. The result was a three-dimensional three-part control system that was derived from the planar case with very little conceptual complication [RaS83].

For the biped generalization, that runs like a human with alternating periods of support and flight, the one-leg control algorithms were applied directly. Because the legs were used in alternation, only one leg was active at a time: only one leg was placed on the ground at a time, only one leg was thrust on the ground at a time, and only one leg exerted a torque on the body at a time [RBM84].

When there were several legs (in the case of quadruped), Raibert could not make the same biped generalization. However, he coordinated legs that shared support simultaneously, making them behave like a single equivalent leg. Several multi-legged gaits were mapped into virtual biped gaits. For example, the trotting quadruped was mapped into a virtual biped running with a one-foot gait [Rai85].

In summary, Raibert reduced the trotting quadruped to a biped, and reduced the biped to a one-legged machine. For the one-legged machine he decomposed the controller into the three primitive parts: hopping, forward speed, and posture.

2.3.2. THE JAPANESE BIPED LOCOMOTIVE ROBOT

The research group at the University of Tokyo have been working on the biped locomotion project "Biper" since 1979 [MiS84] [MSM85]. The group of the CMU hopping machine made the motion controller simpler by confining the motion to a plane. The Japanese designers of the Biper have realized this fact and decomposed the motion of their biped into two components in the sagittal and frontal planes. Attention has been paid to these two planes, since it is in these planes that the problems of balance occur. Moreover, the assumption that the motions in these planes are independent made the control system dynamically simpler.

The Biper project has produced several bipeds that have different leg configurations. All the Bipers are statically unstable, but can perform a dynamically stable walk with suitable control. Biper-1 and Biper-2 walk only sideways. Biper-3 is a stilt-type robot whose foot contacts occur at a point and which can walk sideways, backwards, and forwards. Biper-4's legs have the same degrees of freedom as human legs. In all cases, basically the same control method is applied. The control system is based on the idea that motion during the single-leg support phase can be approximated by the motion of an inverted pendulum. Accordingly, the control system deals with the dynamics of walking as a series of inverted pendulum motions with appropriate conditions of connection.

The inverted pendulum has been used to model the sagittal and the frontal plane dynamics when the biped is standing on one leg. It assumed that the foot does not slip and is large enough to sustain the

required torque without the Biper overturning. Let m be the mass of the pendulum, I_o its inertia about the ground pivot, l its length, and g be the acceleration of gravity. Then the equation of motion in terms of the (as yet unspecified) actuator torque M (that is needed to rotate the pendulum) and the angle θ (that the pendulum makes with the horizontal axis) is derived using the one-dimensional (angular) application of Newton's Law:

$$I_o \ddot{\theta} = M - mgl \cos\theta$$

This simple equation is the basis of the dynamic model of the Biper. Only in the Biper's multi-link structure is each link treated as a free body, with forces and moments applied at each joint. At each joint there is one actuator torque; the remaining forces and torques are due to reaction.

For example, Biper-3 is actuated with four torque motors located at the hip. Motors 1 and 2 cause the legs to move about the roll axis, and motors 3 and 4 move the legs about the pitch axis. Biper-3 walks when motor 1 generates an appropriate torque so that the left foot is detached from the ground for a while. Then, by applying torque from motor 3, stepping can be realized (the leg starts to swing). Applying the inverted pendulum equation to the free links would give the system's equations of motion.

The Biper controller used these equations of motion to calculate the amounts of torque which need to be applied by the motors in order to follow the legs' planned trajectories. The controller used a feedback cycle to compare the actual trajectories with the planned trajectories.

The work on the Biper project is still active at the University of Tokyo. Biper-5 is an improvement over Biper-4: it has all its apparatus, such as the computer, mounted on it and several contact sensors attached to its feet [MSM85].

Recently, McGeer [McG88] has implemented a "two-dimensional" biped. The biped has three legs; the outer legs are connected by a crossbar, and alternated like crutches with a broad-footed center leg. The feet were semicircular and had roughened rubber soles. During each step small motors lifted the swinging feet clear of the ground. McGeer's biped could be viewed as a generalization of the Bipers with a pair of coupled pendulums.

Currently, work on walking machines is continuing at General Electric, Stanford, and JPL. Projects are also underway at the Tokyo Institute of Technology and at the University of Tokyo. Researchers at Ohio State, Ressenlaer Polytechnic Institute (RPI), and CMU are also now working on new legged locomotion machines. In a recent development, Odetics, a small California-based firm, announced a six-legged robot at a press conference in March 1983. According to the press release, this robot, called a "functionoid", can lift several times its own weight and is stable when standing on only three of its legs. Its legs can be used as arms, and the machine can walk over obstacles. Odetics scientists claim to have solved the mathematics of walking. It is not clear from the press release to what extent the Odetics work is a scientific breakthrough, but further investigation is clearly warranted [And85].

In general, one can safely conclude that despite the goal of improved vehicular mobility in difficult terrain and the incorporation of the dynamics of motion, legged vehicles have not yet proved themselves by moving out of the laboratory and into the field. Although several researchers are actively working toward this goal, progress in the area of walking machine was predicted to be slow in the next few years [Mor83]. Perhaps this is because of the need to deal with navigational issues in control and the problems of stability of locomotion.

The four-legged articulated robot proposed in this research is intended to navigate a vaguely described environment containing obstacles, inclines, and rough terrain. Moreover, it can adapt its motion on the fly to accommodate any unpredicted obstacles in the environment.

2.4. MOTOR LEARNING-THE BEHAVIORAL VIEW

It is clear from the previous two sections that both computer animators' views and mechanical and control engineers' views of motion control have relied on solving the equations of motion (either the kinematic or the dynamic) and have virtually ignored the motor learning issue. Evidence shows that only solving the equations of motion can not explain phenomena such as "skilled performances":

(1) Highly skilled performers in dance, music, or sports make their movements appear to be so simple and easy, manifesting incredible smoothness, style, and grace; while beginners in a similar task, are

clumsy, inept, and highly unskilled. If both motor control systems are solving the same equations of motion, how can one explain the difference in performance?

(2) The grasping motion of the hand is done rapidly and without any apparent conscious intervention by adults, whereas it takes an effort of will by a child to do the same grasping. If both motor control systems are solving the same equations of motion, how can one explain the difference in performance?

It is important to indicate here the similarity between the human motor control system and the robotic motion control systems. This similarity has been demonstrated through the many simulation systems that were built to study human and animal motor control and were regularly used to model motion control systems for robots as well [CCP82]. This should be apparent since both are articulated structures that are capable of independent motion due to internal forces.

Investigating these phenomena is a major concern of a field of study in psychology called motor learning. Motor learning has been defined in a variety of ways by various researchers in the field. Three distinct characteristics which serve to define motor learning were first given in [ScW72]. First, motor learning is a process of acquiring the capability for producing skilled actions. That is, motor learning is the set of underlying events, occurrences, or changes that happen when people practice, allowing them to become skilled at some task. Second, motor learning is a direct result of practice or experience. Third, motor learning cannot (at our current level of knowledge) be measured directly, as the processes leading to changes in behavior are internal and are usually not available for direct examination. Rather, one must infer that learning (the process) occurred on the basis of the changes in behavior that can be observed. A synthesis of these three aspects produces the following definition of motor learning: "Motor learning is a set of processes associated with practice or experience leading to changes in skilled behavior" [SZH79].

In fact, psychology literature on motor learning presents a rather disjointed collection of facts largely devoid of unifying themes. Furthermore, the area of motor learning seems to have a "supermarket" quality: a little massed/distributed practice here, feedback there, stacks of reaction time, mental rehearsal, speed/accuracy, short-term memory, and other distinct topics of interest piled about in disarray.

Perhaps the most appropriate way to tackle the motor learning issue is to study motor learning theories. A motor learning theory is a set of theoretical assumptions which attempt to explain such things as how motor skills are acquired, the role of practice, performance variations, learning limits, and the types of cues which cause learning to take place.

In the following subsections, two of the most important modern motor learning theories are looked at.

ADAMS' THEORY

A theory dealing exclusively with motor learning, presented by Adams, generated enormous interest during the 1970s [Ada71] [Ada76]. In the following discussion some of the major theoretical propositions of the theory are presented.

Adams believed that all movements are made by comparing the ongoing feedback from the links during the motion to a reference of correctness that is learned during practice. He termed this reference of correctness the "perceptual trace".

For positioning movements, in which the individual must learn to locate his limb at a proper position in space, the perceptual trace represents the feedback quantities of the correct position. Therefore, minimizing the difference between the feedback received and that of the reference of correctness (the perceptual trace) means that the link is brought to the correct position by closed-loop feedback processes. In some of his writings [Ada71] [Ada76], Adams implies that the perceptual trace represents the path of the action toward the target as well as the target endpoint, with feedback being used to guide the movement along this proper trajectory.

Given the critical role of the perceptual trace in performance, how is the reference of correctness learned with practice? Adams' idea is quite simple: when the individual makes a positioning movement, feedback stimuli are produced that represent the particular locations of the limb in space. It is believed that these stimuli do in some way "leave a trace" in the central nervous system (hence the name perceptual trace). With repeated responses and with knowledge of the result (the information about the success of the movement trials), the individual comes closer and closer to the target on repeated trials; and on each of the

trials another trace is laid down so that eventually a kind of "collection" of traces develops.

Because of the knowledge of results, the learner is responding close to the target after only a few trials, for each trial provides feedback stimuli that tend to represent the correct movement. In turn, the collection of traces (perceptual trace) comes to represent the feedback quantities of the correct movement. On subsequent trials, the learner moves to that position in space for which the difference between the ongoing feedback produced and the weighted average of the collected perceptual trace is minimal. The difference between the weighted average perceptual trace and the feedback represents an error in the movement. The individual seeks to produce an action that results in minimal error on each trial. Since the perceptual trace is stronger with each trial, the errors in performance decrease with practice [Ada71].

In such a view, knowledge of the result does not produce learning directly. Rather, it creates the appropriate situation (i.e. being on target) so that the actual learning mechanisms (i.e. the feedback producing an increment in "strength" for the perceptual trace) can operate.

Some of the limitations of Adams' theory are: (1) It focuses almost entirely on slow, linear-positioning type of movements, which are not sufficiently representative of the many other kinds of skills such as locomotion. (2) It has a limitation in recognizing the role of open-loop processes in movement control (see Schmidt theory next). (3) Polit and Bizzi [PoB78] found that some organisms deprived of all sensory feedback from the limbs can respond skillfully, and they can even learn new actions. If the only mechanism for controlling skilled actions was the use of feedback in relation to a perceptual trace, then these individuals should not have been able to produce the actions they did. (4) Evidence against Adams' theory was also provided in the psychology literature by what is called "variability in practice" (a teaching technique in which the goal response to be made is systematically varied from trial to trial). Because the perceptual trace is the feedback representation of the correct action, making movements different from the correct action (in variable practice) will not result in the development of an increment of perceptual trace strength. Thus, Adams' theory predicts that variability-in-practice sequences, in which the learner experiences a number of targets around a central criterion target, should be less effective in learning the criterion target than practice on the target itself. The evidence showed, however, that variability in practice

is even superior to practicing the transfer target itself [ShZ81].

SCHMIDT'S THEORY

Largely because of the dissatisfaction with Adams' theory, Schmidt [SZH79] formulated a theory that can be considered as a rival to Adams'. Schmidt's primary concern with Adams' position was the lack of emphasis on open-loop control processes. Yet, the Schmidt theory has borrowed heavily from the ideas of Adams in hopes of keeping the most effective parts and eliminating defective ones.

The basic premise is that with practice, individuals develop rules similar to production systems rules, about their own motor behavior. This relates to the ideas of a generalized motor program (each particular movement has a parameterized motor program) to which a set of parameters must be applied in order to perform an action. Schmidt's theory proposes that rules are learned during practice. These rules are a relationship between all the past environmental outcomes that the individual produced and the values of the parameters that were used to drive the generalized motor programs to produce those outcomes.

In other words, the rules represent the relationship between what the individual "told the motor system to do" and what the motor system actually accomplished. These rules, maintained in memory, are called the "recall schemas", and can be used to select a new set of parameters for the next movement situation that involves the same motor program. Knowing the rule and what environmental outcome is to be produced, the individual can select the parameters for the program that will produce the desired motion.

Schmidt's theory does not specify where the motor programs originate. The theory had to assume that programs are developed in some way and that they can be carried out by executing them with the proper parameters. The theory also says nothing about how the rules about parameters and sensory consequences are developed. Another important point is that, according to the theory, it is not clear how an individual makes the initial responses before any generalized program can exist.

Compared to Adams' theory, Schmidt's theory has the advantages that it accounts for more types of movements; also it seems to account for error detection capabilities more effectively, and it seems to explain the production of novel responses in an open-skills situation. However, the complexity of the problem of motor learning is such that many issues have not yet been investigated nor understood [ShZ81].

The hybrid locomotion control system proposed in this research utilizes Schmidt's ideas of developing production rules during motion practice. Moreover, it incorporates a set of well-defined techniques that show how these rules are developed (or firmed up), how the system functions before their existence, and how movements are produced differently as a result of practice and experience.

2.5. MOTION DYNAMICS IN BIOMECHANICS

Biomechanics is the study, using mechanical principles, of animal bodies in motion and rest. Thus, the science of biomechanics lies at a juncture between anatomy, physiology, physics, and engineering; and consequently it combines the study of motor control and motor learning [Akk79]. Most of the work in biomechanics has been done on human bodies, but the fundamental principles are similar for other higher animal species and locomotive robots. A desirable goal of biomechanics and, for that matter, of movement simulation systems for computers, is a model which, given nervous signals, would provide muscular forces and, given muscular and other internal and external forces, would provide the positions, velocities, and accelerations of each limb. The complexity of the problem is such that this has not yet been achieved for the body as a whole, though many of the issues involved are fairly well understood [WiL85].

An early landmark use of computers for biomechanical analysis was the study by Chaffin [Cha69]. Chaffin developed a simple seven-segment model of the human body and used a computer program to determine the muscular torques and reaction forces which account for the static configuration of joints in the sagittal plane. Given masses, locations of centers of gravity, moments of inertia, and the positions of joints and segments, the program returned net muscular forces required at each joint to maintain stability as well as reaction forces at each joint. Chaffin used the same program to model in more detail the shear and compressive forces on the spinal vertebrae in varying positions. In comparing results of his model to experimentally-determined maximum voluntary forces exertable at joints, Chaffin found that the model returned predictions of muscle forces within an acceptable range.

In 1970 Kane and Scher published a description of the dynamics of the human body in free space [KaS70]. By 1976, Huston and Passarella developed a set of dynamics equations for a body with 34

degrees of freedom [HPH76]. Another very complete mathematical model of the human body is that of Hatze [Hat77]. Also R.B. Stein [Ste82] [STM83] [SMT85] studied nervous system control in limb movements during locomotion in cats.

A sophisticated computer analysis system which included some use of simple computer graphics images was developed by William and Seireg in 1970 [WiS78]. This system could analyze the entire body, or smaller segments such as the jaw, and included a facility for displaying the bodies on an interactive graphics display. Dynamic analysis for this model was quasi-static, i.e. nonlinear, velocity-dependent forces such as the Coriolis force were not included. This simplification is acceptable as long as the bodies being studied are not moving very rapidly.

In concluding this chapter we should mention that the idea of replacing part of the heavy dynamics computations of articulated bodies has been around for some time. For example, Albus [Alb75] [Alb79] [Alb81] proposed a unique control scheme called the Cerebellar Model Articulation Controller that was based on models of human memory and neuromuscular control. This control scheme was based on a mathematical module that, in a table look-up fashion, produced a vector output in response to a discrete state vector input (the Cerebellar Model Arithmetic Computer, CMAC). In the controller, the state vector input was composed of position and velocity feedback from the robot joints, as well as additional state variables that provided a command input to the system. The output vector was the drive torques to the robot actuators. Assuming that the values in the table were adjusted correctly (through training techniques), the robot would automatically follow the correct trajectory if put in the correct initial state and given the correct command state. (The correct input state would result in a set of actuator drives that would cause the arm to move, generating a new input state that would result in new actuator drives, and so on.) While the system was capable of generating such "learned responses" once the memory was trained, training techniques that would make the control approach suitable for use were impractical [Rai77].

Hock [Hoc66] proposed a method by which the control computer could "learn" to control a "pony-sized" walking vehicle on difficult terrain. The vehicle was made to walk on level ground, using preprogrammed leg motions, while values of each of several quantities were recorded as "ideal" values.

Later, when the vehicle walked over irregular ground, the instantaneous value of each quantity was compared to its ideal value. The result of each comparison was encoded as +1, -1, 0, accordingly as the measured value was larger than, smaller than, or about the same as the ideal value. The pattern of +1's, 0's, and -1's so obtained was interpreted by a pattern-recognition program in the control computer. Hock proposed that perturbations of the actions recorded during preprogrammed-gait training period would constitute an adequate repertoire of control actions.

CHAPTER 3

SYSTEM ARCHITECTURE AND THE KINEMATICS OF MOTION

3.1. INTRODUCTION

It is the work of this chapter to lay the foundation for the design of the hybrid numerical/knowledge-based locomotion control system for the four-legged articulated robot. Toward this end, the overall architecture of the proposed system is described in the first section of this chapter. The four-legged articulated robot has been modeled and described in section two. Section three investigates the kinematics of the robot motion. This involves the development of the joint coordination control algorithms. Such algorithms include automatic leg positioning and automatic body height, pitch, and roll regulation over undulating terrain.

3.2. THE ARCHITECTURE OF THE PROPOSED SYSTEM

Despite the growing number of useful knowledge-based systems that are based on purely heuristic symbolic knowledge, several recent knowledge-based systems are coupled with various modes of numerical computing such as numerical analysis, statistics, and simulations. This coupling has taken several forms, including analysis of numerical data, reduction of large sets of numerical results into symbolic information, and guidance of complex numerical computing processes [ABC86], [CoK86], [GlK86], [LoT86], [TEC86] and [Unm86].

For example, Unmeel [Unm86] suggests that knowledge-based systems can be useful in computational aerodynamics and fluid dynamics, and he describes a hypothetical knowledge-based system called AERODYNAMICIST. As computational aerodynamics is applied to increasingly complex and time-consuming problems requiring many calculations, knowledge-based systems have been found to have the potential to *assist* computational aero-dynamicists in their analysis and design tasks. Unmeel identifies

two related aspects of computational aerodynamics: reasoning and calculating. Reasoning is based on factual and heuristic symbolic knowledge of both computational procedures and aerodynamics and can be carried out by symbolic processing and inferential procedures. Calculation is *guided by* reasoning and based on numerical algorithms. These two modes of computation are interwined and it is impossible to do one without the other. Symbolic processing can be applied to many elements of computational aerodynamics such as algebraic formula manipulation, analysis of experimental data, management of databases, selection of problem solving methods, analysis of computational results, and facilitating man-machine interfaces.

One of the currently accepted methods of aerodynamic design is the numerical optimization approach to produce designs that achieve user-specified objectives. The designer specifies an initial aerodynamic body in terms of design: variables, objectives, constraints, and conditions. An optimization method is used to optimize a selected design objective while satisfying the postulated constraints. A knowledge-based system can be used to guide the process of optimization. Its knowledge would utilize databases of previous designs and experience related to speeding up the interactive optimization process.

Existing coupled symbolic and numeric computing systems can be classified into two classes. The first, and most common approach, is coupled systems that treat the numerical routines essentially as "black boxes". These systems are referred to as shallow coupled systems. As an example of such class is a shallow coupling system that is utilized by a Carnegie-Mellon expert system to solve non-linear algebraic equations in [TEC86]. This system manages the application of three different numerical programs during the solution process. The expert system component determines which program to apply, depending upon how each affects convergence of the state variables. This system was reportedly able to solve sets of equations that none of its constituent numerical programs could solve individually. Other shallow coupled systems can be found in [CaO86] [Cha86] [NGS86].

A second coupling approach is to develop systems utilizing extensive knowledge of all participating processes (such as functions, inputs and outputs, usage constraints and limitations, and the like). This knowledge is integrated with other information and used directly by the knowledge-based system component during problem solving. An example of a deep coupled system is the Engagement Analyst's

Apprentice (EAA) developed by North American Rockwell [ABC86], and it is intended to assist the user in constructing a simulation from simulation modules and to evaluate the results. EAA employs extensive knowledge of each simulation module in order to achieve its objective.

Often, however, coupling is not required, but either the symbolic or numerical processes are enhanced by it. For example, intelligent interfaces may not be necessary, but often they greatly improve the utility of numerical programs. Symbolic computing can sometimes be enhanced by the acquisition of procedural knowledge in the form of numerical algorithms (algorithms capable of replacing computationally intensive search routines) and by the improved precision of numerical processes.

This thesis proposes a shallow coupled system for the locomotion control of a four-legged articulated robot. The system architecture is depicted in figure 3.1. The system consists of two controllers, a numerical controller (NC) and a learning controller (LC). Initially, the LC does not share in the control of the robot locomotion. Rather, it develops its experience by observing the NC solution sequences that solve locomotion problems currently beyond the LC's own planning abilities. General motor skills are automatically assimilated in the form of production rules in the LC via a knowledge-based analysis of how the observed solution sequences achieved their goals. These production rules contain knowledge about what forms of torque and force profiles are needed by the robot to achieve sustained stable locomotion.

Sustained stable locomotion means:[1]

(1) maintaining the robot orientation, having control over its velocity, and avoiding obstacles,

(2) choosing the most appropriate locomotive skill at any point during navigation (e.g, walk, trot, climb, etc.),

(3) dealing intelligently with the environment (see later),

(4) maintaining the robot's static and dynamic stability,

[1]The most important of these requirements are (1), (3), (4) and (8).

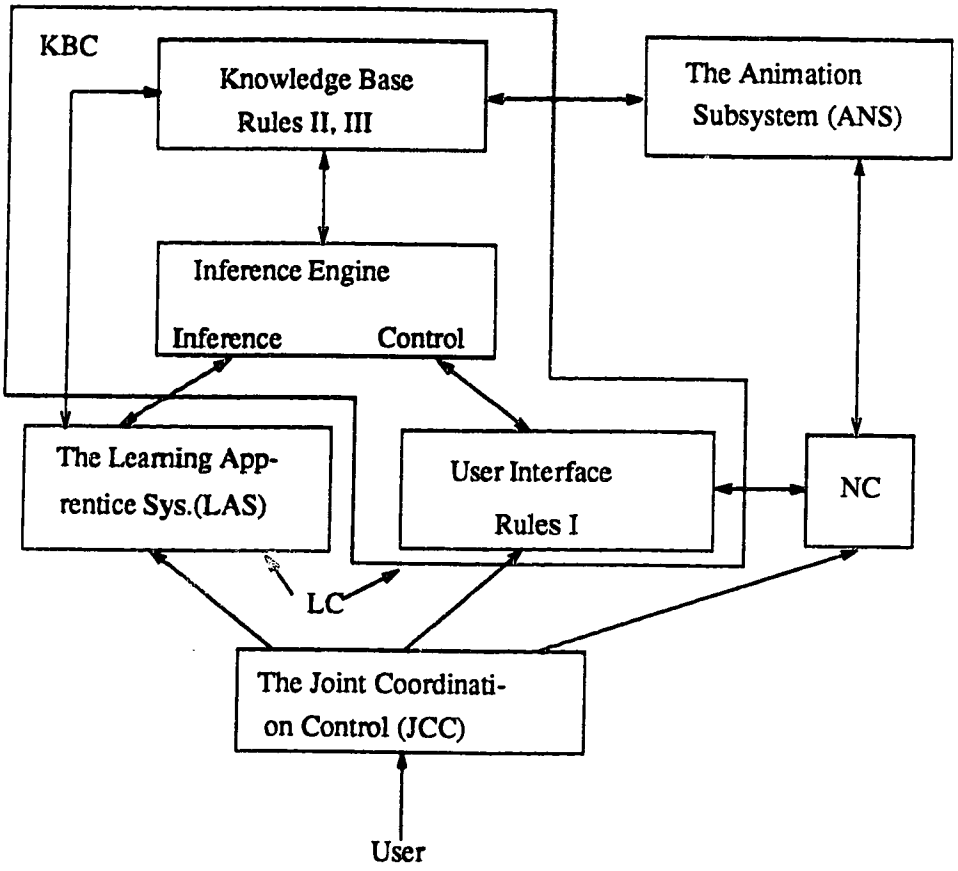

Figure 3.1 The Architecture of the Proposed System

(5) combining different locomotion skills (e.g, turning while running),

(6) achieving smooth transitions between different locomotive skills,

(7) preferring the paths traveled on before,

(8) reducing total energy consumption in executing missions.

The four-legged articulated robot "lives" in an environment that contains obstacles such as blocks, holes, inclines, and rough terrain. The robot has a simulated onboard camera (since we are not building a real system) fixed on top of it. This camera regularly feeds to a navigation system the local obstacles that the robot faces. The initial traversals of the robot are based on a local navigation strategy that relies on the

on-board camera information. At any stage in the navigation the terrain is characterized by a partially built world map.[2] This world map is updated from time-to-time by integrating information from the on-board camera. The robot employs a global navigation strategy that uses the learned world map in the regions it is available, and resorts to local navigation in the regions the learned map is not available. This navigation strategy is similar in many ways to the strategies proposed by Meystel in [Mea86a] [Mea86b] [Mea87b] [Mea87a].

Since the robot cannot anticipate where small road obstacles are, it has to edit its motion on the fly in order to avoid or overcome the "perceived" obstacles. The necessity to edit the motion on the fly has been investigated before in [WiB78]. Winkless defined intelligent robot motion as ".. the ability to do appropriate motion under unpredictable conditions". Thus, a preprogrammed robot motion, highly accurate, precisely measured and well understood cannot be considered an intelligent one, since the robot would have no ability to cope with unpredictable situations and to choose between alternatives. Such preprogrammed motion has been tried in the experimental system. The evaluation of the produced motion can be found in chapter 6.

Actually the problem of controlling the motion of autonomous articulated robots is multifaceted. In the general case, the robot must navigate in terrain which is unknown and beset with obstacles. This requires perceiving the local situation, planning, obtaining at least suboptimal solutions to some combinatorial problems, and solving equations of motion with sufficient precision to predict and control the behavior of a dynamic system. In this research, our main interest is in the problem of controlling motion, assuming that the terrain is perfectly perceived and known in the robot's vicinity, and that a general plan has already been formed. The solutions sought are those which have potential of execution in real time.

The LC contains a learning apprentice system (LAS) that acquires knowledge of the desired control of the robot's limbs with the help of a human teacher, who prescribes a mission and corrects torque profiles to achieve the desired motion control. The LAS incrementally "compiles" the training experience into a set of rules to be used by the LC (according to Schmidt's theory- see section 2.4). This incremental acquisition

[2]See the details of the navigation system in Section 6.2.

of locomotive knowledge makes the LC gradually go through a natural growing process from an apprentice to the NC, to an assistant and finally to a partner to the NC. This control upgrading takes place gradually and incrementally.

First the learned LC's rules are added as a symbolic interface to manage the NC's computations (symbolic front end). But no matter how smart the symbolic interface is, the NC is limited by the open/closed kinematic chains dynamics computation speed which can't be performed in real time (see Chapter 4). What is needed to be looked at, therefore, was how to replace parts of these heavy computations with more efficient symbolic computations. This was achieved by upgrading the LC capabilities to an intelligent locomotive solver which solves locomotion problems on its own when it is possible. As will be explained in Chapter 5, the LC will gradually transform from a high level symbolic interface to an intelligent locomotive solver that will in some (eventually in the majority) of the cases be able to solve locomotive problems on its own without the help of its numerical partner, the NC. Consequently, the robot will be able to control and coordinate its motion using the LC instead of solving the dynamics equations of motion.

The advantage here will be the saving of the dynamics computation time needed by the NC and the real time motion control of the robot. Moreover, the LC will be able to do purposeful activities under unpredictable conditions in the environment (*e.g.* obstacles). These purposeful activities are formed in redundant situations, i.e., in situations which cannot be exhaustively overviewed and properly programmed. To deal with the unpredictability of the environment, the LC develops fine-tuning and terrain adaptation rules that manipulate the torque profile and tailor the motion to the actual details of the terrain during actual motion execution. These rules are also learned by the LAS (a part of the LC). This approach can be viewed as "fine-tuning of the dynamic behavior of the motion" or simply what can be called "dynamic regression", where rules are used as post processes that modify the robot motion on the fly during the actual motion execution.

The idea of editing torque and force profiles comes mainly from the work of John Hollerbach who used a mass-spring model to simulate handwriting in his dissertation [Hol78]. Hollerbach noticed that one

of the features of springs is that they oscillate when set in motion. He conceptualized the muscles that make up the finger movements as four springs (left- right- up- down) with the finger links concentrated as a mass in the middle. If this mass is thrown in a diagonal direction and then released, it will oscillate in a circle. All that is required to keep the mass oscillating is an occasional pulse of force delivered at the proper time in the sequence. If we imagined that a paper is translated leftward under this oscillating spring system, the resulting trajectory perhaps will be a series of loops. Hollerbach's thesis is that if the pattern of the force application to the spring system is changed very slightly, a set of loops that forms the handwritten letters "e","u","n", etc. can be formed.

Asaytryan also has a similar idea of editing force and torque profiles [SZH79] that are acting on a mass-spring model of a human figure. Asaytryan was trying to prove that complex mass-spring structures could, in certain gross ways, behave like muscles. His model responded in an intuitively acceptable manner to changes in joint torque and force histories. The simulated human figure was capable of reasonable reproduction of the actual movements, with some small differences manifested as the computations proceeded.

Platt and Badler [PlB81] also developed a dynamic spring-based model for facial expressions simulations. Patterns of force and torque profiles were used to generate different facial expressions. The model was successful in implementing many different realistically looking facial expressions.

Simulation of a kicking motion of a rugby player punting a football in which torque histories were manipulated was reported in [MJW85]. The movement was modified as follows. In order to extend the reach of the leg of the kicker, the magnitude of the knee extensor torque history was increased by 40% (i.e. multiplied by a scaling factor that is not necessary linear). Given the form of the produced simulated image sequence, it was recognized that modifications to other joint torques were necessary. Increasing the ankle torque history by 18%, coupled with the previous changes to the knee torque produced the desired movement sequence. Good agreement was achieved between the displayed angular displacements for the original and simulated movements.

In our system a convenient method for modifying the joint angles and torque profiles uses curve control functions. The use of curve control functions allows the teacher to specify and edit complex motions and to produce smooth motion patterns. The user interface of the system provides an interactive graphical editor which allows the teacher to edit the curves. In order to edit a joint's movement, the teacher selects the joint (using the mouse). The curves (both torques and angles) for all degrees of freedom at this joint will be displayed for editing. Using point modifications, the curves can be modified.

In the following sections brief descriptions of the modules in figure 3.1 are presented.

3.2.1. THE NUMERICAL CONTROLLER (NC)

The NC deals with the dynamics of the robot motion. Given both the body's desired trajectory and the leg joint commands (the outputs of the JCC) of the robot, the NC uses two recursive formulas to solve the inverse dynamics problems for both the open and closed kinematic chains. The NC implements a "pipelined" computation model on a network of processors for fast execution of the heavy dynamics calculations. In calculating the forces and torques needed, the NC deals with the legs that are in contact with the ground using a linear programming formalism developed in chapter 4. The legs that are not in contact with the ground form open kinematic chains and, therefore, are dealt with using open chain inverse dynamics equations.

The robot uses several types of gaits and leg cycles. These gaits are characterized by patterns of movements that are repeated over and over, step after step. Certainly, there are variations between steps due to the nature of the terrain (i.e., roughness and slopes). Due to these varying patterns of gaits and leg cycles, a number of rather complicated sequences of closed and open chains are formed many times in the course of the robot navigation. This imposes the requirement to have a general numerical locomotion control system that takes care of the formation and breaking of the chains during the motion. The computational requirements of this locomotion control system fall into the "supercomputer" class, and cannot be executed in real time on our system (an IRIS 2400 system- see chapter 6).

The details of the robot motion dynamics are described in chapter 4.

3.2.2. THE LEARNING CONTROLLER (LC)

The LC maintains three (initially empty) sets of production rules of the form "situation implies action" (S $\rightarrow$ A). These rules regulate the robot's locomotion behavior. They go through modifications and enhancements by the LAS as a result of practice and experience. A rule that still has an incompletely learned S and/or A is called a "rule shell". The three sets of rules are:

Skill Usage Rules: These rules identify the contexts under which the various learned skills should be used (called the conditions-of-success) and how to drive the NC to implement them. An example of a rule shell in this set is Walk(A,B): If the robot's four legs are in some orientation (X,Y,Z,W) within a particular leg cycle (i) (for the definition of orientation and leg cycle see section 3.4), and the goal broadcast is ($A \rightarrow B$) such that the road from A to B is flat and the distance between these two points is less than or equal to 5 units of distance, and both points are located along a NS (North-South) street; then start to drive the NC with the lc_1 leg cycles for all legs such that the legs start from the closest positions to the initial leg settings (X',Y',Z',W') in lc_1. This is in order to facilitate the smooth transitions between different types of gaits. The linear body speed is within the range [a,b], and the stopping conditions are of the destination point B (within some tolerance).[3]

Prototypical Torque Profile Rules: These rules use torque profiles that are calculated by the NC and are previous experiences to draw upon. A rule shell for a walk skill in this rule set would be similar to the previous rule example replacing the right hand side with the torque profile calculated by the NC. Building these rules is nothing but priming the LC with the torque profiles produced by the NC (previous experience), but in compact form (since these rules will have been firmed up by generalizations and discrimination algorithms of the LAS - see chapter 5). The torque profiles are used to directly drive the direct dynamics equations in the animation subsystem (see section 3.2.4).

---

[3]The actual forms of these rule types are described later.

Motion Enhancement Rules: These rules describe the short-term monitoring that is needed to allow the robot to edit the torque profiles on the fly during the actual motion execution. They identify the various types of obstacles that have been perceived before and prescribe various ways of editing the torque profiles in order to overcome or avoid these obstacles.

As shown in figure 3.1, the LC consists of two components: a Knowledge-Based Controller (KBC) and a Learning Apprentice System (LAS). The details of the LAS structure are described next section. The KBC has a structure different from conventional expert systems (such as R1/XCON). In our context the KBC has to integrate sensory data, robot capabilities, and task constraints in order to generate acceptable locomotion movements. The KBC has to be able to do all that and operate at speeds comparable to the real-time constraint of the locomotion task. For example R1/XCON (the VAX configuration expert system) has no real-time constraints, it ran at approximately 5 rules/sec on a main frame DEC-10 [MMS85]. For our KBC we would like to execute a rule on the order of every millisecond, 200 times faster, on a microcomputer on-board the robot. In order to achieve that, our KBC have to rely on techniques to compile the rules, to have a limited function inference engine, to divide the rules into smaller and smaller sequential pieces, and to use conventional languages to describe the rules in order to have the capability to rapidly process both symbols and numbers.

Russell Anderson's ping pong robot has a controller similar to KBC that he calls "Real-time Expert Controller" [And88]. Anderson built his controller, that is rule-based and executes loosely specified strategies in real-time, in order to show that reasonably intelligent behavior could be emulated by the ping pong robot.

Conventional robot systems operate slowly and methodologically, often playing back a pretaught sequence of positions and orientations. Anderson tried to exploit human-like heuristics to cope with a dynamic environment quickly enough. Given an incoming ball trajectory, the robot controller picked an initial suitable return shot that satisfied the constraints of the robot and the rules of ping pong. Because the robot system had to act before it actually sensed, the controller continually adapted this initial suitable return as additional, more accurate sensor data arrived. This is what Anderson called "temporal updating".

The controller was able to adapt that ball return plan because it did not commit itself right from the beginning to a detailed return shot. There were many ways to return the ball. Anderson called this "redundant DOFs problem" and invested on the controller the authority to alter them. In other words, Anderson's controller was successful in capturing expert knowledge about how to generate feasible solutions rapidly.

Similar to Anderson's controller, our KBC is implemented in conventional "C" language. Its inference engine is very simple because the data operated on are known and fixed in quantity. This has eliminated the need for pattern matching between rules and variables, and consequently allowed for generating feasible solutions rapidly.

3.2.3. THE LEARNING APPRENTICE SYSTEM (LAS)

This module is responsible for the process of incorporating domain knowledge into the KBC's knowledge base by extracting it from a domain expert (a teacher) and encoding it into production rules (this is what we call "skill transfer"- see later).

Learning Apprentice Systems (LASs) are defined as knowledge-based consultant systems that directly confront the knowledge acquisition bottleneck by learning from interactions with an information-rich external environment [MMS85]. The idea of building a LAS for motor control is drawn from the important field of machine learning in artificial intelligence. Machine learning research has long sought to construct complete, autonomous learning systems that start with general inference rules and learning techniques and gradually acquire complex skills and knowledge through continuous interaction with an information-rich external environment. Learning Apprentice Systems are currently being developed in the domain of VLSI design [MMS85], well-log interpretation [WSK86], and medical diagnosis [WCB86].

In our context the LAS acquires knowledge of the desired control of the robot's limbs with the help of a human teacher, who prescribes a mission and corrects torque profiles to achieve the desired motion control. The LAS incrementally "compiles" the training experience into a set of rules of the three types described above.

At any stage in the robot navigation, the robot would rely on its LC to take charge in controlling its locomotion. If a sustained stable locomotion has been achieved by the robot, the teacher will not intervene. On the other hand, if some type of motion bug (i.e. something that will lead to disturbing the sustained stable locomotion of the robot) is detected by the teacher, he or she will intervene to update the motion of the simulated robot (the torque profiles). In this case, the LAS will take advantage of this teacher intervention and use rule-learning algorithms to modify the current sets of the LC rules. The LC rules are modified because they contain faults, which can be of two types: (1) motion faults: a rule contains an action that calculates incorrect motion; (2) control faults: the rules have undesirable control behavior (e.g. some rules fire when they are not supposed to).

The LAS's main learning algorithm is:

Until all the rules are stabilized (no more changes for a specific number of training missions):

(a) Identify a fault with a rule from a rule set.

(b) Modify the selected rule to attempt to remove the fault.

This mechanism will provide real-time control over the robot motion without disrupting the dynamic integrity of the resulting motion, since the edited torque profile still has to drive the direct dynamics module in order to produce the motion (see the animation subsystem next).

The details of the LAS algorithms will be described in Chapter 5.

3.2.4. THE ANIMATION SUBSYSTEM (ANS)

The animation subsystem is the front end of the experimental system. It is responsible for displaying the simulated robot and interacting with the user. This module communicates with either the NC or the LC by sending and receiving packets over an interprocess communication facility. The animation subsystem invokes either the NC or the LC when the user wants to assign a task to the robot. The appropriate controller will take over the task execution and will send a set of packets to the animation subsystem. These packets represent the next animation frame's forces and torques on the robot limbs. There is one packet in

this set for each limb of the robot. The animation subsystem implements the direct dynamics recursive algorithm of Armstrong that was originally developed for the space Shuttle Remote Manipulator System (CANADARM) [Arm79] later developed in the graphics context [AGL86], and then generalized to the multiprocessor case [AMO87].

The details of Armstrong's direct dynamics equations will be described in chapter 4 and the animation subsystem will be described in chapter 6.

3.2.5. THE JOINT COORDINATION CONTROL (JCC)

This module deals with the kinematics of the robot motion. It contains algorithms which permit the synthesis of gaits suitable for realization of the desired robot trajectory, taking into account the constraints imposed by terrain conditions. In this module, algorithms are implemented for automatic leg positioning and automatic body height, pitch, and roll regulation over undulating terrain. These algorithms involve the study of the kinematics of the robot motion and are developed in section 3.4 of this chapter.

3.3. THE ARTICULATED ROBOT KINEMATIC MODEL

The articulated four-legged robot is modeled in Figure 3.2. A crude estimate of the number of degrees of freedom (DOF) which are needed for free locomotion of this robot was obtained by observing that during locomotion it must ideally be possible to control the six DOF of the body (three translational and three rotational) when it is supported by each of the two alternating sets of legs(at least two legs should be in contact with the ground at any time to achieve static stability- assuming that two legs along with the force of inertia can approximate static stability conditions). So one might expect about twelve DOF to be a minimum for the four-legged robot. If twelve DOF are taken as a rough estimate, they can be distributed among the robot's four legs of three DOF each. The robot of figure 3.2 has three DOF for each of its four legs: two at the hip (one for elevation and another for lateral movements) and one DOF on the knee.

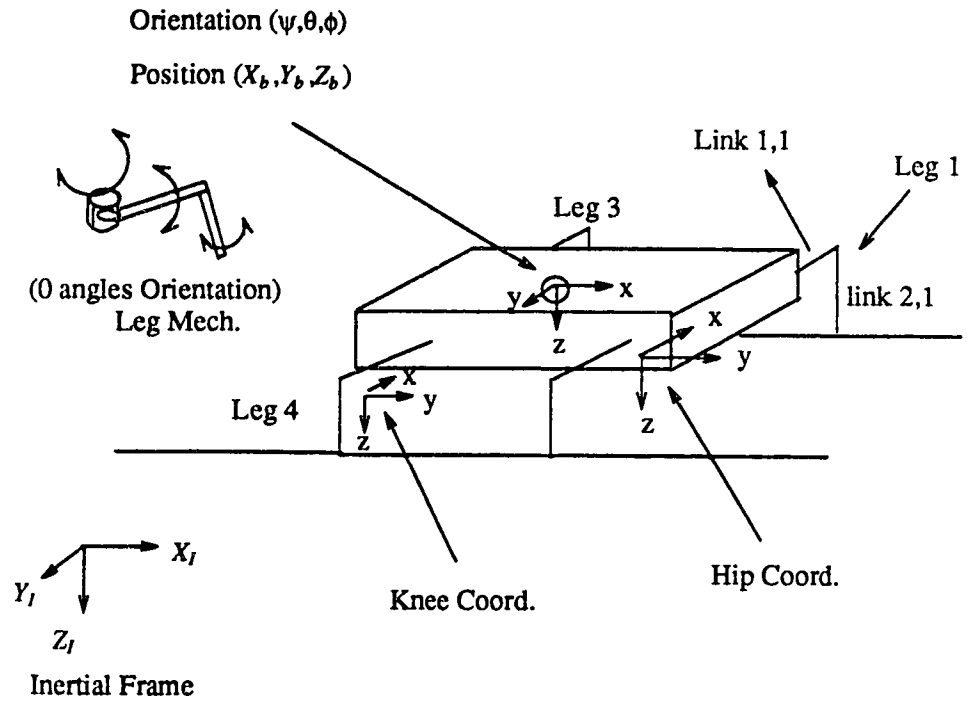

Figure 3.2 The Kinematic Model of the Four-Legged Articulated Robot

The following convention has been assumed for the coordinate axes. The X_I coordinate axis is directed towards the desired initial direction of travel, the Z_I coordinate axis is in the direction of gravitational acceleration (positive downward), and the Y_I axis is in the direction of the vector cross product of Z_I and X_I. The (X_I,Y_I,Z_I) frame defines the inertial frame that is fixed in the nonrotating earth. The body is modeled as a single rigid rectangular box with the legs adjoined to the body at or close to the outside edges of the side surface. A right-handed body fixed coordinate frame (X,Y,Z) is established with its origin fixed at the center of gravity of the body. The total state of the articulated robot is described by five state vectors:

(1) The body state vector.

(2) The four legs' state vectors.

The body state vector is a twelve-element vector that describes:

(a) The position of the center of gravity of the body relative to the inertial frame (X_I, Y_I, Z_I). This is shown in figure 3.2 as $(X_b, Y_b, Z_b)^T$.

(b) The components of the translational velocity of the center of gravity $(\dot{X}_b, \dot{Y}_b, \dot{Z}_b)^T$.

(c) The body orientation Bryant angles (also known as Cardan angles) (ϕ,θ,ψ). These angles are defined such that they are all simultaneously reduced to zero, when the (X_I, Y_I, Z_I) axes are parallel to the (X,Y,Z) body axes, respectively. The rotation from the earth fixed inertia frame (X_I, Y_I, Z_I) to the fixed body frame (X,Y,Z) is accomplished by first rotating an angle ψ about the Z-axis (azimuth), then an angle θ about the rotated Y-axis (elevation), and finally an angle ϕ about the rotated X-axis (roll).[4]

The transformation matrix between the body fixed coordinate frame (X,Y,Z) and the earth fixed frame (X_I, Y_I, Z_I) is given by:

$$T_{Ib} = \begin{bmatrix} \cos\theta\cos\psi & \sin\psi\cos\theta & -\sin\theta \\ (\cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi) & (\cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi) & \cos\theta\sin\phi \\ (\sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi) & (\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi) & \cos\phi\cos\theta \end{bmatrix} \qquad 3\text{-}1$$

Bryant (Cardan) angles are particularly useful in cases where a body is moving in such a way that body fixed coordinate frame deviates only little from the inertial coordinate. For sufficiently small angles the linear approximation $\sin\alpha \approx \alpha$, $\cos\alpha \approx 1$ $(\alpha = \phi,\theta,\psi)$ yields:

$$T_{Ib} = \begin{bmatrix} 1 & \psi & -\theta \\ -\psi & 1 & \phi \\ \theta & -\phi & 1 \end{bmatrix}$$

Consequently, we can write the approximated transformation matrix on the form:

$$T_{Ib} \approx E - (\phi,\theta,\psi)^{\tilde{}}$$

Where E is the identity matrix, and the 'tilde' operator converts the vector $(\phi,\theta,\psi)^T$ into a matrix form:

---

[4]The sequence of rotations chosen is significant, for each sequence uniquely defines a different body orientation. There are six permutations of the three axes about which rotation will occur. Therefore there are six kinds of Cardan angle systems. Cardan angles of the first kind, in which rotations occur about X-, Y-, and Z-axis respectively are referred to as a body-three: 1-2-3 system where the three rotations axes are fixed within the body, and the order of rotation is X, Y, and Z [TuP87]. The Cardan angle system chosen here is of the body-three: 3-2-1 rotation.

$$(\phi,\theta,\psi)^{\tilde{T}} = \begin{bmatrix} 0 & -\psi & \theta \\ \psi & 0 & -\phi \\ -\theta & \phi & 0 \end{bmatrix}$$

It makes no difference whether $(\phi,\theta,\psi)^T$ are interpreted in the inertial frame or in the body fixed frame. This can be shown as follows. If $(\phi,\theta,\psi)^T$ designates the body orientation in (X,Y,Z) then the body orientation in the inertial frame is the linear approximation:

$$T_{Ib}\ (\phi,\theta,\psi)^T \approx (E - (\phi,\theta,\psi)^{\tilde{T}})\ (\phi,\theta,\psi)^T$$

Because of the identity $(\phi,\theta,\psi)^{\tilde{T}}\ (\phi,\theta,\psi)^T=0$, this is identical with $(\phi,\theta,\psi)^T$. The result of these considerations is that within linear approximations small rotation angles can be added like vectors.

(d) The body angular velocity $(\omega_x,\omega_y,\omega_z)^T$ expressed in body coordinate (X,Y,Z).

The relation between the body angular velocity $(\omega_x,\omega_y,\omega_z)$ and the rotation rates $(\dot{\phi},\dot{\theta},\dot{\psi})$ is given by:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \sin\theta & 0 & 1 \\ -\cos\theta\sin\psi & \cos\psi & 0 \\ \cos\theta\cos\psi & \sin\psi & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad 3\text{-}2$$

Again for sufficiently small angles, the linear approximation $\sin\alpha\approx 0$, $\cos\alpha\approx 1$ yields:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

Each leg consists of two rigid links with a one-degree of freedom knee. Two other degrees of freedom at the hip make a total of three kinematic degrees of freedom for each leg. In order to describe the state of the robot legs, three additional coordinate frames for each of the legs are defined:

(a) The coordinate frame $(X_h,Y_h,Z_h)_i$, has its origin at the hip of leg i. The Z_{hi}-axis is parallel to the body Z-axis and directed downward. The Y_{hi}-axis is perpendicular to the plane of the leg segments, and the X_{hi}-axis is uniquely determined by maintaining a right hand coordinate system.

(b) The coordinate frame $(X',Y',Z')_i$, has its origin at the hip of leg i and rotates with the upper limb segment of the leg. The X_i'-axis is directed along the upper limb segment. The Y_i'-axis is perpendicular to the plane of the upper and lower limb segments. Finally, the Z_i' is uniquely determined by maintaining a right hand coordinate system.

(c) The coordinate frame $(X",Y",Z")_i$, has its origin at the knee of leg i and rotates with the lower limb segment of the leg. The $Z_i^{"}$-axis is directed along the lower limb segment into the supporting surface. The $Y_i^{"}$-axis is parallel to the Y_{hi} and $Y_i^{'}$ axes, and the $X_i^{"}$ is uniquely determined by maintaining a right hand coordinate system.

One may note that when all leg angles are reduced to zero, the $(X_h,Y_h,Z_h)_i$, $(X',Y',Z')_i$, $(X",Y",Z")_i$ coordinate frames are all parallel to the body coordinate frame (X,Y,Z).

Each leg state vector is a six-element vector that describes:

(a) The leg joint angles $(\psi_i,\theta_{hi},\theta_{ki})$. Where ψ_i is leg i's azimuth angle. It is the rotation angle around the Z_{hi}-axis (i.e. it is the angle between the leg plane and the body longitudinal axis, X). θ_{hi} is the hip elevation angle. It is the rotation angle around the $Y_i^{'}$-axis (i.e. it is the angle between the body plane, XY, and the upper limb segment). θ_{ki} is the knee elevation angle. It is the rotation angle around the Y"-axis (i.e. it is the angle between the lower limb and a perpendicular to the upper limb).

(b) The leg joint angle rates $(\dot{\psi}_i,\dot{\theta}_{hi},\dot{\theta}_{ki})$.

Both the body and leg frames are shown in figure 3.2.

3.4. JOINT COORDINATION CONTROL (JCC) ALGORITHMS

In this section the kinematics of the robot motion is described. Kinematic control of the robot involves control of the motion without regard for the forces which produce this motion. The algorithms that synthesize the body and leg trajectories of the robot have been developed. These include algorithms for automatic leg positioning and automatic body height, pitch, and roll regulation over undulating terrain. These algorithms were inspired by the work in [PaF73], [FrV69], and [OrM73]. The kinematic algorithms accept commands from a human animator, and with a knowledge of the present state of the body of the robot relative to the supporting surface, it synthesizes joint commands that direct the robot over the desired trajectory (desired angles, angle rates).

The animator develops a strategy for carrying out a sequence of operations to accomplish specific objectives (e.g. Walk from A to B). Then, he/she begins to provide commands to the kinematic algorithms. These commands are broken into three basic categories:

(a) Speed control commands: Here the heading of the robot remains constant while the desired trajectory for the center of gravity of the robot body is provided by the animator.

(b) Heading control commands: The heading of the robot turns with rates commanded by the animator who controls the forward, backward, or sideways velocities.

(c) Gait Cycles: The animator specifies the type of gait to be performed by the robot. Currently the implemented prototype has fourteen different locomotive skills. These are: Walk, Climb, Rack, Turn-left, Turn-right, Turn-in-place, Trot, Pace, Gallop, Stop, Run, Pronk, Walk-backward, and Walk-sideways.

Many of the characteristic variations in pattern of limb movements can be visualized by two men walking behind each other. If they keep in step, the combined pattern of their leg movements is that of racking; if they are completely out of step, the pattern is that of a trotting; if they could be partially out of step, they would illustrate walking; if they are jumping and at the same time keeping in step, they would illustrate pronking. In all patterns of movement the frequency of movement is the same for all limbs and, except in a gallop, each limb is always one-half of a complete cycle out of phase with its fellow limb on the other side of the body. The distinctive feature of a gallop is partial synchronization of the two fore and two hind limbs. Two fairly clear types of gallop exist: transverse and rotatory, and in neither case is synchronization of the two fore or two hind limbs complete. The sequence of the leg movements are shown in Figure 3.3. If in a transverse gallop the left fore leg leaves or strikes the ground before the right fore, the two hind limbs also move in that order; in a rotatory gallop the reverse is the case, if the left fore leg moves before the right fore leg, the right hind moves before the left hind limb.

Figure 3.3 shows some of these locomotive skills. The rectangles in the figure indicate the times the foot is not in contact with the ground. Notice the possible transitions from one gait to another. For

example, the transition from walking to trotting can be continuous (i.e. smooth transition): an increase in walking speed would make a front leg begin to step before the opposite back leg touches the ground. The more one increases the speed, the more the walking switches to trotting.

In setting up kinematic models for gaits, certain definitions are useful [McG68]. These are:

(1) Stride Length λ_x: is the longitudinal distance by which the body is translated in one complete locomotion cycle.

(2) Lateral Length λ_y: is the lateral distance by which the body is translated in one complete locomotion cycle.

(3) Angular Displacement λ_ψ: is the amount by which the body is rotated in one complete locomotion cycle.

(4) Period τ: is the time required for one complete locomotion cycle of the gait.

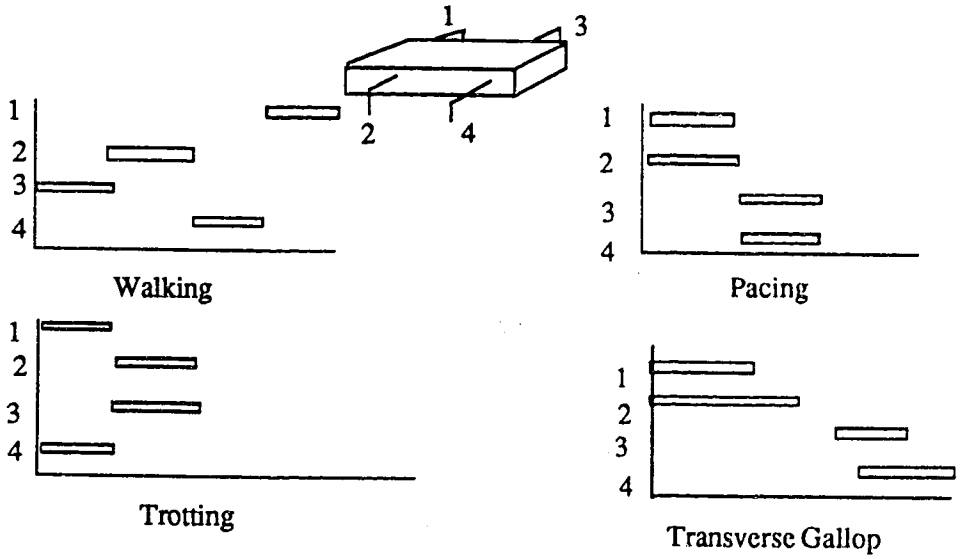

Figure 3.3 Some of the Robot's Basic Locomotive Skills

(5) Duty Factor β_i: is the relative amount of time spent on the ground by each leg during one locomotion cycle.

(6) Relative Phase ϕ_i: is the amount of time by which leg i (i=2,3,4), lags behind that of leg 1 expressed as a fraction of the time required to complete one locomotion cycle.

For straight-line locomotion, the period is equal to the stride length divided by translational rate.

$$\tau_x = \frac{\lambda_x}{\dot{X}_b}$$

For the turn control and the side motion control:

$$\tau_\psi = \frac{\lambda_\psi}{\omega_z}$$

$$\tau_y = \frac{\lambda_y}{\dot{Y}_b}$$

In order for the animator to specify a configuration of a leg, two methods of description are known, namely joint space (Figure 3.4) and Cartesian space (Figure 3.5). From the point of view of the kinematic control system and the formulation of dynamics (see next chapter), the most convenient description of a leg configuration is as a vector of joint variables. However, animators will experience considerable difficulty relating joint vectors to positions and orientations in Cartesian space, that are more naturally described in orthogonal, cylindrical, or spherical coordinate frames. The joint vector and Cartesian space descriptions of a configuration are reconciled through the kinematic equations of the leg.

Generally, the transformation from joint vector to Cartesian space is simple and efficient to compute. However, the inverse is often intractable except by numerical methods; and even where it is solvable, several distinct joint vectors typically give rise to the same Cartesian space description of a configuration.

To achieve a leg gait, a leg is driven in a cyclical trajectory. The motion of the foot viewed from the side in body coordinates is shown in Figure 3.5. The trajectory consists of two phases: a transfer phase in which the foot is off the ground with zero reaction forces between the foot and the ground , and a support phase when the foot is in contact with the ground and exerts some nonzero force on the supporting surface.

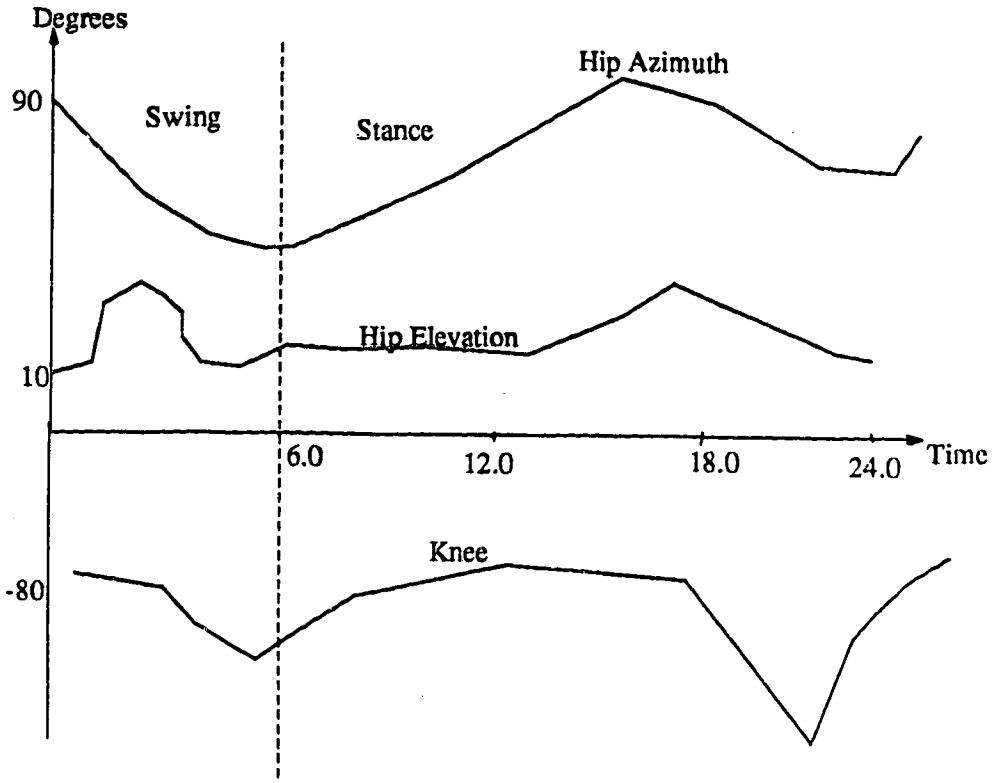

Figure 3.4 A Configuration of a Leg in the Joint Space

Each movement from i to i+1 in Figure 3.5 represents the position of the leg with respect to the body's coordinate system. For translating each Cartesian leg position of Figure 3.5 into a set of leg joints of Figure 3.4 the inverse kinematics is evaluated analytically (this is how we resolved the redundant solution problem). The details of the inverse kinematics formulas are described in the next section.

It is assumed that each foot could be placed on the ground at a sequence of pre-computed positions determined prior to the start of the motion. These positions are evenly spaced along the desired direction of travel and separated by one stride length for any given foot. Neighborhood places are considered as potential footholds in case of the existence of obstacles along the direction of travel. This is possible since the length of each leg measured from its hip joint to its foot is allowed to vary through knee flexure. Also the automatic body regulation algorithm (see section 3.4.2) provides a wider selection of allowable

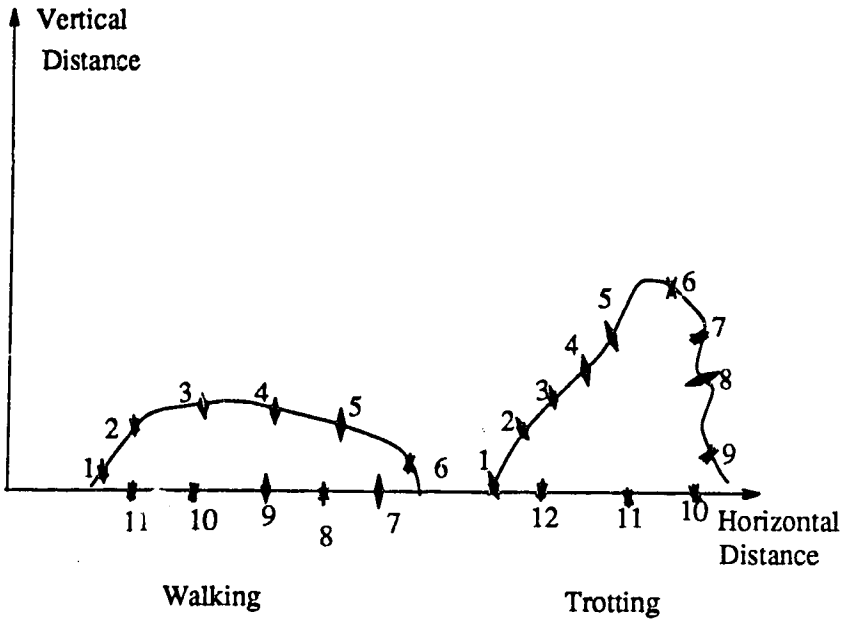

Figure 3.5 The Robot Foot viewed from (Cartesian Space) the body Coordinate frame

footholds and is used to maintain maximal stability.

3.4.1. AUTOMATIC LEG POSITIONING ALGORITHM

In the inverse kinematics calculations the leg angles and their rates should be calculated from a knowledge of the state of the body and the position of the foot. Let the position of the foot of leg i be given by the vectors $(X',Y',Z')_i$, $(X,Y,Z)_i$, $(X_E,Y_E,Z_E)_i$ expressed in the hip, body, and inertia frames respectively. The relationship between these representation are:

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = T_{Ib} \begin{bmatrix} X_{iE}-X_b \\ Y_{iE}-Y_b \\ Z_{iE}-Z_b \end{bmatrix} \qquad 3\text{-}3$$

Where T_{Ib} is the inertia-to-body transformation matrix (transformation matrix that converts vector representations in the inertial frame to the representations in the body frame) given by equation (3-1), And

$$\begin{bmatrix} X_i' \\ Y_i' \\ Z_i' \end{bmatrix} = T_{bh} \begin{bmatrix} X_i - a_i \\ Y_i - b_i \\ Z_i - c_i \end{bmatrix} \quad 3\text{-}4$$

Where T_{bh} is the body-to-hip transformation matrix. T_{bh} can be obtained from [3-1] by setting ϕ=0, $\theta = \theta_{hi}$, and $\psi = \psi_i$. This gives:

$$T_{bh} = \begin{bmatrix} \cos\theta_{hi}\cos\psi_i & \sin\psi_i\cos\theta_{hi} & -\sin\theta_{hi} \\ -\sin\psi_i & \cos\psi_i & 0 \\ \cos\psi_i\sin\theta_{hi} & \sin\psi_i\sin\theta_{hi} & \cos\theta_{hi} \end{bmatrix} \quad 3\text{-}5$$

$(a_i, b_i, c_i)^T$ are the coordinates of the hip socket for leg i as expressed in the body fixed frame (X,Y,Z) (see figure 3.2).

Let l_1 be the length of the upper limb segment and l_2 the length of the lower limb segment, then from figure 3.2, it can be noted that:

$$\begin{bmatrix} X_i' \\ Y_i' \\ Z_i' \end{bmatrix} = \begin{bmatrix} l_1 + l_2\sin\theta_{ki} \\ 0 \\ l_2\cos\theta_{ki} \end{bmatrix} \quad 3\text{-}6$$

Combining 3-4 and 3-6,

$$\begin{bmatrix} X_i - a_i \\ Y_i - b_i \\ Z_i - c_i \end{bmatrix} = T_{bh}^T \begin{bmatrix} l_1 + l_2\sin\theta_{ki} \\ 0 \\ l_2\cos\theta_{ki} \end{bmatrix} \quad 3\text{-}7$$

$$= \begin{bmatrix} (l_1\cos\theta_{hi} + l_2\sin(\theta_{hi}+\theta_{ki}))\cos\psi_i \\ (l_1\cos\theta_{hi} + l_2\sin(\theta_{hi}+\theta_{ki}))\sin\psi_i \\ -l_1\sin\theta_{hi} + l_2\cos(\theta_{hi}+\theta_{ki}) \end{bmatrix} \quad 3\text{-}8$$

Writting 3-8 in scalar form and dividing 3-8-b by 3-8-a gives:

$$\frac{Y_i - b_i}{X_i - a_i} = \tan\psi_i$$

$$\psi_i = \tan^{-1}\left[\frac{Y_i - b_i}{X_i - a_i}\right] \quad 3\text{-}9$$

Now adding the squares of 3-8-b and 3-8-c after multiplying the latter with $\sin\psi_i$ gives:

$$(Y_i - b_i)^2 + (Z_i - c_i)^2\sin^2\psi_i = l_1^2\sin^2\psi_i + l_2^2\cos^2\theta_{ki}\sin^2\psi_i + l_2^2\sin^2\theta_{ki}\sin^2\psi_i + 2l_1l_2\sin\theta_{ki}\sin^2\psi_i \quad 3\text{-}10$$

or

$$\theta_{ki} = \sin^{-1}\left[\frac{\frac{(Y_i-b_i)^2}{\sin^2\psi_i}+(Z_i-c_i)^2-l_1^2-l_2^2}{2l_1l_2}\right] \qquad 3\text{-}11$$

Performing similar manipulation to 3-8-a and 3-8-c gives:

$$\theta_{ki} = \sin^{-1}\left[\frac{\frac{(X_i-a_i)^2}{\cos^2\psi_i}+(Z_i-c_i)^2-l_1^2-l_2^2}{2l_1l_2}\right] \qquad 3\text{-}12$$

Now consider equation 3-8-c

$$(Z_i-c_i)=-l_1\sin\theta_{ki}+l_2\cos(\theta_{ki}+\theta_{ki})$$

$$=-l_1\sin\theta_{ki}+l_2\cos\theta_{ki}\cos\theta_{ki}-l_2\sin\theta_{ki}\sin\theta_{ki}$$

$$-l_2\cos\theta_{ki}\cos\theta_{ki}+(l_1+l_2\sin\theta_{ki})\sin\theta_{ki}+(Z_i-c_i)=0 \qquad 3\text{-}13$$

Let $X = \sin\theta_{ki}$ then $\cos\theta_{ki} = \sqrt{1-X^2}$

Substituting in 3-13 gives:

$$-F\sqrt{1-X^2}+EX+G=0$$

$$F^2(1-X^2) = E^2X^2+2EGX+G^2$$

$$(E^2+F^2)X^2+2EGX+(G^2-F^2)=0$$

Solving[5] for X:

$$X = \frac{-EG+F\sqrt{E^2-G^2+F^2}}{E^2+F^2}$$

In other words:

$$\theta_{ki} = \sin^{-1}\left[\frac{-EG+F\sqrt{E^2-G^2+F^2}}{E^2+F^2}\right] \qquad 3\text{-}14$$

Where

$$E = l_1+l_2\sin\theta_{ki}$$

$$F = l_2\cos\theta_{ki}$$

[5]There are two signs for the roots of this equation which represent the two cases of leg up and down. The expression under the square root is always positive. A mathematical proof could be given for that.

$$G = Z_i - c_i$$

Equations 3-9 through 3-14 give the leg angles ($\psi_i, \theta_{ki}, \theta_{ki}$) as functions of the foot position expressed in body coordinates. The joint angle rates may be obtained by differentiating equations 3-9 through 3-14:

From 3-9

$$(X_i - a_i)\sin\psi_i = (Y_i - b_i)\cos\psi_i$$

Taking the derivatives of both sides:

$$(X_i - a_i)(\cos\psi_i)\dot{\psi}_i + (X_i - a_i)(\sin\psi_i)\dot{X}_i = -(Y_i - b_i)(\sin\psi_i)\dot{\psi}_i + (Y_i - b_i)(\cos\psi_i)\dot{Y}_i$$

$$-\left[(X_i - a_i)\cos\psi_i + (Y_i - b_i)\sin\psi_i\right]\dot{\psi}_i = (X_i - a_i)\sin\psi_i\dot{X}_i - (Y_i - b_i)\cos\psi_i\dot{Y}_i$$

Substituting $\cos\psi_i$ from 3-9:

$$\left[\frac{(X_i - a_i)^2 + (Y_i - b_i)^2}{(Y_i - b_i)}\right]\dot{\psi}_i = -(X_i - a_i)\dot{X}_i + (X_i - a_i)\dot{Y}_i$$

$$\dot{\psi}_i = \frac{-(Y_i - b_i)}{(X_i - a_i)^2 + (Y_i - b_i)^2}\left[(X_i - a_i)\dot{X}_i - (X_i - a_i)\dot{Y}_i\right] \qquad 3\text{-}15$$

From 3-11

$$2l_1 l_2 \sin\theta_{ki} = \frac{(Y_i - b_i)^2}{\sin^2\psi_i} + (Z_i - c_i)^2 - l_1^2 - l_2^2$$

Taking the derivatives of both sides:

$$2l_1 l_2 \cos\theta_{ki}\,\dot{\theta}_{ki} = \left[\frac{2\sin^2\psi_i(Y_i - b_i)\dot{Y}_i - (Y_i - b_i)^2(\sin 2\psi_i)\dot{\psi}_i}{\sin^4\psi_i}\right] + 2(Z_i - c_i)\dot{Z}_i$$

Therefore:

$$\dot{\theta}_{ki} = \frac{1}{2l_1 l_2\cos\theta_{ki}}\left[\frac{2\sin^2\psi_i(Y_i - b_i)\dot{Y}_i - (Y_i - b_i)^2(\sin 2\psi_i)\dot{\psi}_i + 2(Z_i - c_i)\dot{Z}_i\sin^4\psi_i}{\sin^4\psi_i}\right] \qquad 3\text{-}16$$

Similarly from 3-12

$$\dot{\theta}_{ki} = \frac{1}{2l_1 l_2\cos\theta_{ki}}\left[\frac{2\cos^2\psi_i(X_i - a_i)\dot{X}_i + (X_i - a_i)^2(\sin 2\psi_i)\dot{\psi}_i + 2(Z_i - c_i)\dot{Z}_i\cos^4\psi_i}{\cos^4\psi_i}\right] \qquad 3\text{-}17$$

Finally from 3-14

$$(E^2+F^2)\sin\theta_{hi} = -EG+F\sqrt{E^2-G^2+F^2}$$

Taking the derivatives of both sides:

$$(E^2+F^2)\cos\theta_{hi}\dot{\theta}_{hi} =$$

$$\left[-E\dot{G}-\dot{E}G+\dot{F}\sqrt{E^2-G^2+F^2}+\frac{F}{2}(E^2-G^2+F^2)^{-1/2}(2E\dot{E}-2G\dot{G}+2F\dot{F})-\sin\theta_{hi}(2E\dot{E}+2F\dot{F})\right]$$

$$\dot{\theta}_{hi} = \frac{1}{(E^2+F^2)\cos\theta_{hi}} \qquad 3\text{-}18$$

$$\left[-E\dot{G}-\dot{E}G+\dot{F}\sqrt{E^2-G^2+F^2}+\frac{F}{2}(E^2-G^2+F^2)^{-1/2}(2E\dot{E}-2G\dot{G}+2F\dot{F})-\sin\theta_{hi}(2E\dot{E}+2F\dot{F})\right]$$

Where

$$\dot{E} = l_2\cos\theta_{ki}\dot{\theta}_{ki}$$

$$\dot{F} = -l_2\sin\theta_{ki}\dot{\theta}_{ki}$$

$$\dot{G} = \dot{Z}_i$$

Equations 3-15 through 3-18 give the joint angle rates as a function of the foot velocity with respect to the hip.

These equations complete the solution of the inverse kinematic problem.

3.4.2. AUTOMATIC BODY HEIGHT, PITCH, AND ROLL REGULATION

In the kinematic control of the robot motion, it is necessary to specify both the trajectory over which the robot is to travel and the trajectory of the individual legs.

The variables in the body state vector that are associated with the body translation velocity in the forward/backward and side directions and the body azimuth rotational rate are directly updated by the animator speed control commands. Here the heading of the robot remains constant while the desired trajectory for the center of gravity of the body is provided by the animator. The body translational velocity components ($\dot{X}_b$, $\dot{Y}_b$, $\dot{Z}_b$) and the body azimuth rotation rate are updated such that: if there is a difference in the current velocity or orientation and the value given by the animator, then the robot accelerates or

decelerates to remove the difference. The exact mathematical relationships are:

$$\begin{aligned} \ddot{X}_b &= K_1(\dot{X}_b - \dot{X}_d) \\ \ddot{Y}_b &= K_2(\dot{Y}_b - \dot{Y}_d) \\ \ddot{Z}_b &= K_3(\dot{Z}_b - \dot{Z}_d) \\ \dot{\psi}_b &= K_4(\psi_b - \psi_d) \end{aligned} \qquad 3\text{-}19$$

Where $\dot{X}_d, \dot{Y}_d, \dot{Z}_d, \psi_d$ are the animator's desired velocities and rotation angles, and K_i are constants.

The body height above the supporting surface and the body pitch and roll angles are determined by fitting an approximate plane to the points of support by a least squares method. The body, then, is commanded to be parallel to this estimated plane of the feet and at a constant perpendicular height above it. To calculate the approximating plane of the supporting feet, the coordinates of the feet are computed from the automatic leg positioning routine. For each of the four legs, a four-element vector is defined:

$$(X_{iE}, Y_{iE}, Z_{iE}, L_i) \quad \text{for } i = 1,2,3,4 \qquad 3\text{-}20$$

Where (X_{iE}, Y_{iE}, Z_{iE}) is the position of foot i expressed in earth fixed coordinates (see figure 3.5). L_i is an indicator of foot position such that L_i=1 means that foot i is on the ground, L_i=0 means that foot i is off the ground.

The least squares regression method that is used here is based on a multiple regression with two independent variables as a sequence of straight-line regressions [DrS66].

$$\hat{Z}_{iE} = \alpha_0 + \alpha_1 X_{iE} + \alpha_2 Y_{iE}$$

The problem now is to find estimates for the α's (the regression parameters). First we regress Z_{iE} on X_{iE}. This straight line regression is represented as:

$$\hat{Z}_{iE} = \hat{a}_0 + \hat{a}_1 X_{iE} \qquad 3\text{-}21$$

Where:

$$\hat{a}_0 = \bar{Z} - \hat{a}_1 \bar{X}$$

$$\hat{a}_1 = \frac{\sum_{i=1}^{4} L_i (X_{iE} - \bar{X})(Z_{iE} - \bar{Z})}{\sum_{i=1}^{4} L_i (X_{iE} - \bar{X})^2}$$

$$\bar{X} = \sum_{i=1}^{4} \frac{L_i X_{iE}}{n}$$

$$\bar{Z} = \sum_{i=1}^{4} \frac{L_i Z_{iE}}{n}$$

and

$$n = \sum_{i=1}^{4} L_i$$

Then we regress Y_{iE} on X_{iE}. This straight line regression is represented as:

$$\hat{Y}_{iE} = \hat{a}_2 + \hat{a}_3 X_{iE} \qquad 3\text{-}22$$

Where

$$\hat{a}_2 = \bar{Y} - \hat{a}_3 \bar{X}$$

$$\hat{a}_3 = \frac{\sum_{i=1}^{4} L_i (X_{iE} - \bar{X})(Y_{iE} - \bar{Y})}{\sum_{i=1}^{4} L_i (X_{iE} - \bar{X})^2}$$

and

$$\bar{Y} = \sum_{i=1}^{4} \frac{L_i Y_{iE}}{n}$$

Finally, $(Z_{iE} - \hat{Z}_{iE})$ is regressed against $(Y_{iE} - \hat{Y}_{iE})$ by the following straight line fitting:

$$(Z_{iE} - \hat{Z}_{iE}) = \hat{a}_4 (Y_{iE} - \hat{Y}_{iE}) + \hat{a}_5 \qquad 3\text{-}23$$

Where[6]

$$\hat{a}_4 = \frac{\sum_{i=1}^{4} L_i \left[(Y_{iE} - \hat{Y}_{iE}) - \overline{(Y_{iE} - \hat{Y}_{iE})} \right] \left[(Z_{iE} - \hat{Z}_{iE}) - \overline{(Z_{iE} - \hat{Z}_{iE})} \right]}{\sum_{i=1}^{4} L_i \left[(Y_{iE} - \hat{Y}_{iE}) - \overline{(Y_{iE} - \hat{Y}_{iE})} \right]^2}$$

[6]Since we are using two sets of residuals whose sums are zero, thus the line must pass through the origin. (So if we did put $\hat{a}_5$ term in, we shall find $\hat{a}_5 = 0$ in any case).

$$\overline{Y_{iE}-\hat{Y}_{iE}} = \sum_{i=1}^{4} \frac{L_i(Y_{iE}-\hat{Y}_{iE})}{n}$$

$$\overline{Z_{iE}-\hat{Z}_{iE}} = \sum_{i=1}^{4} \frac{L_i(Z_{iE}-\hat{Z}_{iE})}{n}$$

By substituting for $\hat{Z}_{iE}$ and $\hat{Y}_{iE}$ in 3-23, we get the estimated plane equation:

$$(Z_{iE}-(\hat{a}_0+\hat{a}_1X_{iE})) = \hat{a}_4(Y_{iE}-(\hat{a}_2+\hat{a}_3X_{iE}))$$

or

$$\hat{Z}_{iE} = \hat{a}_0+(\hat{a}_1-\hat{a}_4\hat{a}_3)X_{iE}+\hat{a}_4Y_{iE}-\hat{a}_2\hat{a}_4 \qquad 3\text{-}24$$

The next step is to express the unit vector normal to this plane, $\vec{N}$, in two ways and equate the components. A unit vector normal to the desired body orientation will also be normal to the estimated surface and thus have a body Z component only. That is, $\vec{N}$, may be expressed in the earth fixed coordinate system as:

$$\vec{N} = T_{Ib}^{T}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}\sin\psi\sin\phi+\cos\psi\sin\theta\cos\phi\\ \sin\psi\sin\theta\cos\phi-\cos\psi\sin\phi\\ \cos\phi\cos\theta\end{bmatrix} \qquad 3\text{-}25$$

Another representation for this same unit vector that is normal to the plane of the feet can be obtained from 3-24 (by taking the partial derivatives w.r.t. X_{iE}, Y_{iE}, Z_{iE} and dividing by the length of the vector):

$$\vec{N} = \begin{bmatrix}\dfrac{-(\hat{a}_1-\hat{a}_4\hat{a}_3)}{\sqrt{1+(\hat{a}_1-\hat{a}_4\hat{a}_3)^2+\hat{a}_4^2}}\\ \dfrac{-\hat{a}_4}{\sqrt{1+(\hat{a}_1-\hat{a}_4\hat{a}_3)^2+\hat{a}_4^2}}\\ \dfrac{1}{\sqrt{1+(\hat{a}_1-\hat{a}_4\hat{a}_3)^2+\hat{a}_4^2}}\end{bmatrix} = \begin{bmatrix}P\\Q\\R\end{bmatrix} \qquad 3\text{-}26$$

Equating equations 3-25 and 3-26 gives:

$$\cos\phi\cos\theta = R$$

$$\sin\psi\sin\theta\cos\phi-\cos\psi\sin\phi = Q \quad (\textit{multiply both sides by } \cos\psi)$$

$$\sin\psi\sin\phi+\cos\psi\sin\theta\cos\phi = P \quad (\textit{multiply both sides by } \sin\psi)$$

Subtracting the last two equations:

$$\sin\phi = P\sin\psi-Q\cos\psi$$

Therefore:

$$\phi = \sin^{-1}(P\sin\psi - Q\cos\psi) \qquad 3\text{-}27$$

From the second $\vec{N}$ scalar equation:

$$\theta = \sin^{-1}\left[\frac{Q + \cos\psi\sin\phi}{\sin\psi\cos\phi}\right] \qquad 3\text{-}28$$

From the first $\vec{N}$ scalar equation:

$$\theta = \sin^{-1}\left[\frac{P - \sin\psi\sin\phi}{\cos\psi\cos\phi}\right] \qquad 3\text{-}29$$

Note that these values for θ and ϕ should be the steady state values for the body to be parallel to the supporting plane. For a given value of the body azimuth angle, ψ, the roll and pitch of the feet plane, ϕ and θ, which determine the desired orientation of the body are given by the equations 3-27 to 3-29.

By driving the kinematics algorithms described in the last two sections, the robot will follow the desired trajectory and maintain its sustained stable locomotion kinematically.

Both the automatic leg positioning and automatic body regulation algorithms make the robot highly maneuverable for extremely rough terrain situations. However, each leg has a limited reach, due to the leg's kinematic limits. The position of any abnormally extending leg is monitored to insure it is never extended beyond the kinematic limits. With the automatic body regulation algorithm, if this extending leg reaches the kinematic limits, then the robot body is automatically commanded to move in such a direction as to accommodate the desired motion of that individual leg. This accommodation increases the ability of the robot to reach a desired foothold. Of course, the position of all of the support legs must be monitored during this accommodation movement to insure that the body movement does not extend a support leg beyond its kinematic limits. Also, the robot stability must be monitored to insure that the body is not shifted to a position where the robot is unstable. This concept of automatic body regulation is analogous to the situation where a human may lean his body in such a manner as to help him reach his hand or foot to a desired position.

The kinematic algorithms of the JCC have been implemented in the experimental system. The evaluation of the motion produced is given in chapter 6.

CHAPTER 4

THE NUMERICAL CONTROLLER (NC)

4.1. INTRODUCTION

Not only do legged articulated bodies contain closed kinematic chains (chains closed through the ground), but a number of rather complicated sequences of them may be formed and broken many times in the course of locomotion. In such systems, in addition to the *intralimb* coordination problem, there is an *interlimb* coordination problem. That is, cooperative action among the limbs is essential. One of the most difficult problems in coordinating the limbs in such systems is the necessity to actively control the assignment of forces and torques within the mechanism. Failure to do so may result in an unrealistic appearance of the movement and/or a "binding effect" of the legs (one leg dragging another). Given desired trajectories for the body and legs (the outputs from the JCC), the numerical controller (NC) of the proposed locomotion control system solves dynamically the problem of efficiently driving the four-legged robot under the closed kinematic chains conditions to produce the desired motion. In the absence of error, if torques and forces are generated continuously throughout the motion of the robot, then the robot can follow the desired reference trajectory.[1]

This chapter investigates the dynamics of the robot's motion and develops the algorithms of the NC. These algorithms involve setting and solving a closed chain inverse dynamic problem, a linear programming problem, and a direct dynamics problem. In section 4.2.1 the articulated robot model of section 3.3 has been extended to incorporate various dynamic variables, then in section 4.2.2 the inversion of Armstrong's direct dynamics equations for open kinematic chains [Arm79] is performed and the need for

[1]The proposed system is an open-loop control model in which movements are executed without regard for the effects that they may have on the environment. In a closed-loop model, all movements are made by comparing an ongoing feedback from the links during motion to a reference of correctness (the desired motion). The kind of errors that might be responsible for not achieving the desired motion could be errors due to numerical inaccuracy, problems with the motion planning, or impossible motions (these are motions which are kinematically possible but not dynamically possible given only the control at the joints).

extending the inverted equations to the case of closed kinematic chains in order to provide a means for modeling locomotion tasks is explained. Section 4.3 provides a mathematical approach to the problem of force and torque assignment in the four-legged robot. The approach is based on the Newton-Euler formalism for open kinematic chain mechanisms derived in section 4.2 and requires the setting and solution of a linear programming problem. In section 4.4 the structure of the numerical controller (NC) that calculates the forces and torques needed for a given body and leg trajectories is described. The NC deals with the legs that are in contact with the ground by solving the linear programming problem formed in section 4.3. The legs that are not in contact with the ground form open kinematic chains and therefore are dealt with using the open chain inverse dynamics formalism derived in section 4.2.2. Section 4.5 analyzes the high degree of parallelism inherent in the NC computations and presents a mathematically exact formulation for a near real-time solution of the locomotion control of the four-legged articulated robot.

4.2. THE DYNAMICS OF THE ROBOT MOTION

One of the main problems of modeling articulated bodies using dynamic analysis is controlling and coordinating the motion (see section 2.2). The users of these systems are supposed to provide control functions to specify forces and torques acting on individual degrees of freedom. The force/torque control is not familiar to users, who therefore have no idea about the torques and forces required to produce the motion they want.

A solution to this problem introduced in [Wil86] suggested that the user describe the motion at the joints as kinematic motion changes (joint rotations vs. time and joint translations vs. time- similar to the outputs of the JCC described in the previous Chapter) and have the program calculate appropriate forces and torques to accomplish the motion. Simple equations were used for these calculations:

For sliding joints:

$$\Delta x = x_d - x_p$$

Where x_d is the desired position at the next time sample (or next animation frame), and x_p is the present position.

$$\Delta v = \frac{\Delta x}{\Delta t} - v_p$$

Where Δt is the time between samples (or equivalently animation frames), and v_p is the present velocity. Δv is the amount the velocity must be altered to achieve the desired position at the next time sample.

$$f = m\frac{\Delta v}{\Delta t}$$

f is the estimated force that must be applied to achieve this. m is the mass of the segment(s) distal to this degree of freedom.

For revolute joints the same formulas were used, but the velocities are angular:

$$N = I\frac{\Delta \omega}{\Delta t}$$

N is the estimated torque, I is the inertia of the link, and $\Delta\omega$ is the amount the angular velocity must be altered to achieve the desired rotation at the next time sample. A similar solution was proposed by Armstrong, Green, and Lake [AGL86] where the user had to specify the new angle between the link and its parent (θ_d) and the program computes the sequence of torques (one for each time step of the dynamic calculation- again the time step is proportional to animation frame) to produce a smooth motion. The equations that were used to estimate these torques were:

$$T(\theta_0,\theta_d) = \alpha(e^{\beta(\theta_d - \theta_0)} - 1) \quad \text{if } \theta_0 \geq \theta_d$$

$$= -\alpha(e^{\beta(\theta_0 - \theta_d)} - 1) \quad \textit{otherwise}$$

$$\textit{mid point angle} = \theta_m = \frac{\theta_0 + \theta_d}{2}$$

with each iteration after the mid-point, the maximum-step torque is reduced using the following equation:[2]

$$T = T_{max}\left(\frac{\theta_c - \theta_d}{\theta_m - \theta_d}\right)^{\frac{1}{2}}$$

where θ_c is the current angle, θ_0 is the initial angle, α, β are constants, T_{max} is the maximum-step torque, and θ_m is the mid-point angle.

[2]This is a bang-bang control.

The problem with the previous solutions is that the estimated forces and torques are not produced from dynamically-based coupled calculations, so the resulting motion might not satisfy the requirements for realism. Estimating the forces and torques is actually quite a complicated process because of the high degree of nonlinearity inherent in articulated bodies. It is not acceptable to estimate the values of forces and torques needed to manipulate individual joints without taking into account the extensive coupling of forces and torques from adjacent links. These effects can not be neglected especially when dealing with rhythmic motions like locomotion (but neglecting them had no serious effect on the "force-effect" types of movements mentioned in section 2.2).

For example, the torques applied must compensate for the inertia of the links, gravitational force, the Coriolis and centrifugal forces, and viscous friction at the joints. All these terms vary in a highly nonlinear fashion depending on the configuration at a given point in time. Moreover, locomotive motion needs special care, since it imposes a new inter-limb coordination problem in addition to the intra-limb coordination problem. The legs form and break many chains during their cycles, and one should be extremely careful during the locomotion control computations about treating each case as it arises.

In order to avoid all these problems, the proposed NC suggests the use of articulated dynamics analysis to calculate the joint forces and torques that would produce the desired motion (that is, to solve the inverse dynamics problem). In this way, the animation system is forced to create a motion that is produced by dynamically-based limitations on forces and torques, and the resulting motion will satisfy these obvious requirements for realism.

4.2.1. THE ARTICULATED ROBOT DYNAMICS MODEL

Before deriving the inverse dynamic equations that calculate the joint forces and torques that would produce the desired motion, the articulated robot model of section 3.3 has to be extended to incorporate dynamics variables (Figure 4.1). The robot links are numbered such that the body is numbered 0, while the individual links of each leg are assigned two numbers, i and k, where i is the number of the link within a leg and k is the number of the leg containing that link. Several force and torque vectors are also introduced in

the model:

f_k^{n+1} is the reaction force from the ground to the tip of leg k (force applied from a hypothetical link n+1 to link n) expressed in the inertial frame. Its components are $[f_{kx}^{n+1}\ f_{ky}^{n+1}\ f_{kz}^{n+1}]^T$. These are four (k=1,2,3,4) 3×1 vectors (note n=2 for all legs).

g_k^{n+1} is the reaction torque from the ground to the tip of leg k (torque applied from a hypothetical link n+1 to link n) expressed in the inertial frame. Its components are $[g_{kx}^{n+1}\ g_{ky}^{n+1}\ g_{kz}^{n+1}]^T$. These are four (k=1,2,3,4) 3×1 vectors.[3]

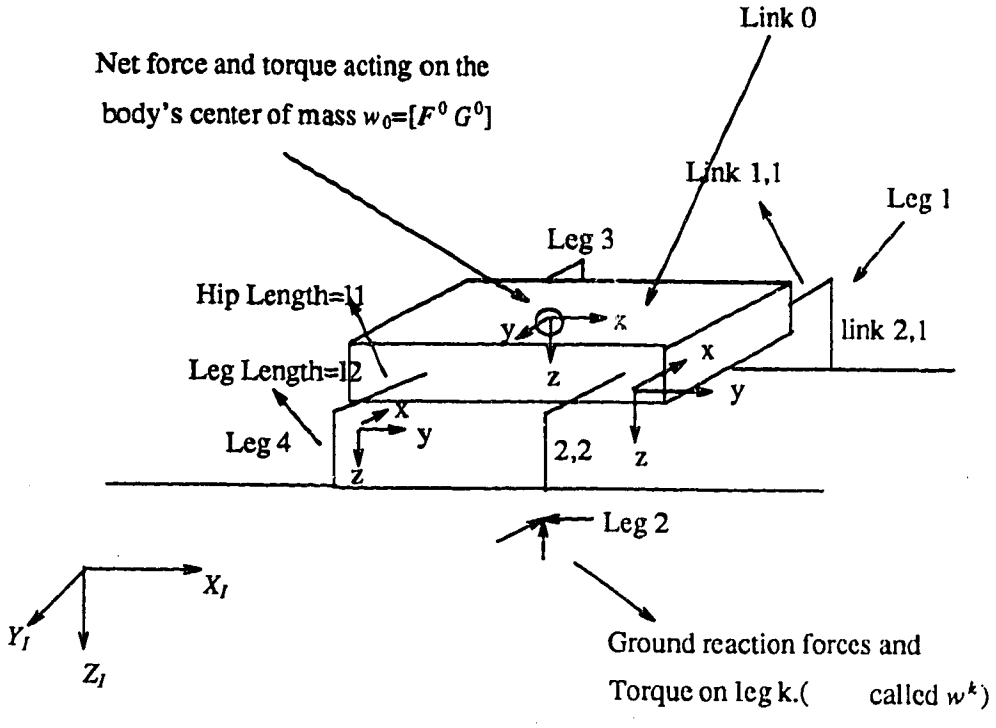

Figure 4.1 The Dynamics Model of the Four-Legged Articulated Robot

[3]As the kinematic limits of the robot joints are represented as restorative torques at the joints (see Section 6.3), it is part of studying the robot in static equilibrium to examine the supports in order to deduce what types of force they apply to the robot. These forces are called constraint forces because they constrain (restrict) the motion of the robot. The characteristics of these constraint forces can be established by considering the manner in which the robot is supported. In determining how to depict the constraint forces we need only

w^k is the reaction force and torque from the ground to the tip of leg k. It consists of two components, the force and the torque (f_k^{n+1} and g_k^{n+1}). These are four 6×1 vectors.

F^0 is the net force acting on the body (with components given in the inertial coordinate system). Its components are $[F_x^0 F_y^0 F_z^0]^T$. This is one 3×1 vector.

G^0 is the net torque acting on the body (with components given in the inertial coordinate system). Its components are $[G_x^0 G_y^0 G_z^0]^T$. This is one 3×1 vector.

w^0 is the net force and torque acting on the body by the legs. It consists of the two components F^0 and G^0. This is one 6×1 vector.

F_k^i is the net force acting on link i of leg k. These are eight (k=1,2,3,4 and i=1,2) 3×1 vectors.

G_k^i is the net torque acting on link i of leg k. These are eight (k=1,2,3,4 and i=1,2) 3×1 vectors.

Articulated bodies with tree structures (open kinematic chain systems) are less frequent in practice than systems with closed chains (such as our articulated robot). However, any system with closed chains can be transformed into a system with a tree structure (called its reduced system) by cutting suitably selected hinges [Wit77]. Thus in order to obtain the equations of motion for our four-legged robot, all that is necessary is to add internal forces and kinematic constraints, for the cut hinges, to the equations of motion for its reduced system.

Figure 4.2 shows how the robot's reduced system can be represented as a tree linkage structure (open kinematic chain system). Following the assumption in [Arm79] and for ease of analysis, the joints of this structure are assumed to allow three rotational degrees of freedom (instead of 2 DOF hips and 1 DOF knees of section 3.3 model). This corresponds to a ball and socket joint. The reason for this assumption is that it is more difficult to formulate the equations of motion in the presence of constraints which reduce the

---

two facts. First, because a force represents a pushing or pulling effect, a point in a foot is prevented from moving in a specific direction by the action of a reaction force in the opposite direction. Second, because a couple represents a twisting effect, a leg is prevented from rotating about a specific axis by the action of a reaction couple whose torque is parallel to the axis of rotation, twisting opposite the sense in which rotation would occur. Because the ground is assumed rigid and we don't want the robot to collapse or its legs to slip on the ground, it follows that there can be no rotation about any of the coordinate axes. This means that there must be a reaction couple whose components $[g_{kx}^{n+1}, g_{ky}^{n+1}, g_{kz}^{n+1}]^T$ for leg k represent the constraints necessary to prevent leg k's rotations. We usually show the components of general reaction forces and couples in order to be certain that we remember to account for each unknown component.

number of rotational degrees of freedom.

A typical linkage of the robot's linkage structure is shown in Figure 4.3. The Figure shows the quantities associated with a typical link r (any one of the nine links in Figure 4.2). Let the following quantities be associated with the typical link r. Here vectors, unless otherwise noted, are represented in a frame attached to link r located at the proximal joint of link r and moving with link r (this is similar to the $(X',Y',Z')_i$ and $(X'',Y'',Z'')_i$ coordinates of Figure 3.1); see Figure 4.3. m_r is the mass of link r; a_G is the acceleration of gravity (in the inertial frame X_I, Y_I, Z_I); f_E^r is an external force (inertial frame) acting on link r at the point p_E^r (the vector from the proximal hinge of link r to the point of application of the external force f_E^r to link r); g_E^r is an external torque (inertial frame) acting on link r; a^r is the linear acceleration of the proximal (parent) hinge of link r; r^r is the linear acceleration of the center of mass of link r; ω^r is the angular velocity of link r; c^r is the vector from the proximal hinge to the center of mass of link r; f^r and g^r are the force and torque which link r exerts on its parent at the proximal hinge; l^s is the vector from the proximal hinge of link r to the proximal hinge of child s of link r.

The following quantities are transformation matrices that are also associated with link r. R^r converts vector representations in the frame of link r to the representations in the frame of the parent link (this has

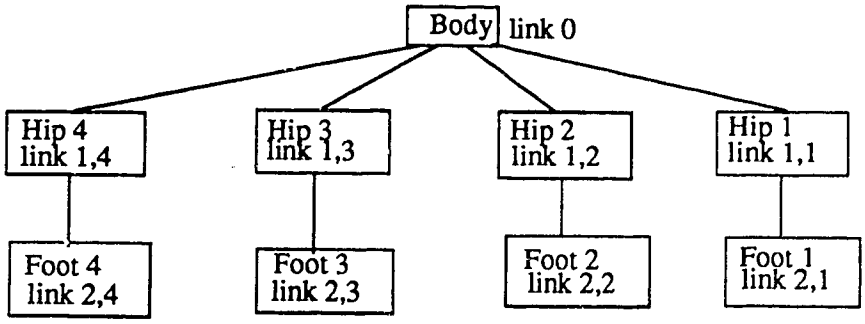

Figure 4.2 The Robot Reduced Open-Chain Linkage Structure

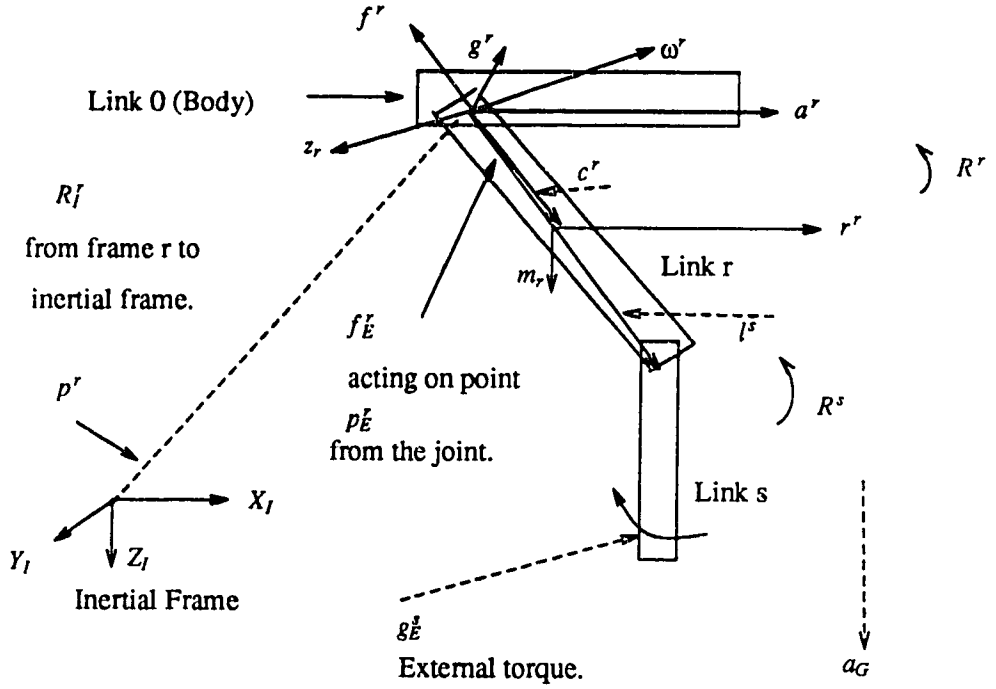

Figure 4.3 A Typical Linkage of the Robot's Linkage Structure

the form of T_{bA}^T matrix of equation 3-5); R_I^r converts to representations in the inertial frame (this has the form of T_{Ib}^T in equation 3-1); J^r is the 3×3 moment of inertia matrix of link r about its proximal hinge; S_r is the set of all links having link r as parent (there is always one link in our case).

4.2.2. DERIVING THE INVERSE DYNAMICS EQUATIONS

In 1979 Armstrong described an O(n) recursive method for solving the direct dynamics problem in open chain kinematic mechanisms [Arm79]. His method has been used in the simulation of the Space Shuttle Remote Manipulator System (CANADARM) [AGL86], and in the animation of the "force-effect" type dynamics movements mentioned in section 2.2 [ArG85]. In this section, these equations are inverted in order to use them in estimating the force and torque profiles needed to produce different types of locomotive motions in the four-legged robot.

The following are Armstrong's direct dynamics equations:

The first equation expresses the fact that the rate of change of angular momentum of link r is equal to the applied torques from various sources:

$$J^r\dot{\omega}^r = -\omega^r\times(J^r\omega^r) - g^r + \sum_{s\in s.} R^s g^s + R_f^{r^T} g_E^r$$

$$+ P_E^r\times R_f^{r^T} f_E^r + m_r c^r\times R_f^{r^T} a_G - m_r c^r\times r^r + \sum_{s\in s.} l^s\times R^s f^s \qquad 4\text{-}1$$

The term $J^r\dot{\omega}^r$ represent the rate of change of angular momentum of link r (note $\frac{d}{dt}(J^r\omega^r)=J^r\dot{\omega}^r+\omega^r\times(J^r\omega^r)$). The term $-\omega^r\times(J^r\omega^r)$ is a torque coming from the rotation of the frame r with angular velocity ω^r which causes the angular momentum, $J^r\omega^r$, to appear to rotate. This torque term would not appear if the equations had been formulated in the inertial frame. In the inertial frame, however, the inertia matrix J^r would not be constant, making that frame less appropriate for formulating the equations for purposes of computer simulation.

The torque $-g^r$ is the negative of the torque which, by definition, link r exerts on its parent at its proximal hinge. It is the parent's reaction which is "equal and opposite". To be added in next are the torque terms coming from the son links, which must be converted from their representations in the sons' frames, and the external torque, which must be converted from its representation in the inertial frame. It doesn't matter where the torques are applied, since the links are considered rigid. The force of gravity, $m_r a_G$, converted from the inertial frame, causes a torque at the proximal hinge of link r when applied at its center of mass; and similarly for the external force f_E^r acting at P_E^r.

The term $-m_r c^r\times r^r$ comes from the fact that the frame r in which this equation is formulated is accelerating with respect to the inertial frame, giving the effect of the force $-m_r r^r$ applied at the center of mass of the link. In the next term, the forces f^s coming from all the sons of link r are first transformed from the coordinates of the son link, where they are represented, to the frame of r by applying R^s. Then the cross product of l^s with the force gives the torque at the proximal hinge of link r due to the force from the son link.

The next equation gives the force f^r acting on the parent of link r at the proximal hinge of link r:

$$f^r = R_i^{rT}(f_E^r + m_r a_G) + \sum_{s \in s_r} R^s f^s - m_r \omega^r \times (\omega^r \times c^r) + m_r c^r \times \dot{\omega}^r - m_r a^r \qquad 4\text{-}2$$

The first term shows the contribution from the external force and gravity acting on the link. The second term represent the forces from the son links as they are converted to frame r and communicated to the parent at the hinge. The term $-m_r \omega^r \times (\omega^r \times c^r)$ represent the centrifugal force from the rotation of the frame of link r. The term $m_r c^r \times \dot{\omega}^r$ comes from the fact that the frame r is rotating at an acceleration angular velocity, which causes the center of mass to accelerate with respect to the inertial frame. Finally, the term $-m_r a^r$ comes from the fact that the frame r is accelerating.

The next equation relates the acceleration at the proximal hinge of a son link s of link r to the linear and angular acceleration at the proximal hinge of r:

$$R^s a^s = \omega^r \times (\omega^r \times l^s) + a^r - l^s \times \dot{\omega}^r \qquad 4\text{-}3$$

The next equations hypothesize a linear relationship between the linear acceleration a^r of a link and the amount of angular acceleration it undergoes and between a^r and the reactive force on the parent:

$$\dot{\omega}^r = K^r a^r + d^r \qquad 4\text{-}4$$

$$f^r = M^r a^r + f'^r \qquad 4\text{-}5$$

If one imagined giving a configuration of links, distal to a certain hinge, a push at the hinge, this would cause a certain acceleration a^r, which consequently would cause a certain angular acceleration $\dot{\omega}^r$, and a certain reactive force f^r on the parent. The relation among these quantities are assumed to be linear as in 4-4 and 4-5. Armstrong assumed this linearity for all the sons of link r, and developed a computational method for solving the equations of motion by calculating the "recursive" coefficients of the linear relations, K^r, d^r, M^r, and f'^r.

The solution of the equations gives the following recursive coefficients:

$$K^r = (J^r + \sum_{s \in s_r} \tilde{l}^s R^s M^s R^{sT} \tilde{l}^s)^{-1} (\sum_{s \in s_r} \tilde{l}^s R^s M^s R^{sT} - m_r \tilde{c}^r) \qquad 4\text{-}6$$

$$d^r = (J^r + \sum_{s \in s_r} \tilde{l}^s R^s M^s R^{sT} \tilde{l}^s)^{-1} (-\omega^r \times (J^r \omega^r) - g^r + \sum_{s \in s_r} R^s g^s \qquad 4\text{-}7$$

$$+ R_I^{rT} g_E^r + m_r c^r \times R_I^{rT} a_G + p_E^r \times R_I^{rT} f_E^r + \sum_{s \in S_r} \tilde{l}^s R^s (f'^s + M^s R^{sT} (\omega^r \times (\omega^r \times l^s)))$$

$$M^r = -m_r I + m_r \tilde{c}^r K^r + \sum_{s \in S_r} R^s M^s R^{sT} (I - \tilde{l}^s K^r) \qquad 4\text{-}8$$

$$f'^r = -m_r \omega^r \times (\omega^r \times c^r) + R_I^{rT} (f_E^r + m_r a_G) + m_r c^r \times d^r \qquad 4\text{-}9$$

$$+ \sum_{s \in S_r} (R^s f'^s + R^s M^s R^{sT} (\omega^r \times (\omega^r \times l^s) - l^s \times d^r))$$

The tilde operator of a 3-vector v, with components v_1, v_2, v_3 produces a 3×3 matrix V such that for any 3-vector w, the vector $v \times w$ is equal to the product of V and the column-vector w.

$$\tilde{v} = V = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

In the solution K^r and M^r are 3×3 matrices, whereas d^r and f'^r are 3×1 vectors.

In summary, Armstrong's equations of motion for each link are formulated in terms of a moving frame attached to the proximal hinge of the link, and consist of equations giving the effect of torques (4-1), equations giving the effect of forces (4-2), equations relating the accelerations at parent and son nodes (4-3), and equations relating linear and angular accelerations and the produced reactive force (4-4 and 4-5). In order to solve these equations to get the motion (i.e. accelerations, velocities, positions, and orientations of the links through time) given the torques at the hinges and the external forces and torques: at each time step, the external forces f_E^r, the external torques g_E^r, and the hinge torques g^r are set according to the control parameters to produce the desired motion. Then one solves for K^r, d^r, M^r, and f'^r, starting at the leaves of the tree of links (Figure 4.2) and proceeding towards the root (the robot body). This is called the "inbound" phase of the computations. Then, using the fact that the root link (link number 0 - the body) is not subject to forces and torques from its parent (since it has no parent), one can solve the equations to get the linear and angular accelerations of all links starting from the root of the linkage tree towards the leaves. This is called the "outbound" phase of the computations.

Intuitively, this solution method can be understood as follows. Suppose some agent accelerates a certain point on a rigid body r by an amount a^r. Then there will be a certain change of the angular velocity vector (represented in the frame of link r) given by the derivative $\dot{\omega}^r$. At the same time, there will be a force

f^r (represented in frame r) acting upon the agent. The agent is, in this case, the parent link of link r. In the absence of any child links, the equations of motion will suffice to determine $\dot{\omega}^r$ and f^r as a function of a^r. The function will be linear as expressed in equations 4-4 and 4-5 through the coefficients K^r, d^r, M^r, and f'^r (equations 4-6 through 4-9). If there are child links, then link r will act as an agent to accelerate them. Assuming that the coefficients of 4-4 and 4-5 for the child are known, and without knowing any of the accelerations, the coefficients for link r can be computed [ArG85]. In this manner the coefficients for all links can be computed in an *inbound* pass towards the root link. Then, using the fact that the root is not subject to forces and torques from any parent link in the tree, one can solve the equations to get the linear and angular accelerations of all links in an *outbound* pass toward the leaves.

The recursive structure of this direct dynamic calculation applied to the reduced system of the four-legged articulated robot of figure 4.2 is shown as inbound-outbound directed trees in Figure 4.4 (both the inbound and the outbound computations are separated). In the inbound tree, a node must receive data from all its descendants (children) before it can perform its computation and send the results to its parent. The computation of the recursive coefficients K^r, d^r, M^r, and f'^r is done in this inbound tree. Then the outbound tree propagates the linear and angular acceleration of the links and performs their integration.

In the figure, tree nodes represent total processing associated with each link, and recursion is implemented by connecting each node's output to the next node's input.

The linear computation time coefficient is determined by the longest time required to propagate any single variable across the nodes. In determining this time, only K^r, M^r, d^r, f'^r, a^r are to be considered, since other variables are not propagated up or down the recursion chain. The computation structure of Figure 4.4 is used in the animation subsystem (ANS) as will be described in Chapter 6.

Meanwhile, Armstrong's direct dynamics equations are inverted in the following paragraphs.

$$J^r\dot{\omega}^r = -\omega^r\times(J^r\omega^r) - g^r + \sum_{s\in s_r} R^s g^s + R_I^{rT} g_E^r$$

$$+ P_E{}^r\times R_I{}^{rT} f_E{}^r + m_r c^r\times R_I{}^{rT} a_G - m_r c^r\times r^r + \sum_{s\in s_r} l^s\times R^s f^s$$

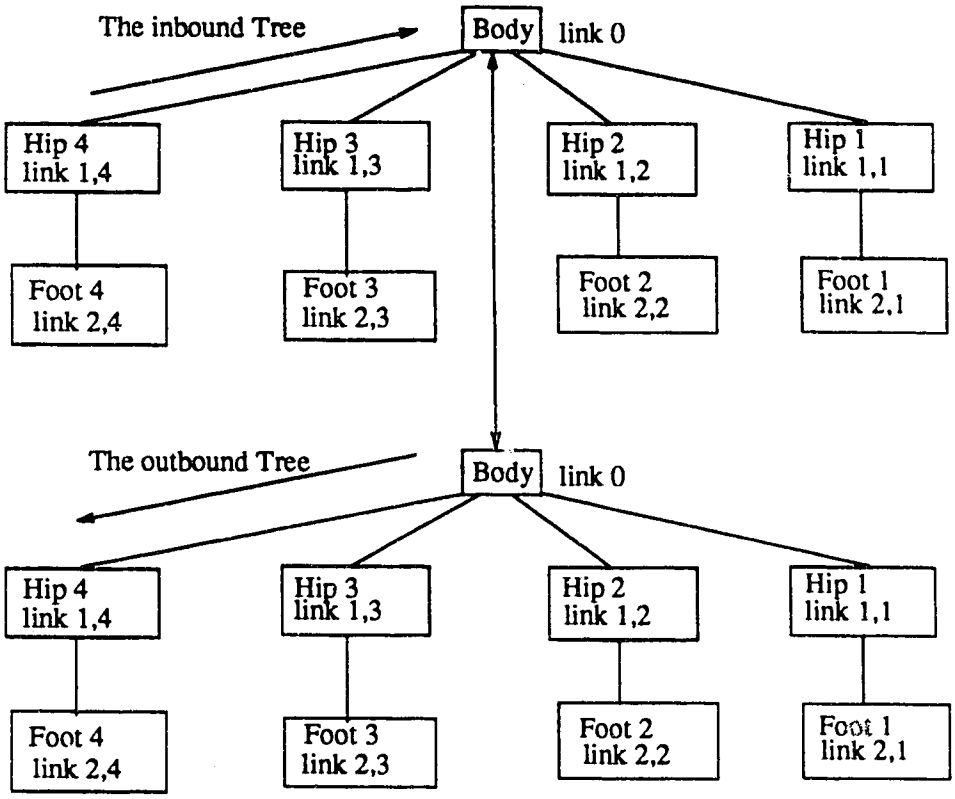

Figure 4.4 The Inbound-Outbound Directed Trees for the Direct Dynamics Calculations

Put it in the form

$$J^r\dot{\omega}^r + \omega^r \times (J^r\omega^r) = N^r$$

which is the Euler's equation that relates the angular velocity and acceleration to the total torque N^r acting on link r with inertia J^r, where

$$N^r = -g^r + \sum_{s \in s_r} R^s g^s + \sum_{s \in s_r} l^s \times R^s f^s + R_f^{rT} g_E^r$$

$$+ P_E^r \times R_f^{rT} f_E^r + m_r[-c^r \times r^r + c^r \times R_f^{rT} a_G]$$

Therefore,

$$g^r = \sum_{s \in s_r} R^s g^s - N^r + \sum_{s \in s_r} l^s \times R^s f^s + R_f^{rT} g_E^r + c^r \times F^r$$

which is the amount of torque that link r exerts on its parent at the proximal joint.

$$F^r = -m_r r^r + R_I{}^{rT}(f_E{}^r + m_r a_G)$$

assuming that $P_E^r = c^r$.

Then from the force equation 4-2:

$$f^r = R_I^{rT}(f_E^r + m_r a_G) + \sum_{s \in s_r} R^s f^s - m_r \omega^r \times (\omega^r \times c^r) + m_r c^r \times \dot{\omega}^r - m_r a^r$$

which can be put in the form

$$f^r = F^r + \sum_{s \in s_r} R^s f^s$$

which is the force that link r exerts on its parent at the proximal joint, where

$$F^r = -m_r \omega^r \times (\omega^r \times c^r) + m_r c^r \times \dot{\omega}^r - m_r a^r + R_I^{rT}(f_E^r + m_r a_G)$$

Now we want to move the coordinates from the joints to the links' centers of mass. This is done by using a known transformation procedure [LWP76]. Let O' be situated at the center of mass and O^* be situated at the hinge as in Figure 4.5 and assume:

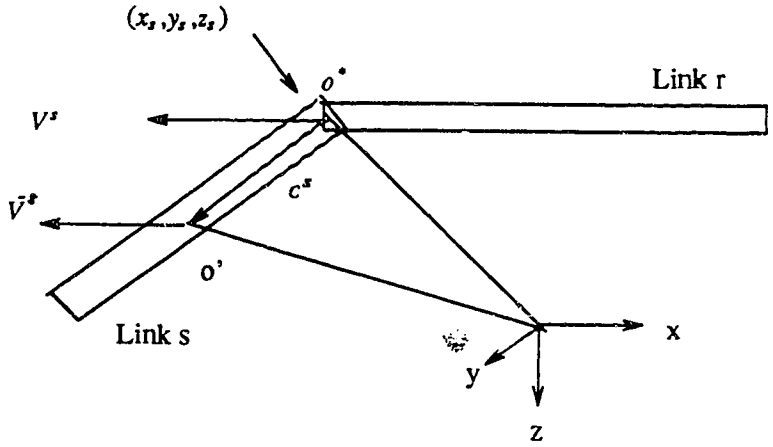

Figure 4.5 Moving the Coordinates from the Joints to the Links' Centers of Masses

$\bar{V}^s$ is the linear velocity of the center of mass of link s with respect to the base coordinate (r^s is its acceleration).

V^s is the linear velocity of the frame of link s with respect to the base coordinate (a^s is its acceleration).

From the Figure,

$$\bar{V}^s = \frac{\partial^* c^s}{\partial t} + \omega^s \times c^s + V^s$$

where $\frac{\partial^*}{\partial t}$ represents the time derivative with respect to the moving frame r.

$$r^r = \frac{\partial^{*2} c^s}{\partial t^2} + \dot{\omega}^s \times a^s + 2\omega^s \times \frac{\partial^* s}{\partial t} + \omega^s \times (\omega^s \times c^s) + a^s$$

where the term $2\omega^s \times \frac{\partial^* s}{\partial t}$ represents the Coriolis acceleration and $\omega^s \times (\omega^s \times c^s)$ represents the centrifugal acceleration. Putting $\frac{\partial^* c^s}{\partial t} = 0$, and $\frac{\partial^{*2} c^s}{\partial t^2} = 0$ since the coordinate of frame s is fixed on link r, we get

$$\bar{V}^s = \omega^s \times c^s + v^s$$

$$r^s = \dot{\omega}^s \times c^s + \omega^s \times (\omega^s \times c^s) + a^s$$

Comparing this to the second F^r equation, we get

$$F^r = -m_r r^r + R_I{}^{rT}(f_E{}^r + m_r a_G)$$

which is the first F^r equation.

Now in the direct dynamics, one relates the links' linear accelerations as follows: as the root (base) moves with acceleration a^0, it acts as an agent to accelerate its children (s) by giving them acceleration (a^s)- according to equation 4-3:

$$R^s a^s = \omega^r \times (\omega^r \times l^s) + a^r - l^s \times \dot{\omega}^r$$

But as the goal in the inverse dynamics computations is to relate the linear acceleration of each link (s) to the linear acceleration of the base (r), one can relate the distal links' linear accelerations to the base by the relation:

For all[4] s ε S_r

[4]Note that the following equation is not derivable from the previous one.

$$R^{sT}a^r = -a^s + \dot{\omega}^s \times l^s + \omega^s \times (\omega^s \times l^s)$$

Finally, the equations for the angular velocities and accelerations are hypothesized as:

For all $s \varepsilon S_r$

$$\omega^s = R^{sT}(\omega^r + z_r \dot{q}_s)$$

$$\dot{\omega}^s = R^{sT}(\dot{\omega}^r + z_r \ddot{q}_s + \omega^r \times z_r \dot{q}_s)$$

where z_r is joint r axis of rotation between link r and link s. q_s is the angle between link s and its parent (link r) measured about axis z_r in the right sense (q_s is also known as "joint coordinate" of link s). $\dot{q}_s$ is the rate of rotation change between link r and s (the velocity of link s with respect to link r), $\ddot{q}_s$ is the acceleration of link s with respect to its parent (link r).

This completes the set of inverse equations.

These equations could also be arranged in an "inbound- outbound" computation organization similar to the direct dynamics case. The inverse dynamics out-bound recursion is:

$$\omega^s = R^{sT}(\omega^r + z_r \dot{q}_s) \quad 4\text{-}10$$

$$\dot{\omega}^s = R^{sT}(\dot{\omega}^r + z_r \ddot{q}_s + \omega^r \times z_r \dot{q}_s) \quad 4\text{-}11$$

$$a^s = -R^{sT}a^r + \dot{\omega}^s \times l^s + \omega^s \times (\omega^s \times l^s) \quad 4\text{-}12$$

$$r^s = \omega^s \times (\omega^s \times c^s) + \dot{\omega}^s \times c^s + a^s \quad 4\text{-}13$$

$$F^r = -m_r r^r + R_I{}^{rT}(f_E{}^r + m_r a_G) \quad 4\text{-}14$$

$$N^r = J^r \dot{\omega}^r + \omega^r \times (J^r \omega^r) \quad 4\text{-}15$$

The in-bound recursion is:

$$f^r = F^r + \sum_{s \varepsilon s_r} R^s f^s \quad 4\text{-}16$$

$$g^r = \sum_{s \varepsilon s_r} R^s g^s - N^r + \sum_{s \varepsilon s_r} l^s \times R^s f^s + R_I{}^{rT} g_E{}^r + c^r \times F^r \quad 4\text{-}17$$

$$\zeta^s = \begin{cases} z_r & g^s \\ z_r & f^s \end{cases} \quad 4\text{-}18$$

In summary,[5] the inverse dynamics equations are formulated by relating the accelerations (linear and

[5]The ζ^s in the previous equation is either the joint coordinate torque or force (depending on the type of the hinge).

angular) of each link to the acceleration of the base by abstracting away the intervening joint accelerations. This is done in equations 4-10 through 4-13.

Then one relates the distal forces and torques g^r, f^r acting on each link to the distal forces and torques acting on the tip(s) g^{n+1}, f^{n+1} by abstracting away the intervening joint forces and torques. This is done in equations 4-16 and 4-17. To solve these equations, one must solve to get the torques, g^r, given the motion (i.e. accelerations, velocities, positions, and orientations of the links through time- $\ddot{q}_s, \dot{q}_s, q_s$). At each time step, the state of the motion and the desired links accelerations are set according to the control parameters, then one solves an outbound recursion from the base to the tips. This recursion (equations 4-10 through 4-15) may be regarded as mapping a vector of $S^s = [\omega^s, \dot{\omega}^s, r^s, a^s]$ presenting the base $S^0=[\omega^0, \dot{\omega}^0, r^0, a^0]$ through the S links (equivalently, seen from the s-th link), together with the s-th input $q_s, \dot{q}_s, \ddot{q}_s$, into a vector $S^r = [\omega^r, \dot{\omega}^r, r^r, a^r]$ representing the base seen through the r-th input. Then one solves an inbound recursion from the tips to the base. This recursion (equations 4-16 and 4-17) solves for the forces and torques g^r, f^r acting on each link.

The recursive structure of these open chain inverse dynamics calculations when applied to the reduced system of our four-legged articulated robot of Figure 4.2 is shown in the inbound-outbound directed trees in Figure 4.6 (both the inbound and the outbound calculations are separated). Acceleration is propagated outbound (outwards from base to tips), incorporating at each stage the next joint acceleration. Inter-link forces and torques are thereafter propagated inbound (inwards from the tips to the base).

Tree nodes in the figure represent the total processing associated with each link in the inbound and the outbound recursion, and directed arcs represent data dependencies.

Recursion is implemented by connecting each node's output to the next node's input. While on either the inbound or the outbound recursion, the linear computation time coefficient is determined by the longest time required to propagate any single variable across the nodes. In determining this time, only ω^r, $\dot{\omega}^r, r^r, f^r$ $a^r$and g^r need be considered, as the other variables are not propagated up or down the recursion chain.

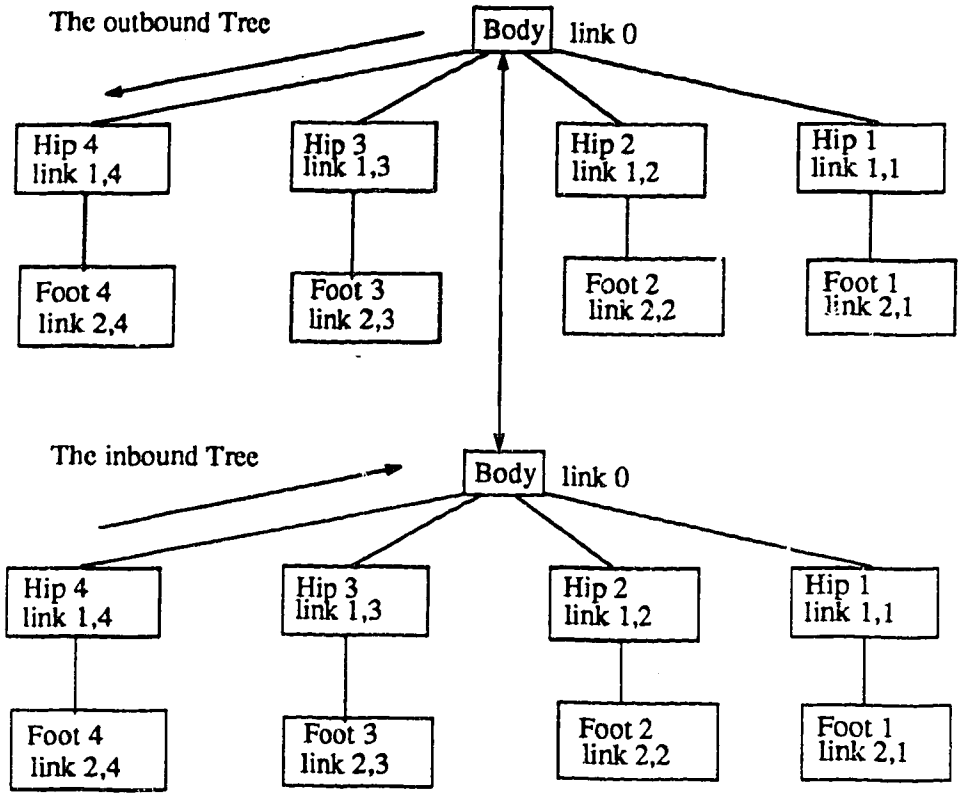

Figure 4.6 The Outbound-Inbound Directed Trees for the Inverse Dynamics Calculations

The produced set of inverse dynamics equations is similar to the equations of Luh, Walker and Paul [LWP76] which are basically a recursive form of the Newton-Euler equations of motion. The closed form dynamics equations may be written as:

$$g = H(q)\ddot{q} + e(q,\dot{q})\dot{q} + G(q) + C(q)^T w$$

which represent a set of n nonlinear differential equations that describe the motion of an n DOFs system in the inertial frame coordinates. Here $q = [q_1 q_2 .... q_n]^T$ is the joint coordinate angles between the links, and $\dot{q}, \ddot{q}$ are corresponding velocities and accelerations. g εR^n is the driving torque vector for the n DOFs. H(q) is the variable inertia matrix in the inertia frame. e(q,$\dot{q}$) is n × n matrix specifying centrifugal and Coriolis effects. C(q) is 6 × n Jacobian matrix specifying the torques (forces) created at each joint due to

external forces and moments exerted on link n (the tip), $C(q)^T$ indicates the transpose. w is 6×1 vector of external moments and forces exerted on link n (the tip). The first three components are g^{n+1} and the last three components are f^{n+1}. G(q) εR^n is the gravitational force vector. This closed form was shown by Hooker and Margulies [HoM65]to need $O(n^4)$ computational time (see for example the Gibbs-Appel method) compared to O(n) for the recursive form.

Now, in order to provide a means for modeling locomotion for the robot of Figure 4.1, an extension of these inverse dynamics equations to include closed kinematic chains for the legs that are in contact with the ground is needed. Consider the case when more than one leg of the four-legged robot is in contact with the ground. This is necessary in order to satisfy the conditions of static stability during locomotion. These legs are creating closed kinematic chains and consequently introduce, in addition to the intralimb coordination, a new interlimb coordination control problem. The robot must apply forces and torques, which will cause relative motion between the links of the body that consequently will produce reaction forces and torques from the ground (w_k for k=1,2,3,4 in Figure 4.1). The values of these ground reaction forces and torques that are needed to produce a desired body's trajectory are to be computed. But, as we will see in the next section, the force and torque assignment equations relating ground reaction forces and torques to the desired body and leg motion trajectories result in fewer equations than unknown variables. This underspecified nature of the problem has been solved by formulating the problem in terms of linear programming.

4.3. FORCE AND TORQUE ASSIGNMENT IN THE FOUR-LEGGED ROBOT

In order to obtain the equations of motion for our robot (a closed kinematic chain system), we first cut the hinges for the legs that are in contact with the ground in such a way that a system with a tree structure (like the one in Figure 4.2) is produced (this is called the reduced system). This reduced system is divided by the body into four dynamically independent subsystems, each of which is coupled with the body by a single hinge. First this reduced system is solved; then constraints which have been eliminated in the process of generating the reduced system are reintroduced. In this way the original system with closed

kinematic chains will be recovered [Wit77].[6] Each subsystem (k) in Figure 4.2, which is now (after cutting the hinges)a simple open kinematic chain containing 3 links, is governed by the Newton/Euler inverse dynamics equations of motion derived in the previous section, that have the following closed form format:

$$\tau_k = H_k(q)\ddot{q} + e_k(q,\dot{q})\dot{q} + G_k(q) + C_k(q)\, w_k$$

where $H_k(q)$ is the inertia matrix, $e_k(q,\dot{q})$ is the Centrifugal and Coriolis matrix, $G_k(q)$ is the gravity matrix, $C_k(q)$ is the inverse Jacobian matrix (this is transposed), τ_k is the vector of all joint torques for the kth subsystem, and w_k is the vector of external forces and torques.

This equation can be rewritten in the form:

$$\tau_k = C_k\, w_k + D_k \qquad 4\text{-}19$$

Therefore the equation of the reduced system is:

$$\begin{bmatrix} C_1 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 \\ 0 & 0 & C_3 & 0 \\ 0 & 0 & 0 & C_4 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} + \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{bmatrix} \qquad 4\text{-}20$$

Equation 4-20 was possible since the reduced system is divided by the body into four dynamically independent subsystems. The main purpose of dividing the reduced system into these dynamically independent subsystem is to be able to consider the formation and breaking of the kinematic chains during the locomotion. This division would allow us to deal with each leg's state (swinging or supporting) individually.

Wittenburg [Wit87] describes two different methods to deal with systems with closed kinematic chains. In the first method, in each closed kinematic chain one joint is removed. This produces tree-structure system for which the equation:

$$\delta\dot{q}^T(H\ddot{q} - B) = 0$$

can be formulated. The elements of $\delta\dot{q}$ are no longer independent when we re-introduce the constraints in the removed joints. To explain that, in the system of Figure 4.1 constraint forces as well as spring forces may have been removed in producing the reduced system of Figure 4.2. As a result of the re-introduction of

[6]Cutting hinges increases the total number of DOFs of the system. See later explanation.

kinematic constraints, the variations of the generalized coordinates of the reduced system are no longer independent. Constraints that account for eventually internal forces and torques in the cut hinges, as well as that are determined by the external forces and torques should be included.

In his second method to deal with closed kinematic chain systems, Wittenburg produced a tree-structure system by increasing the number of links. In every closed chain one link is replaced by two identical "half links" which are allowed to move without mutual constraints. Each half link has half the mass density and the same physical dimensions as the original link. Each joint of the original link is located on only one half link. All other constraints among the half links are removed.

The robot's body moves as a result of the supporting legs' forces. Therefore, one can claim that for any particular kinematic state for the robot, the relationship between the body's net forces and the legs' ground reaction forces and torques may be written as (see Figure 4.1):[7]

$$w_0 = \sum_{k=1}^{m} (A_k w_k + B_k) \qquad 4\text{-}21$$

w_0 results from the net action of the legs on the body. The term $(A_k w_k + B_k)$ represent the forces and torques applied to the body (link 0) by leg k. It has two components. The first is due to the joint torques and forces exerted by the leg on the ground, and the second is due to inertia and gravitational acceleration. Here A_k is a 6×6 matrix and is a function of the present position and orientation of the kth leg. B_k is a 6×1 matrix and accounts for the inertial and gravitational acceleration forces of the members of the kth leg. The quantity m is the number of legs that are in contact with the ground.

Since the required body trajectory is given (the outputs of the JCC algorithms of Chapter 3), the net forces acting on the body may be determined from the knowledge of the motion through Newton's law:

$$F^o = \frac{d}{dt}(m_0 v^0) = m_0(a^0 - g^E)$$

$$F_x^0 = m_0(a_x^0 - g_x^E)$$

$$F_y^0 = m_0(a_y^0 - g_y^E) \qquad 4\text{-}22$$

[7]The reason for the B_k terms in equation 4-21 is to have a similar form as that of equation 4-19.

$F_z^0 = m_0(a_z^0 - g_z^E)$

where m_0 is the mass of the body, g^E is the vector gravitational acceleration, and a^0 is the acceleration of the body with respect to the inertial frame. Similarly, the net moments on the body may be determined from Euler's equation:

$$G^0 = \frac{d}{dt}(J\omega^0) = J\dot{\omega}^0 + \omega^0 \times (J\omega^0)$$

$$G_x^0 = J_{xx0}\dot{\omega}_x^0 + (J_{zz0} - J_{yy0})\omega_y^0\omega_z^0$$

$$G_y^0 = J_{yy0}\dot{\omega}_y^0 + (J_{xx0} - J_{zz0})\omega_x^0\omega_z^0 \qquad 4\text{-}23$$

$$G_z^0 = J_{zz0}\dot{\omega}_z^0 + (J_{yy0} - J_{xx0})\omega_x^0\omega_y^0$$

where J_{xx0}, J_{yy0}, and J_{zz0} are the principal moments of inertia of the body, $\dot{\omega}^0$ is its angular acceleration, and ω^0 is its angular velocity with all components expressed in a coordinate system aligned with the principal axes of the body (remember we are using Bryant angles to describe the body's orientation- see equations 3-2 and Figure 3.2).

If the parameters of equation 4-21, were known (A_k,B_k), one could compute the reaction forces and torques that the legs exert on the ground (w_k). The inverse dynamic recursive algorithm for the open chain mechanism derived in the previous section may be used to determine the parameters of equation 4-21 A_k and B_k. The steps to do that are:

(1) The elements of w_k are first set equal to $[0\ 0\ 0\ 0\ 0\ 0]^T$, thereby indicating that the chain is open at one end and that no reaction forces or torques are being applied at its tip. This is done in order to calculate the amount that the inertial and gravitational acceleration forces from leg k contribute to the known motion of the body. The solution to this open chain inverse dynamics problem results in a determination of f_k^1 and g_k^1, the net forces and torques applied by leg k to the body at the joint between the two. These may be transferred to the center of gravity of the body to give

$$F_{0,k} = f_k^1 \qquad 4\text{-}24$$

$$G_{0,k} = g_k^1 - r_k^1 \times f_k^1$$

where r_k^1 is a position vector from the joint to the center of gravity of the body (the components of this vector are $(a_k,b_k,c_k)^T$ as described in the kinematic model of the robot and the automatic leg

positioning algorithm of Chapter 3). $F_{0,k}$ and $G_{0,k}$ are the net forces and torques acting on the body due to leg k. Since $w_k = [0\,0\,0\,0\,0\,0]^T$, from equation 4-21 and 4-24, then B_k may be determined as

$$B_k = [F^x_{0,k}\ F^y_{0,k}\ F^z_{0,k}\ G^x_{0,k}\ G^y_{0,k}\ G^z_{0,k}]^T$$

(2) The first column of A_k may be determined by setting $w_k = [1\,0\,0\,0\,0\,0]^T$ and solving for the kth leg, which results in a determination of f_k^1 and g_k^1, the net forces and moments applied by the leg to the body at the joint between the two. These may be transferred to the center of gravity of the body as before. Once new values for $F_{0,k}$ and $G_{0,k}$ are obtained, the first column of A_k, then, is just

$$A_k^1 = [F^x_{0,k}\ F^y_{0,k}\ F^z_{0,k}\ G^x_{0,k}\ G^y_{0,k}\ G^z_{0,k}]^T - [B_k]$$

(3) The second column of A_k follows by assigning $[0\,1\,0\,0\,0\,0]^T$ to w_k, and the procedure is continued until all six columns have been determined. All legs are considered simultaneously as suggested by the independence shown in equation 4-20, resulting in all of the values for each of matrices A_k and B_k for the legs that are in contact with the ground.

It should be noted that all the open chain inverse dynamics problems are solved for the particular kinematic motion of the system under which equation 4-21 was originally formed.

Now, in equation 4-21, w_0 is known from equations 4-22 and 4-23 and A_k and B_k are known from steps 1-3. Thus, equation 4-21 represents 6 scalar equalities in 24 terminal links ground reaction forces and torques unknown. Actually, the case is not exactly so, since during locomotion two or three legs of the robot are forming a closed kinematic chain, and these are the ones that must be considered for calculating the ground reaction variables. Trying to estimate the values of these unknowns given the 6 equations shows the underspecified nature of the problem. Many solutions are possible; therefore, some method should be introduced to choose among these solutions.

This problem was raised in [OkP73], [OGA74] [PaF73] and [OrM73]. The principles that were suggested as a solution were based on imposing a new set of constraints such as:

(a) The maximal value of the normal reaction forces in the supporting points (the legs that are in contact with the ground) should be kept at a minimum all the time.

(b) The maximal angle between the direction of the reaction force and the axis of the friction cone (which is used as an envelope for the leg positions) should be kept at a minimum.

These constraints were realized by solving two problems of linear programming. The first constraint provided some evenness in the support force distribution, while the second one provided an increase in the friction margin.

In the following, the problem is formalized in a form of one linear program to compute the joints' torques and forces to optimize the combination of power exertion on the ground and load balancing. The set of constraint equations and the objective function of the linear programming problem are derived in steps (a) to (c).

(a) Since the joint torques and forces will be used in the objective function (see later) to compute the power exertion, one should constrain them. The joints' torques are related to the terminal reaction forces and torques through equation 4-19:

$$\tau_k = C_k w_k + D_k \qquad 4\text{-}25$$

for k=1,2,3,4 where

$$\tau_k = [\tau_{1,k}\ \tau_{2,k}\ \tau_{3,k}]^T$$

$\tau_{i,k}$ is the ith joint actuator torque in the kth leg, while τ_k is the vector of all joint actuator torques for the kth leg. C_k is a matrix of dimension 3×6, and D_k is a vector of dimension 3×1. Again, the inverse dynamic procedure for the open chain mechanism may be used to determine C_k and D_k as before. Actually, the parameters of equation 4-25 can be calculated during the calculations of the parameters of equation 4-21, as will be described in the computation model in the next section.

The set of constraint equations on the joints' torques are the physical limitations on the joint actuator torques:

$$C_k w_k + D_k \le \tau_k(max)$$

for k=1,2,3,4 where

$$\tau_k(max) = [\tau_{1,k}(max)\ \ \tau_{2,k}(max)\ \ \tau_{3,k}(max)]^T$$

(b) Next, one can constrain the reaction forces on the tip of each leg by imposing a maximum normal component on the leg's tip reaction force onto the supporting surface so as to balance the load among the supporting legs, as was done in [McO76]. This may be used to prevent penetration of the leg into soft ground.

$$f_k^N \le f_{max}^N \qquad k=1,2,3,4$$

where $f_k{}^N$ is the normal component of the reaction force of the kth leg onto the ground, and it may be calculated as

$$f_k^N = [0 \;\; 0 \;\; 1]\, N_k\, [f_k^{n+1,x} \;\; f_k^{n+1,y} \;\; f_k^{n+1,z}]^T$$

for k=1,2,3,4 where N_k is the orientation matrix of the local surface that the supporting legs form with the ground (remember the computations of this plane in Section 3.4.2). f_{max}^N is the maximum normal component of the terminal reaction forces of the legs onto the ground.

(c) Concerning the objective function, one optimization criterion which has been used extensively in the literature is that of energy minimization [McO76], [Vuk73]. The suggested objective function minimizes the maximum normal terminal reaction forces and the legs exerted power.

$$\phi = \sum_{k=1}^{m} [\xi H_k \omega_k{}' \tau_k + (1-\xi) f_k{}^N]$$

where H_k are 1×3 matrices and equal to [1 1 1]. $\omega_k{}'$ is a diagonal matrix of the angular velocities of the three angles for leg k. ξ is a weighting coefficient.

This completes the formulation of the linear programming problem.

It is important to note that this linear programming problem will be solved only for the legs that are in contact with the ground (i.e., we never have 24 variables at the same time except when the robot is standing still). The forces and torques for a non-contacting leg may be determined directly using the open chain inverse dynamics calculations, and they need not be included in the linear programming analysis (see next section).[8]

[8]It is assumed that the swinging legs impact negligible torques and forces to the robot body. This is in conformity with the more liberal approach that have been taken in [IsC87] and [GiM85], where legs were operated in two different "modes", a kinematic mode and a dynamic mode. Under their arrangement, any leg pushing on the ground was operated in dynamic mode, whereas the legs in swing phase were operated in kinematic mode.

At each time step, we set up the linear programming problem and use a linear programming algorithm to solve the problem and calculate the resulting joint torques for the legs in the support phase. As will be described in Chapter 6, Charnes' M method for solving the linear program (a variation of the simplex method) [DrS66] is used in the experimentation system.

4.4. THE NC STRUCTURE

As described in Chapter 3, the outputs of the JCC are fed as inputs to the NC. Due to the varying patterns of gaits and leg cycles, a number of rather complicated sequences of closed and open chains are formed many times in the course of the robot navigation. This imposes the requirement to have a general enough motion control system that takes care of the formation and breaking of the chains during the motion.

Figure 4.7 shows the proposed[9] numerical controller (NC) process organization that calculates the forces and torques needed for given body and leg trajectories. Each time step, the NC deals with the legs that are in contact with the ground by using the linear programming formulation derived in the previous section (Box A and D). The legs that are not in contact with the ground are forming open kinematic chains and therefore are solved using the open chain inverse dynamics equations derived in the section 4.2.2 (Box B). The figure also shows the direct dynamics module (Box C) that is used to produce the motion as a result of the driving forces and torques in the animation system which are based on Armstrong's direct dynamic equations presented in section 4.2.2. Note that the direct dynamic module is not a part of the NC- see figure 3.1- it is a part of the animation subsystem (ANS) and its implementation details are described in Chapter 6.

In the next section a distributed computation model that reduces the computation time needed by the NC will be proposed. The proposed computation model has a very important feature, that of a high

[9]In Figure 4.7, $\dot{X}_d, \dot{Y}_d, \dot{Z}_d, \Psi_d$ are the desired velocities and rotation angle. see Section 3.4.2 for the control of the robot motion.

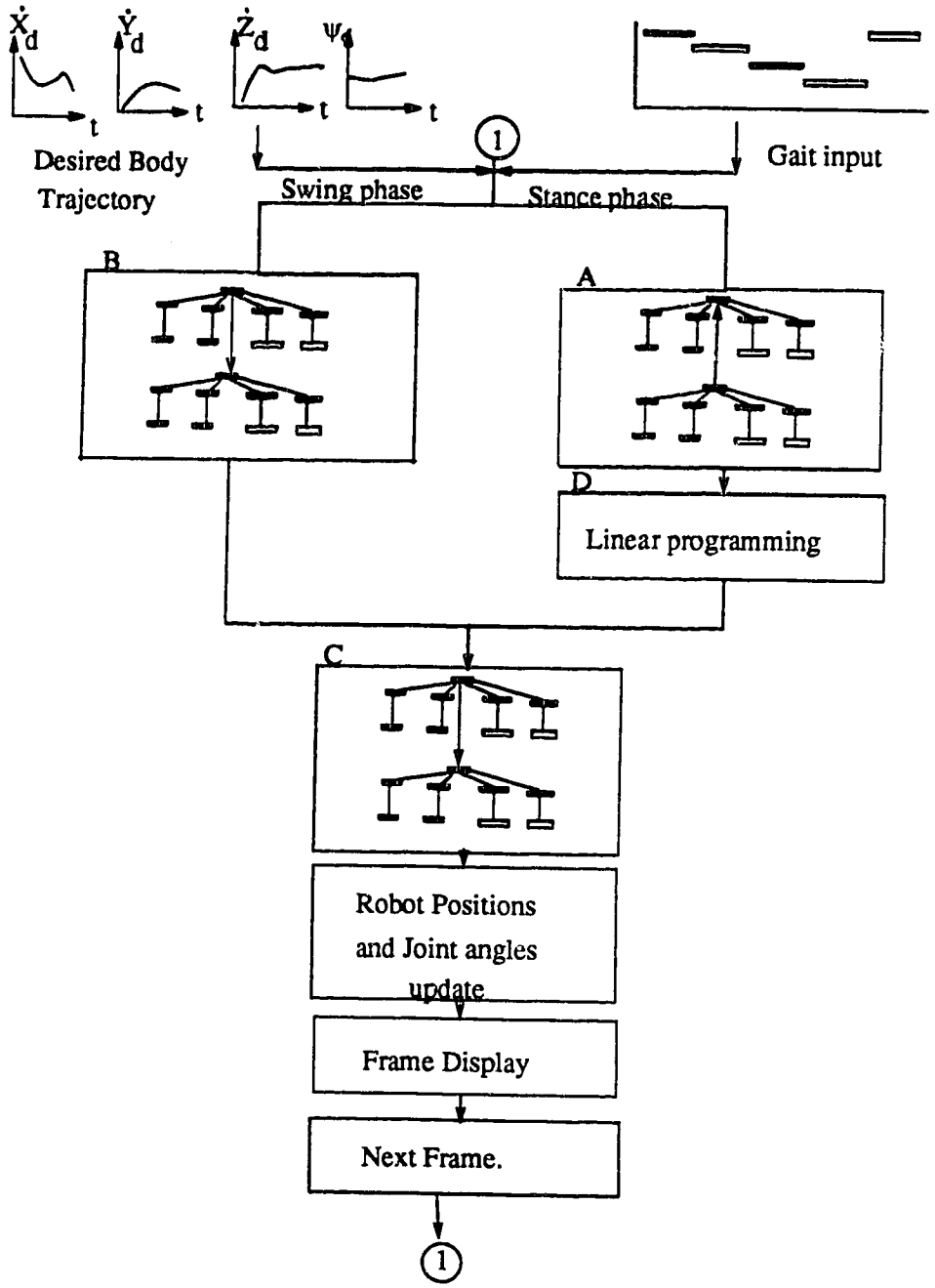

Figure 4.7 The Numerical Controller Processes Organization

throughput computation rate.

4.5. A PIPELINED COMPUTATION MODEL FOR THE NC

For real-time simulation of the previous motion control system, the calculation rate should be at least 10 to 15 times the link structure resonant frequency (when integrating the equations of motion, we need small time step for stability of the solution). This frequency is a function of the mechanical parameters and the masses of the links. This rate of calculation is difficult to achieve due to the large number of vector and matrix operations associated with the dynamic motion control system, and the formation and solution of a linear programming problem each time step.

However, two main approaches have been proposed in the literature to reduce dynamics computation time:

(1) Simplify the dynamics model calculations: (a) Armstrong [AMO87] presented a slowband-fastband division for the computations. Slowband computations are not done in every computation cycle (they compute slowly varying variables). (b) Raibert [Rai77] suggested precalculating some of the dynamics terms in advance and using a look-up table. (c) Bejczy, among others [BeP81], chose to reduce the computation load by neglecting some terms which are less significant compared with others (e.g. the Centrifugal, Coriolis, and coupling terms). (d) Binder [BiH86] achieved more concurrency in computations by substituting "predicted" values for the actual values of variables involved in the recursive equations to increase the pipelining in his model.

(2) Improve the computation structure: (a) Armstrong suggested a tree-like processes structure for the parallel execution of his direct dynamics equations [AGL86]. (b) Binder proposed a two-way ring distributed computation architecture that utilizes "predictions" to achieve concurrency in processing his inverse dynamics calculations [BiH86]. (c) Lathrop [Lat85] proposed a special purpose VLSI robot chip capable of handling general vector arithmetic. The parallel processing was achieved by pipelining complete sets of joint torques at successive time intervals. (d) Lee proposed another similar VLSI implementation in [LeC86].

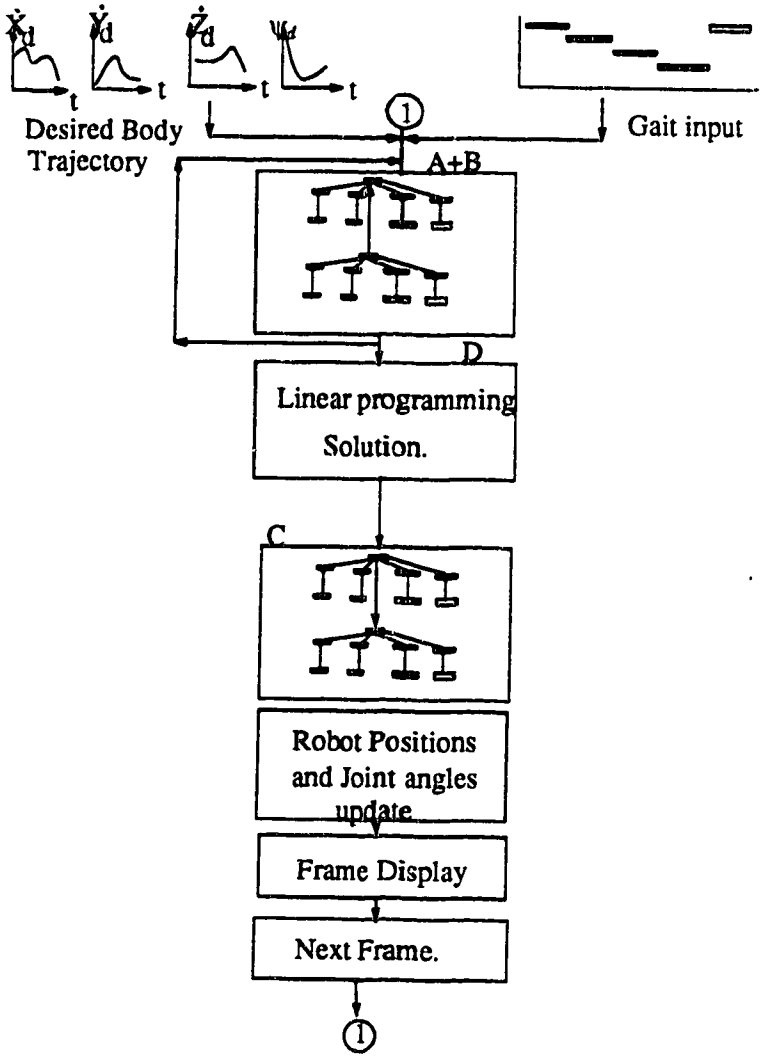

Figure 4.8 Enhancement to the Computation Organization

In the proposed computation model for the NC, a tree processes organization was chosen to be used since it is directly compatible with the inbound-outbound inverse and direct dynamics computations described in section 4.2.2. Figure 4.7 shows that there is one process for each tree node in either the direct or the inverse computations. The recursive structure of the open chain inverse dynamics calculations of Figure 4.6 is shown in parts A and B in Figure 4.7, whereas the recursive structure of the direct dynamic calculations of Figure 4.4 is shown in part C of Figure 4.7.

The calculations of the coefficient matrices of equations 4-21 and 4-25 are done in box A in Figure 4.7. The legs that are in contact with the ground are dealt with by forming and solving the linear programming problem (box D). At the same time, the legs that are not in contact with the ground are forming open kinematic chains and therefore are solved using the open kinematic chain recursive procedure of Figure 4.7 (box B).

Notice that in box A, propagation and calculations are only done for the legs that are in contact with the ground (the legs that are in the stance phase). Whereas in box B, propagation and calculations are only done for the legs that are not in contact with the ground (the legs that are in the swing phase).

Figure 4.8 shows more enhancement to the computation organization of Figure 4.7. The two boxes A and B are merged into one box (A+B). The reason is that the first step in calculating the coefficients of equations 4-21 and 4-25 was actually solving an open kinematic chain for the legs that are in contact with the ground. This is exactly the case for the legs which are in the swing phase. They have no reaction forces or torques applied to their tips. So one can view the situation as: during the inbound-outbound calculations for the case of w_k=0 for all legs, some of the legs are in the swing phase. Some are in the stance phase. Two vectors are calculated for each leg in the stance phase B_k and D_k, whereas the joint torques of the legs in the swing phase are calculated in this step.

The reason for the loop in the A+B box in Figure 4.8 is the pipelining of the w_k of $[1\,0\,0\,0\,0\,0]^T$ through $[0\,0\,0\,0\,0\,1]^T$ for calculating the other coefficient matrices A_k and C_k for legs k that are in stance phase.

According to Figure 4.8, all but one level within each computational tree in the NC is active at a time. All other processes in other levels wait for the values of the recursive variables before they can begin computations. Hence, only one tree level at a time is engaged in useful computation.

In order to increase performance, it is necessary that two or more NC level and/or tree level processes be active simultaneously. There are two ways to achieve this:

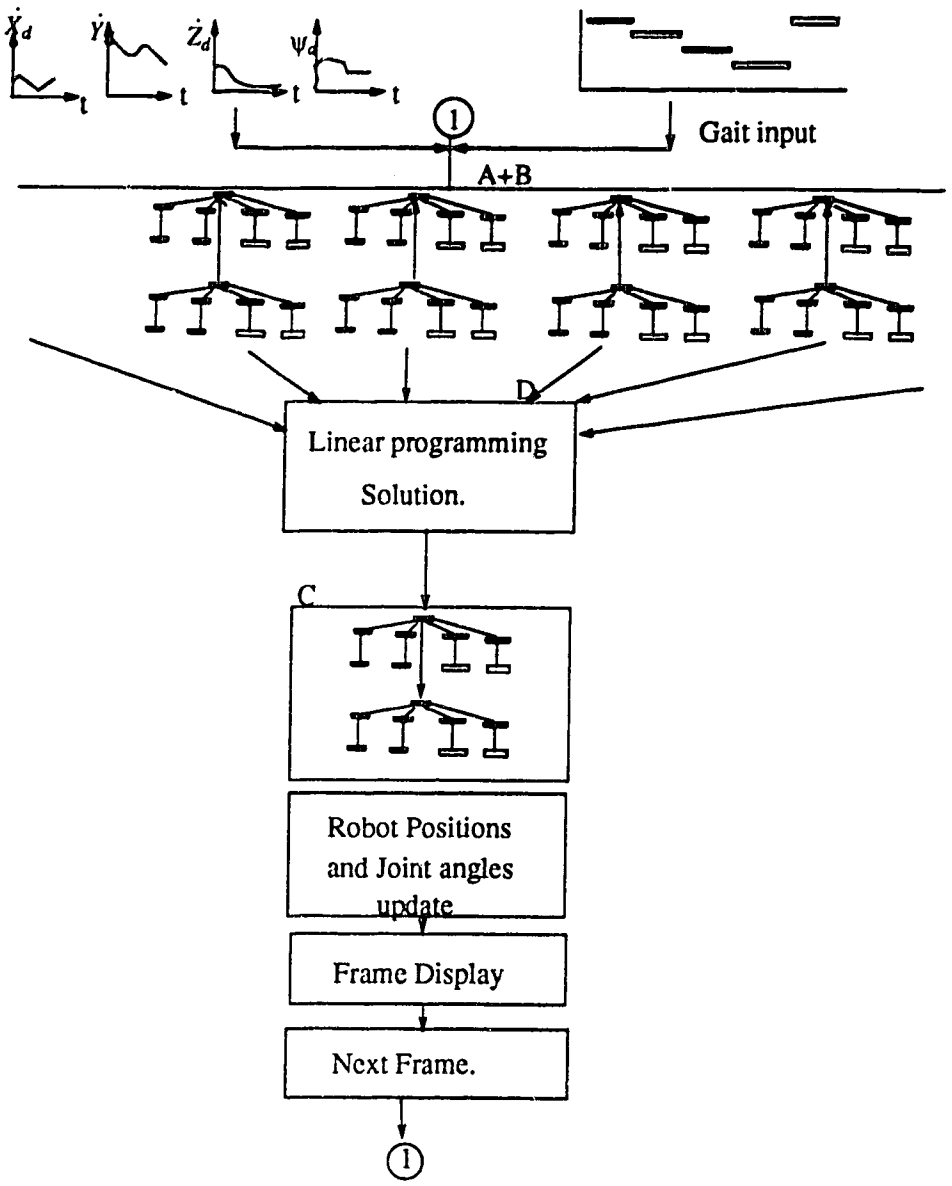

Figure 4.9 Further Enhancement to the Computation Organization

(1) One or more tree levels within each computation tree can receive the values of the recursion variables prior to the time they are provided by their adjacent processes. This could be achieved by "predictors" that provide "acceptable" values of the recursion variables (or some of them) in advance of the time they would normally be provided in the direct application of the pipelining. This is

similar to the slowband variables (subset of the recursion variables) of Armstrong [ArG85] and the "predictors" of Binder [BiH86]. The difference between Armstrong's approach and Binder's approach is that the variables of the slowband were selected to be variables that are known to be slowly varying, so that even if they were not updated each time cycle, the accuracy of the computations was preserved; whereas in the case of Binder [BiH86], predicting the set of all recursive variables introduced some errors in the dynamics computations. Using all or partial predictions, instead of waiting for tree level r-1 to propagate the complete set of recursive computations necessary to start its computations, tree level r can start calculations based on predicted values. One way to estimate predicted values is based on calculated terms in previous sample intervals of a similar locomotive skill. Another is to estimate them based on the previous tree levels in the tree (i.e., the pipelining may not only feed the next level of the tree, but it might go two or three levels ahead).

(2) As indicated in [AMO87] the synchronization overhead, rather than the communication overhead, is the principle cause of any lack in performance in the parallel implementation of the dynamics equations. To reduce synchronization overhead complete sets of desired positions, rates, accelerations and the body's desired motion may be pipelined through the levels of the NC and/or the levels of each individual computation tree in a dataflow manner.

In Figure 4.9 the loop of Figure 4.8 is unfolded to allow parallel determination of the coefficient matrices. Also, Figure 4.9 shows how the successive sets of desired joints' position rates, accelerations, and body's trajectories may be pipelined through the levels of the NC to perform in a "near-real" time locomotion computation. The only problem with that is, since the integration step (in the direct dynamics calculations- Box C) updates the orientation matrices R_I^r and R^s, all torque vectors produced at successive time slots which have entered the pipeline use different orientation matrices in the calculations of their coefficients. But another argument might be that, as the new updated orientation matrices becomes available and are used, they might lead to better approximation.

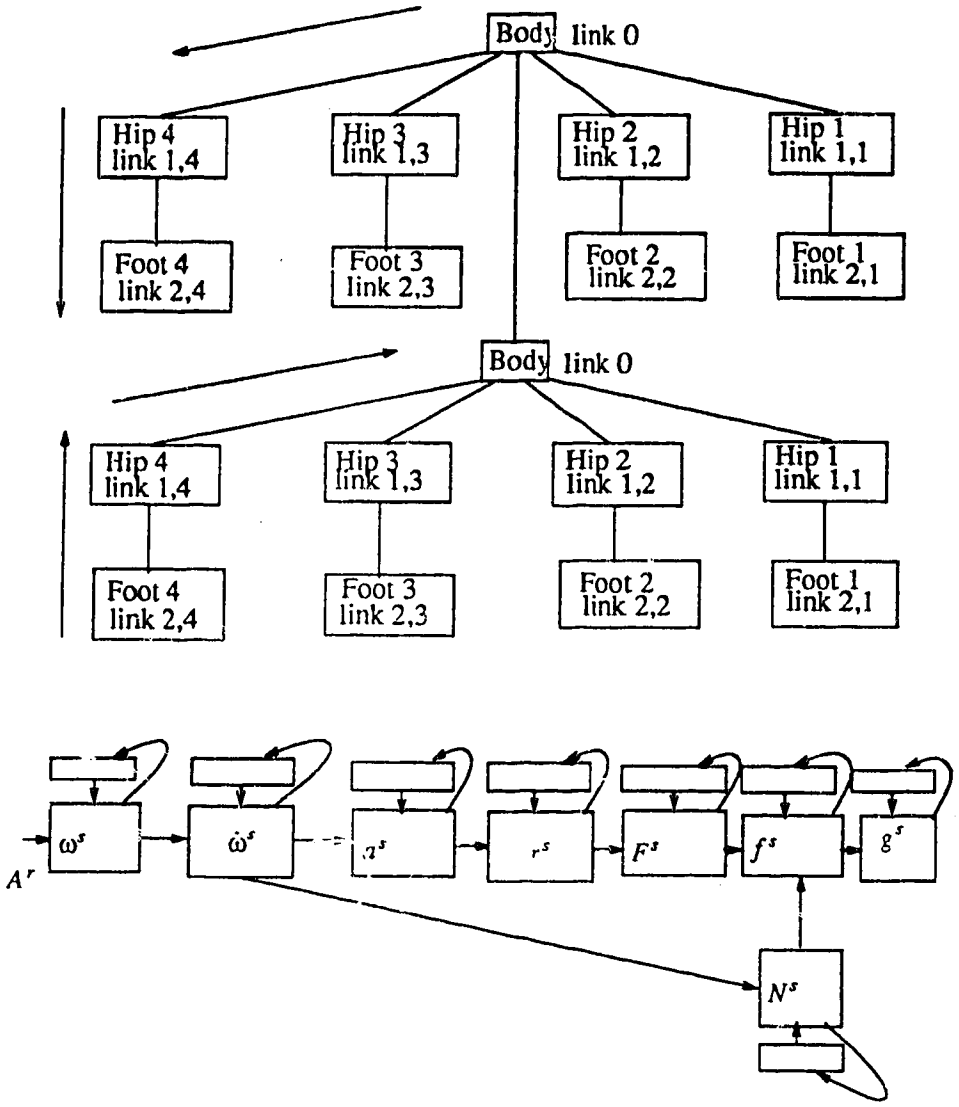

Figure 4.10 Computation Organization for the Inverse Dynamics

Another computational saving might be gained if one restricts the body's trajectory to straight lines. Under such condition, the coefficient matrices A_k, B_k, C_K, and D_k can be considered fixed and could be calculated only once for each particular locomotion skill.

Finally, Figure 4.10 shows how each full tree, at the top of Figure 4.9, is assigned to a processor group in order to allow more efficient pipelining of the successive torque profiles. Also the direct dynamics

tree (Box C) could be assigned to another processor group shown in Figure 4.11. This is similar to Lathrop's suggestion [Lat85] to have only one processor group for all the links and to implement recursion by connecting the output back to the input through a buffer. The only restriction here is that the computation of one set of joint torques must be completed before the next can begin. Otherwise, with one processor group for each joint (link) it is possible to pipeline successive sets of joints torques as described before.

Note that the NC computation are not required to be highly accurate since all that is needed from the NC is a reasonable first approximation to the desired motion. We have deliberately chosen not to include the friction of the ground, the viscosity at the joints, and also used several approximations in computing the dynamic variables in both the inverse and direct dynamics computations. Also added to the inaccuracy of the NC, the slow and fast decompositions of the calculations, the varying of the integration step size, the assumption that the swinging legs don't have impact on the body motion, and finally the assumption of ball-and-socket joints at all the robot hinges.

It is important to distinguish here, again, between animation and the kind of animated simulations that the experimental work of this dissertation describes. What we are dealing with here is what is called "task level simulation", where the problem is to control the robot by specifying a set of events and some constraints on its behavior, and letting the robot fill in the motion details as necessary. So, we are not interested in constraining the simulated robot to do what we want it to do. Rather, we want to let it interact with the environment and adapt its goal-directed motion accordingly. This is the reason for not using any simpler dynamic model in the NC.

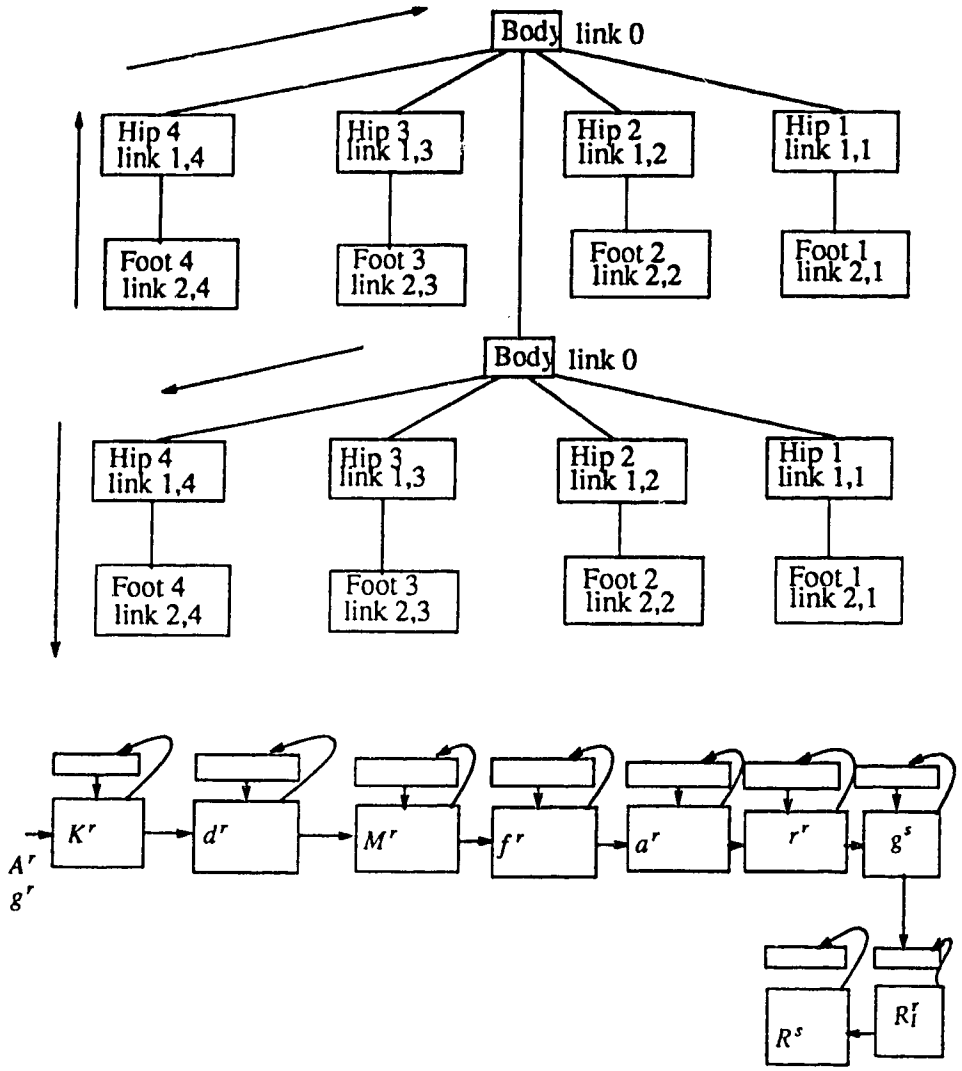

Figure 4.11 Computation Organization for the Direct Dynamics Calculations

CHAPTER 5

THE LEARNING CONTROLLER (LC)

5.1. INTRODUCTION

It should be apparent from the previous chapters that the mathematical principles governing the locomotion of the four-legged articulated robot are both complex and computationally time consuming. It is even more difficult to establish some intuitive link between the parameters of the computations and the resulting motion [IsC87]. For example, it is clear that if the robot found itself suddenly about to hit a small obstacle, it might have to lift its legs and place them on top of the obstacle and adjust its body's orientation to maintain its balance. It is very difficult to analyze such motion through dynamics (i.e. as a series of forces and torques on the joints of the robot) during actual motion execution. It is at this level, however, that the robot would have to maintain its sustained stable locomotion. What needs to be looked at, therefore, is how to replace parts of these heavy non-intuitive non-real time computations with more efficient intuitive real time symbolic computations. In other words, the objective is to develop a skill-based AI controller that is to perform in the same way as the NC with easier and faster abilities to execute its skills , i.e. would be able to integrate sensory data, robot capabilities, and task constraints and operate at speeds comparable to the real-time locomotion task (this is what we mean by "acquiring high agility").

This chapter describes a learning controller (LC) that gradually develops such an intelligent body of locomotive knowledge through practice and experience (i.e. develops motor capabilities). The LC starts with general inference rules and learning techniques and gradually acquires complex skills through interaction with a human teacher. The human supervises the LC when it takes charge of the control of the robot actions. Towards this end, this chapter first analyzes the issues involved in establishing a new model for motion skill acquisition. The model proposed is composed of two major functional blocks: skill transfer from the human expert to the robot and skill discovery by the robot itself. Skill transfer involves the

selection of a set of features as basic vocabularies of the robot's internal language, to provide the means for describing the situations and actions of the robot. It deals with how the robot interprets and operationalizes the skill descriptions provided by the human expert. Skill discovery is the process of the robot's self-formulation of rules connecting situations to actions, based on the experience accumulated. This concerns how the robot can generate its own skills by refining and exploring those skills which have not been presented by the expert teacher. Then the proposed model is applied to the acquisition of locomotion skills by the LC. Finally, the algorithms of the LC are developed.

The organization of this chapter is as follows: Section 5.2 describes some of the skill acquisition models that can be based on existing AI paradigms. In Section 5.3 the new model for motion skill acquisition is described. In Section 5.4, the details of skill transfer and of skill discovery are exploited by applying the model of section 5.3 to the acquisition of locomotion skills for the four-legged articulated robot. Also the LC algorithms are developed in this section. Finally, Section 5.5 reflects on the ideas presented in this chapter.

5.2. SKILL ACQUISITION MODELS THAT ARE BASED ON EXISTING AI PARADIGMS

The term "skill", as defined in behavioral psychology, refers to sensory driven functional motions controlled at the execution level by automatic actions acquired by learning and training [SZH79]. Skill acquisition can be modeled based on some of the existing AI paradigms. One such model may be based on the pattern-directed inference paradigm which generates outputs based on input patterns obtained from the external and internal sensory data [WaH78]. This is often employed in cognitive science to model the human acquisition of a skill [RuN82]. The mapping between input patterns and output actions can be structured hierarchically or distributively with the multilevel abstraction of inputs and outputs as done in goal-directed hierarchical or schema-based problem solving [Alb79] [Arb85]. Furthermore, the inference rules (those relating situations, i.e. input patterns, to actions) can be modified or generated based on the explorations of experience, as is done in genetic algorithms for evolutionary rule learning [HoR78].

A second model may be based on the mathematical and linguistic representation of an adaptation mechanism as is done in "self-organizing" control[1] [Fu70] [PrM79] , and learning stochastic or fuzzy automata [NaT74] [NWM77]. For example, the concept of "baby-robot," devised to investigate the process of early cognitive development in robots, emphasized self-learning with minimal initial knowledge as the structure of the baby robot. This was applied to robot path-finding cognitive development [Mey85]. Another example is the self-learning stochastic automaton applied to the task of high precision insertion using a force/torque sensor that emphasized gradual optimization of quantization levels of input and output variables [SBS82].

A third model may be based on the interactive and automatic refinement and expansion of the knowledge base of an expert system. Many expert systems represent their expertise as large collections of rules that need to be acquired, organized, and extended. Such systems allow the interactive updating of the knowledge base and a term for them has been coined by Waterman as: "adaptive production systems" [Wat75].

For instance, rule-based locomotion control has been applied with considerable success in rehabilitation engineering [Tom81]. Due to its repetitive nature, locomotion is considered to be an ideal application for control by skill based AI systems. As a matter of fact, gait analysis studies provided all that is needed is to encode locomotion in the form of production rules. For example, it is easy to establish that, in biped walking on level ground, the following sensors are exposed to discrete (0-1)- changes: heel contact, middle foot contact, toe contact, etc. A binary variable can be also associated with the presence or absence of flexion and extension terminal angles of leg joints. To each of these sensory patterns corresponds a configuration of discrete joint states (hip, knee, ankle). Consequently, formal representation of events taking place in biped locomotion on level ground can be expressed in rules of the following form:

$$B(x_1, x_2, \ldots, x_n) \rightarrow (J_1(k), J_2(k), \ldots, J_m(k)) \qquad 5\text{-}1$$

where B is a boolean function, x_i are binary sensory inputs, $J_i(k)$ is the discrete joint state of joint i; k=1,2,....,l (i.e. joint i can take l different discrete configurations). The rule base control relying on 5-1 has

[1]This is a term that is used in modern control theory.

been given the name "robot control by artificial reflexes" [ToB86]. This system was later extended to other biped gait modes (stair climbing, walking on ramp, adaptive step length, etc.) including rules reflecting human behavior in the case of abnormal motion [Tom81]. It should be apparent, however, that this system has been designed to capture the "kinematics" of biped locomotion skills (i.e. it deals only with joint angles with respect to time). Similar work that has dealt with joint angles and their rates can be found in [LGI86].

Due to the very nature of skill based AI systems (mainly relaying on symbolic manipulations), the capturing of knowledge expressed in terms of skills' dynamics has not been tackled before. A methodology of skill acquisition that is based on torque and force profiles is still lacking.

5.3. A DYNAMICS-BASED MODEL FOR MOTION SKILL PROGRAMMING

Here we propose a theoretical model for dynamics-based skill programming. The model is composed of two major functional blocks: skill transfer from the human expert to the robot and skill discovery by the robot itself. Skill transfer involves the selection of a set of features as a basic vocabulary of the robot's internal language to provide the description of the situations and actions of the robot. It deals with how the robot interprets and operationalizes the skill descriptions provided by the human expert. Skill discovery is the process of the robot self-formulation of rules connecting situations to actions, based on the robot experience accumulated. This concerns how the robot can generate its own skills by refining and exploring those skills unable to be presented by the expert skill descriptions.

What we want to do is understand how biological creatures organize their own routine, instinctive behaviors. In computer graphics, people generally draw on disciplines of optics and physics to understand the interaction of light with materials (this is an important area in computer graphics called "ray tracing"). For behavior modeling, we are drawing on the work of behavioral psychologists like R. Schmidt and J. Adams who spent long time thinking about how behavior is organized, how motor skills are stored in memory, how they are retrieved, and how they are produced differently as a result of practice and experience, although not quite in terms that lend themselves directly to implementing algorithms. The point is that we want to build hierarchical suites of motor skills, and interconnect them in such a way, that

the simulated robot can do certain kinds of problem-solving on its own.

The proposed model can be formulated by representing a situation and an action in the situation space (S) and the action space (A), respectively; where the situation and the action spaces are structured by their set of primitive features (axes). A situation or an action can be completely specified by assigning values to the individual features defining the situation or the action space.

Following the approaches taken by previous motor learning models [Fu70] [Alb79], we restrict the types of primitive movements to a limited set and discretize the situation and action space axes into a few intervals along each feature.

We also define a complete training set to be any set that contains at least one point from each of the defined regions (formed by the intersections of the intervals). This is in order to successfully identify the values of control torques and forces that are needed for any particular environmental conditions (test cases). These training set cases will be solved by the numerical controller, and then the learning controller will utilize extrapolation techniques to generalize them to any test case. The objective here is to perform the underlying activity (primitive movement) under different environmental conditions without solving each case dynamically.

Dynamically speaking such sensitivity analysis is sound. For example, consider the effect of changing the time duration of a movement on the values of joint torques and forces. Such effect has been investigated before [Hol84]. If the motion $(\theta_1(t),\theta_2(t),....,\theta_n(t))^T$ has been fashioned to be done in a time interval t_f, then if this same motion is to be performed in a time duration t_g, then the new motion $\tilde{\theta}(t)$ will be such that $\tilde{\theta}(t)= \theta(r)$, where r=r(t) is a monotonically increasing function of time with r(0)=0 and $r(t_g)=t_f$. The function r(t) can be considered a time warp which moves the linkage structure along the same path with a different time dependence. r(t) must increase monotonically because time can not reverse itself, and r(0)=0 because the movement must start at the same point.

To see the relation between the two torques, consider:

$$\frac{d\tilde{\theta}(t)}{dt}=\frac{d\theta(r)}{dr}\cdot\frac{dr}{dt}$$

Similarly

$$\ddot{\theta}(t)=\theta''(r)\dot{r}(t)^2+\theta'(r)\ddot{r}(t)$$

Where "dot" means the derivative is with respect to time, and "dash" is the derivative with respect to r.

From the dynamics of motion:

$$\tau(t)=J(\theta(t))\ddot{\theta}(t)+\dot{\theta}(t)C(\theta(t))\dot{\theta}(t)+g(\theta(t))$$

Separating the acceleration and velocity dependent torques:

$$\tau_a(t)=(\tau_{a1}(t)\tau_{a2}(t)....\tau_{an}(t))^T$$

$$\tau(t)=\tau_a(t)+g(\theta(t))$$

This is the torque required for the old motion, now for the new motion:

$$\tau_a(t)=J(\tilde{\theta}(t))\ddot{\tilde{\theta}}(t)+\dot{\tilde{\theta}}(t)C(\tilde{\theta}(t))\tilde{\theta}(t)$$

Substituting from the first and second equations,

$$\tau_a(t)=[J(\theta(r))\theta''(r)+\theta'(r)C(\theta(r))\theta'(r)]\dot{r}^2+J(\theta(r))\theta'(r)\ddot{r}$$

$$\tau_a(t)=\dot{r}^2\tau_a(t)+\ddot{r}J(\theta(r))\theta'(r)$$

This is a potentially significant reformulation of dynamics, indicating how the underlying dynamics change when the duration time changes. The new torque

$$\tau_a(t)=(\tau_{a1}(t)\tau_{a2}(t).....\tau_{an}(t))^T$$

is related to the old torque

$$\tau_a(t)=(\tau_{a1}(t)\tau_{a2}(t).....\tau_{an}(t))^T$$

by the scaling factor $\dot{r}^2$ plus a term proportional to the generalized momentum $J(\theta(r))\theta'(r)$.

With respect to other situation and action space features, Marshall [Marshall85] presented a model in which he varied the initial characteristics (position and velocity) of one segment and examined the changes produced in the movement between the original simulation and the simulation with altered segment parameters. He used this to examine the effects of increasing or decreasing stride length (in human locomotion) on the initial values for the recovery leg and consequently on the required torque histories.

Meno [MMS81] also studied the effects of changing the size, mass or mass distribution. He used simulation experiments to evaluate normal and pathological human gaits. The effect of adding an athletic or

protective device to a segment, or changes in the segmental inertia including amputation were also evaluated. Marshall [MJW85] suggested that by comparing the torques required to produce the new movement with those from the original motion, researchers may find information useful in planning training regimes or skill development.

All this suggests that a very large classes of locomotion can be implemented by predetermining approximations that can be made quite precise as the robot experience increases. The LC can structure these classes of locomotion into sets of potential responses to anticipated conditions. Accordingly, the LC will deal with the motion control problem as search in the action space for the control patterns that are necessary for achieving particular kind of locomotion under the current environment's conditions.

Let F_S^E and F_A^E represent the two feature sets defining the situation space, S^E, and the action space, A^E ,respectively, of the human expert skill descriptions. Similarly, let F_S^R and F_A^R represent the two feature sets defining the situation space, S^R, and the action space, A^R, of the robot. Then:

(1) A skill can be represented as

$$S_i^E \rightarrow A_i^E \qquad 5\text{-}2$$

represents the expert internal skill i. It could be represented as

$$F_{S_i}^E \rightarrow F_{A_i}^E \qquad 5\text{-}3$$

Whereas

$$S_i^R \rightarrow A_i^R \qquad 5\text{-}4$$

represents the robot's internal skill i. It could be represented as

$$F_{S_i}^R \rightarrow F_{A_i}^R \qquad 5\text{-}5$$

In other words, the expert and robot skills are represented by a set of production rules, called the "expert rules" and the "robot rules" respectively. The situation and action parts of these rules are described in terms of a set of features selected by the human/robot to define their situation and action spaces respectively.

(2) Skill transfer can be represented as the mapping of both the situation and action spaces between the human expert and the robot.

$$S_i^E => S_i^R \qquad 5\text{-}6$$

represent a situation mapping (notice => means mapping whereas → implies a rule). This is represented as the mapping

$$F_{s,i}^E => F_{s,j}^R$$

This situation mapping is obtained by dimension expansion (or shrinking) and/or resolution enhancement of both $F_{s,i}^E$ and $F_{s,j}^R$ to make suitable correspondence between the two spaces: S_i^E and S_i^R.

Similarly,

$$A_i^E => A_i^R \qquad 5\text{-}7$$

represents an action mapping. This could be represented as the mapping

$$F_{A,i}^E => F_{A,i}^R$$

This action mapping is obtained by dimension expansion (or shrinking) and/or resolution enhancement of both $F_{A,i}^E$ and $F_{A,i}^R$ to make suitable correspondence[2] between the two spaces: A_i^E and A_i^R. In other words, skill transfer basically transfers the expert rules described in terms of the expert language, into those rules described in terms of the robot internal language.

(3) Skill discovery can be represented as the mapping (from one state of the rule system to another state)

$$(S_i^R \rightarrow A_i^R) => (S_i^{R'} \rightarrow A_i^{R'}) \qquad 5\text{-}8$$

The rules generated at a learning cycle (see section 5.4) may be: (1) error-contaminated; (2) incomplete; i.e. not applicable to every situation; and (3) inconsistent; i.e. having conflicts with other learned rules. Thus, one should distinguish a rule shell from a rule: a rule shell is a rule that has incompletely learned S_i^R and/or A_i^R. A rule shell may either "survive" to become a robot rule, through skill discovery mapping, or "die out". In 5-8 $S_i^R \rightarrow A_i^R$ is a rule shell. $S_i^{R'} \rightarrow A_i^{R'}$ may be considered a rule (see examples later).

---

[2]This process is to be done by the system designer. It could, however, be automated to some extent as we have described in Section 5.4.2.1.

Actually the proposed skill acquisition model borrows many ideas from both Adams and Schmidt's theories described in section 2.4. The idea of skill transfer is derivable from Adams's concept of "perceptual traces". The "perceptual traces" described by Adams are represented in the proposed model as the basic vocabulary of the robot's internal language. The basic premise that has been borrowed from Schmidt is the development of production system rules about the robot motor behavior. This relates to Schmidt's idea of generalized motor programs (each particular movement has a parameterized motor program) for which a set of parameters must be applied to the motor program in order to perform it. Skill discovery models Schmidt's idea of rule refinement and enhancement through practice.

5.4. DYNAMICS-BASED SKILL ACQUISITION IN THE LC

The Learning Controller (LC) (see Figure 5.1) is a system that has a more constrained view[3] of learning, similar to the one adopted by expert systems: learning is the acquisition of explicit knowledge. The LC emphasizes this view by making the acquired knowledge explicit, so that it can be easily organized, verified, and modified. The LC extracts new rules from examples provided by the animator (a human expert) and learns torque profiles for particular skills using skill transfer. Furthermore, the LC allows the animator to interactively update its knowledge base by tuning the dynamic behavior of the torque profile rules through incorporating skill discovery. The LC consists of two components: a knowledge-based controller (KBC) and a learning apprentice system (LAS). The KBC consists of an inference engine, a knowledge base, and a working memory. The inference engine controls the operation of the LC by selecting the rules to use, accessing and executing those rules, and determining when a goal has been satisfied. The rules are implemented in "C" so that the KBC would have the capability to rapidly process both numeric and symbolic data. The inference engine is very simple because the data operated on are known and fixed in quantity. This has eliminated the need for pattern matching between rules and variables. Similar to other "rule time expert systems",[4] the KBC relies on techniques to compile the rules, to divide

[3]Than other machine learning systems.

[4]Refer to Russell Anderson's ping pong robot in Section 3.2.2.

the rules into smaller and smaller sequential pieces. This has allowed the KBC to integrate sensory data, robot capabilities, and task constraints to generate acceptable locomotion movements in real-time.

The knowledge base consists of three (initially empty) sets of production rules that regulate the locomotion behavior of the robot. The working memory contains the description of the current robot status (both its body and legs), the environment details (e.g. obstacle descriptions), and the goals provided by the human animator. Skill transfer is performed by the KBC components, whereas the skill discovery is performed by a knowledge-based consultant module called the "learning apprentice system" or shortly the "LAS" which is attached to the KBC (see Figure 5.1).

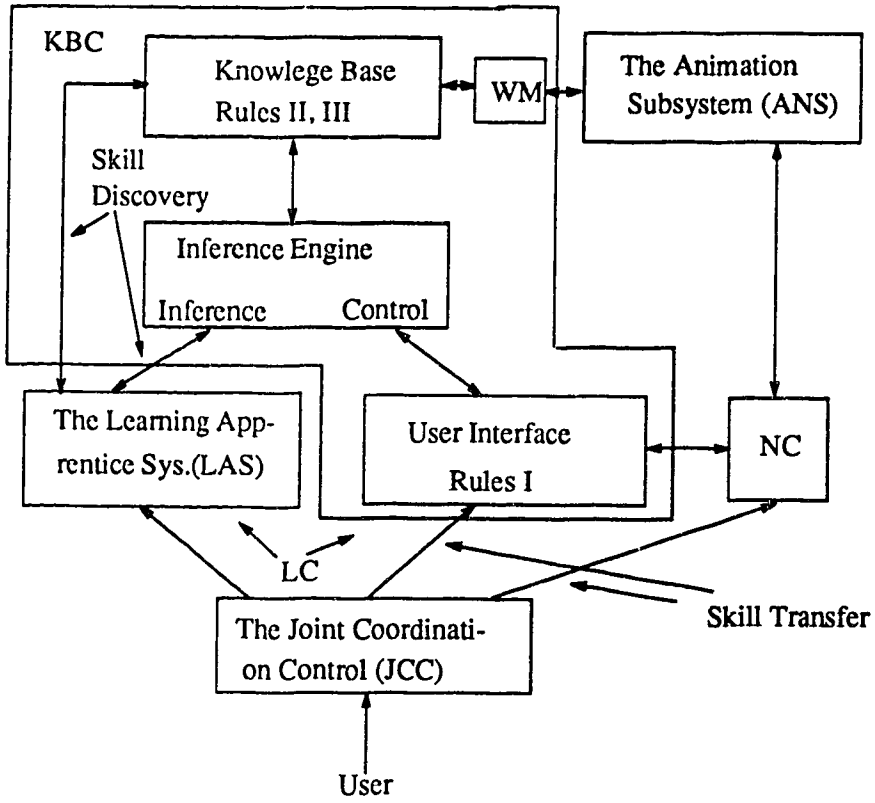

Figure 5.1 The Components of the Learning Controller

The task tackled by the LC is to provide three functions:

(1) Initially, it functions as a high-level symbolic interface to the NC (symbolic front end for choosing among and managing the execution of the NC parameters- see user interface rules I in Figure 5.1).

(2) Later, it functions as an intelligent locomotive solver which solves problems on its own when it is possible. The LC will gradually transform from a high-level symbolic interface to an intelligent locomotive solver that will in some (eventually the majority) of the cases be able to solve locomotive problems on its own, without the help of its numerical partner, the NC (see rules II in Figure 5.1).

(3) Finally, it functions as an intelligent skillful controller that is able to perform purposeful activities under unpredictable conditions in the environment (environment obstacles- see rules III in Figure 5.1).

In other words, the LC will gradually go through a natural growth process from an apprentice to an assistant to a partner and, finally, to a skillful controller.

The LC applies the skill acquisition model of section 5.3 with the following postulates:

(1) Decomposition: The LC synthesizes locomotive skills in two phases. The first phase, taking place in the legs' approach to the ground, is called the "strategic" phase. At this phase the LC determines the set of "milestones" of motion (the first approximation for the motion). The details of the motion between the milestones are not computed at this phase. The actual details of the environment (e.g. obstacles descriptions) are determined during the actual motion execution. Only then, the actual motion will be computed during the second phase which is called the "tactical" phase.

(2) Reduction to geometric primitives: Decomposition of locomotion tasks into preliminary and implementation phases means that the preparation for sustained stable locomotion is done without taking into account the details of the environment. The leg adjustments are done according to the obstacle contours. In order to make this statement operational, a reasonable assumption is to replace the actual obstacle contour by the circumscribed regular geometric body for the sake of reshaping and adjusting the legs' motions. This is what we call the reduction to geometric primitives. Implications

of the reduction postulate to geometric primitives are very important. In this way the robot can handle an infinite number of obstacle shapes contained within a primitive geometric form using just one form of preliminary skill shaping. The number of needed variations of these preliminary skill shapes is thus reduced to the number of geometric primitives (parallelepiped, cylinder, pyramid, sphere, etc.) which is quite small.

Another similar mechanism, used in the preparation for sustained stable locomotion, is related to the selection of preliminary locomotive skills. According to the assigned robot mission, conditions of the road, distance to destination, etc., the preliminary locomotive skill is chosen. On the basis of accumulated experience, it is possible to decide a priori a suitable preliminary locomotive skill. This simplifies greatly the burden of the final tactical control phase.

5.4.1. SKILL TRANSFER

At any instant during the experimental system operation (see Chapter 6), the human animator uses the animation front end subsystem to describe the locomotive skills to be used by the robot. The human animator would provide the required input for the JCC and the NC modules (speed control, heading control, gait control, etc.). As the animator controls the motion of the robot, simultaneously the development of motor skills in the form of production rules takes place through the skill transfer process. Remember this is similar to the case in child motor control systems where motor control and motor learning are parallel activities- see Chapter 1. This involves building the knowledge base of the LC, which consists of three (initially empty) sets of production rules of the form 5-4. In these rules, the LC learns both a generalized left-hand side (see section 5.4.2.1) and a generalized right-hand side (see section 5.4.2.2) . The left-hand sides are generalized by using either concept-learning techniques or by turning constants into variables [Ric83], while the right-hand sides are developed by concatenating subplans or by editing motion torque profiles. A rule that still has an incompletely learned S_i^R and/or A_i^R is called a rule shell (a more precise definition will be given later). These rule shells may either survive to become a robot rule, through skill discovery mapping, or die out. Examples of rule shells from the three rule sets of the knowledge base

follow.

Type I rule shells:

These are skill usage type rules. They identify the contexts under which the various skills learned by the robot should be used and how to drive the NC to implement them. An example of a Walk(A,B) rule shell is:

$$IF\ LEG_1[lc_i(X)]\ \&\ LEG_2[lc_i(Y)]$$

$$\&\ LEG_3[lc_i(Z)]\ \&\ LEG_4[lc_i(W)]$$

$$\&\ NAVIGATION\ GOAL = A \rightarrow B$$

$$\&\ ROAD(A,B) = FLAT$$

$$\&\ DISTANCE(A,B) \leq 5$$

$$\&\ INITIAL\text{-}LOC(A) = NS\ STREET$$

$$\&\ FINAL\text{-}LOC(B) = NS\ STREET$$

$$THEN$$

$$ACTIVATE\ NC\ [lc_1(X'), lc_1(Y'), lc_1(Z'), lc_1(W')]$$

$$UNTIL\ X_E = X\text{-}coordinate\ of\ the\ final\ loc(B)$$

$$\&Y_E = Y\text{-}coordinate\ of\ the\ final\ loc(B)$$

$$WITH\ linear\ speed = [a,b]$$

$$DELETE\ FROM\ WM:$$

$$old\ LEG_i\ locs$$

$$ADD\ TO\ WM:$$

$$INITIAL\text{-}LOC(B) = NS\text{-}STREET$$

$$\&LEG_1\ [lc_1(final\ pos)]$$

$$\&LEG_2\ [lc_1(final\ pos)]$$

$$\&LEG_3\ [lc_1(final\ pos)]$$

$$\&LEG_4\ [lc_1(final\ pos)]$$

This example represents a rule shell for the usage of a locomotive skill: walking. It provides the conditions under which this skill should be selected to implement sustained stable robot locomotion from A

to B. In the action part, it shows how to derive the NC to implement the skill. These rule shells are learned by observing the examples provided by the animator using the JCC and the NC (see Chapter 6). Skill transfer takes place as the mapping of both the situation and action spaces from the human animator's specifications of the JCC and NC control commands, the characteristics of the environment, and the robot's state (in other words the contents of the WM at the time of using the skill) to the left and right hand sides of the rules respectively. The features of the situation parts of the rules are collected from the contents of the WM to form the rules' S_i^R. Similarly, the rules' A_i^R features are extracted from the animator's actions to maintain the robot's sustained stable locomotion.

Intuitively, this rule says: If the robot's four legs are in some orientation (X,Y,Z,W) within a particular leg cycle (i) (see Figure 3.5) ,and the goal broadcast by the animator is ($A \rightarrow B$) such that the road from A to B is flat and the distance between these two points is less than or equal to 5 units of distance, and both points are located on NS (North- South)streets; then start to drive the NC with the lc_1 leg cycles for all legs such that the legs start from the closest walk cycle to the initial legs' settings (X',Y',Z',W') in lc_1 (note that each locomotive skill has its own leg cycle- figure 3.5). This is to facilitate the smooth transitions between different types of gaits. The linear body speed is within the range [a,b], and the stopping conditions are of the destination point B (within some tolerance). The additions and deletions from the working memory (WM) are for the purpose of the LC's consultation mode (see later).

Type II rule shells:

These are similar to type I rule shells, but they use torque profiles instead of leg cycles in the right hand sides of the rules. They are called "prototypical torque profile" rules. They are formed from type I rules by replacing the right hand sides with the torque profiles calculated by the NC when the S_i^A parts of the type I rules are stabilized (see rule stabilization definition in section 5.4.2). A rule shell for a walk skill in the type II rule set would be as follows:

$$IF\ LEG_1[lc_i(X)]\ \&\ LEG_2[lc_i(Y)]$$

$$\&\ LEG_3[lc_i(Z)]\ \&\ LEG_4[lc_i(W)]$$

& NAVIGATION GOAL = A →B

& ROAD (A ,B) = FLAT

& DISTANCE (A ,B) ≤ 5

& INITIAL–LOC (A) = NS STREET

& FINAL–LOC (B) = NS STREET

THEN

DRIVE THE DIRECT DYNAMICS MODULE WITH WALK (X',Y',Z',W')

UNTIL X_E = X–coordinate of the final loc (B)

& Y_E = Y–coordinate of the final loc (B)

DELETE FROM WM:

old LEG_i locs

ADD TO WM:

INITIAL–LOC (B) = NS–STREET

& LEG_1[lc_1(final pos)]

& LEG_2[lc_1(final pos)]

& LEG_3[lc_1(final pos)]

& LEG_4[lc_1(final pos)]

The action part of this rule is learned by a skill transfer process. That is, the torque profile computed by the NC is used directly (WALK(X',Y',Z',W')[5] in the example- the format of this profile is described in section 5.4.2.2) with no need for any processing to understand or interpret these torque profiles because they are memorized to be used later in similar situations. Type II rules are learned to provide the function of the intelligent locomotion solver which solves problems on its own without the help of the NC. Building these rules is nothing but priming the LC with the torque profiles produced by the NC (previous experience- A_i^E transfer to A_i^R in 5-7), but in compact form since these rule shells have been firmed up by generalizations and discrimination throughout their type I lifetime (see Section 5.4.2). The left hand sides'

[5]The orientations (X',Y',Z',W') are the closest leg orientations to the current (X,Y,Z,W) leg orientations (due to current task segment conditions) in the WALK torque profile table.

torque profiles are used to directly drive the direct dynamics module (see chapter 6).

Type III rule shells:

These are the motion-enhancement and fine-tuning rules that are to be used in the tactical phase during motion execution. They identify the kinds of local terrain operators that must be applied to the legs' driving torque profiles in order to adapt them to the roughness of the terrain. An example of this type of rule is:

IF LOCAL MAP = BUMP ahead of leg_i *inWM*

& BUMP CHARACTERISTICS ARE (STEEPNESS, FRICTION, SIZE, ETC.)

THEN

APPLY OPERATOR BUMP (STEEPNESS, FRICTION, SIZE, ETC.)

TO leg_i *'s TORQUE PROFILE*

Generally speaking, stepping over obstacles requires some kind of sensor ranging system [EsA84], and sophisticated control algorithms to provide the capability to step over many different types of obstacles when moving at different speeds [SoW87]. Two approaches are implemented to deal with small obstacles. The first, as shown in Figure 5.2, uses the terrain scanner information (see Section 6.2) to detect small obstacles. If any obstacle is detected within a leg forward stroke, then the leg will be continuously moved upward until the path is free from the obstacle (point B). Then the leg is moved forward to point C, a position which has the same x component (forward) as the end position of the current leg cycle. Finally, it is moved downward to the position, point D. If the obstacle is too wide then the leg will land on top of the obstacle. These forward and upward/downward motions' torque profiles are stored in the fine tuning rules during the teacher's editing the leg cycle to avoid obstacles.

The second approach integrates a ground tracking and obstacle clearance modes as shown in Figure 5.3. In the beginning, with no obstacle in the leg transfer path, information from the terrain scanner is used to track the ground (constant distance between foot and ground is maintained). The tracking will continue until an obstacle is detected within a certain range by a leg forward stroke. In this case the leg switches to the first approach to step past or on top of the obstacle. Then the leg will switch back to keep track of the

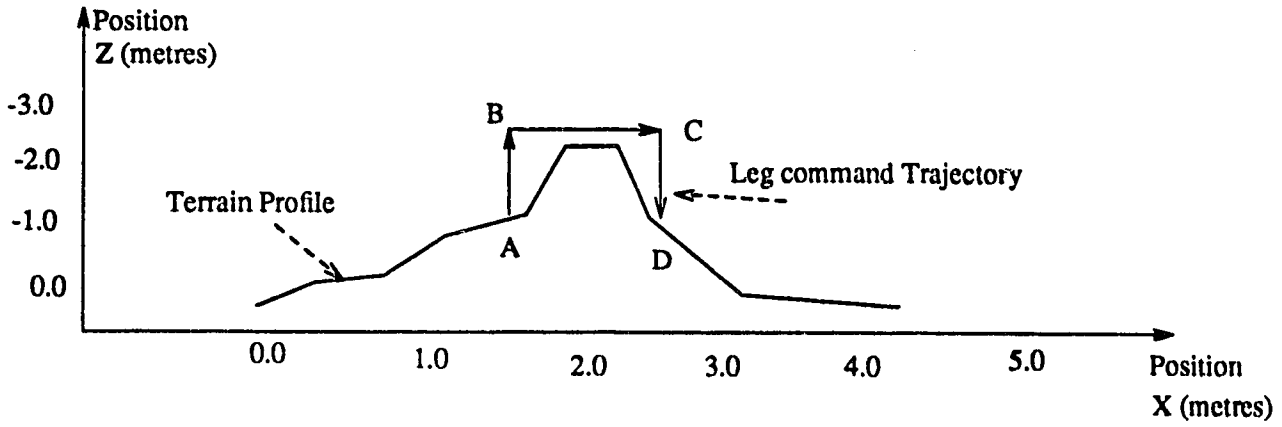

Figure 5.2 Obstacle Clearance Strategy for Stepping over Obstacles

ground.

Each leg updates its own torque profile independently. Since there is only one on-board camera on-top of the robot, hind legs use extrapolation technique to predict the location and where in their cycles they will meet the obstacles. Hind legs use the same approaches as the front ones to navigate. This was possible since the swinging legs are not contributing any torques to the motion of the robot body. Accordingly, only the trajectory of a swinging leg had to be adapted to the shape of the navigated terrain. The effect on the robot body's orientation appears after a swinging leg lands. In this case the automatic body regulation algorithm is activated to maintain the robot balance. The automatic body regulation algorithm is always activated when either the NC or the LC is in control of the robot locomotion. It is actually implemented within the direct dynamic code (see Sections 3.4.2 and 6.3).[6]

In order to analyze the performance of the fine tuning rules, the leg positions (in Cartesian coordinates) and the data from the terrain scanner were stored for off-line analysis. The data stored was used for generating the terrain map (off-line). This experiment is described in section 6.5.

[6]Refer to the navigation algorithm to see how it decides which obstacles to overcome (step on top) and which to circumnavigate (section 6.2).

One set of these fine-tuning rules is associated with each of the robot's four legs. They are responsible for the short-term monitoring of the situation during motion execution and synthesizing of the actual motion trajectory according to the obstacles that should be overcome by the legs. The right-hand sides of these rules are formed from observing the animator editing the leg's torque profiles or joint angles to overcome the obstacles on the road and to enhance the produced motion (see chapter 6). The difference between the original torque profile (before editing) and the edited torque profile is stored as an additive quantity in the operator BUMP in the given example. When a similar obstacle is faced by *any* of the legs while executing *any* leg cycle (*any* locomotive skill), this terrain adaptation torque profile will modify the current torque profile additively. These rules are learned to provide the function of the skillful locomotive controller, which solves locomotive problems under unpredictable environmental conditions. Figure 5.4 show the conceptual structure of the LC and where these three sets of rule shells are located.

The torque profile of the NC produces only an approximate version[7] of the desired motion (this is the strategic phase of the skill synthesizing). The LC learns motion adaptation during type III rule learning (the tactical phase).

In fact, type II and III rules form a hierarchy (preliminary followed by implementation phase) in which type II rules first select the skill to be executed and pass the torque profiles as the first approximation of the motion to the legs. Then, type III rules of each leg will edit their torque profiles on the fly during the actual execution of the motion in order to enhance the produced motion. The traversability of the terrain and the set of small obstacles "faced" by the legs will be determined only during the actual motion (in WM- see Figure 5.1). Obstacle information representing sensory information, is supposed to be provided by sensory devices such as a robot's on-board camera.

Motion on rough terrains is learned by allowing the animator to manipulate the motion produced by the NC (the torque profiles). If the animator sees that the produced motion is incorrect (will disturb any of the sustained stable locomotion requirements- see Chapter 6), he or she has to modify, recompile, and re-

[7]See the end of Chapter 4.

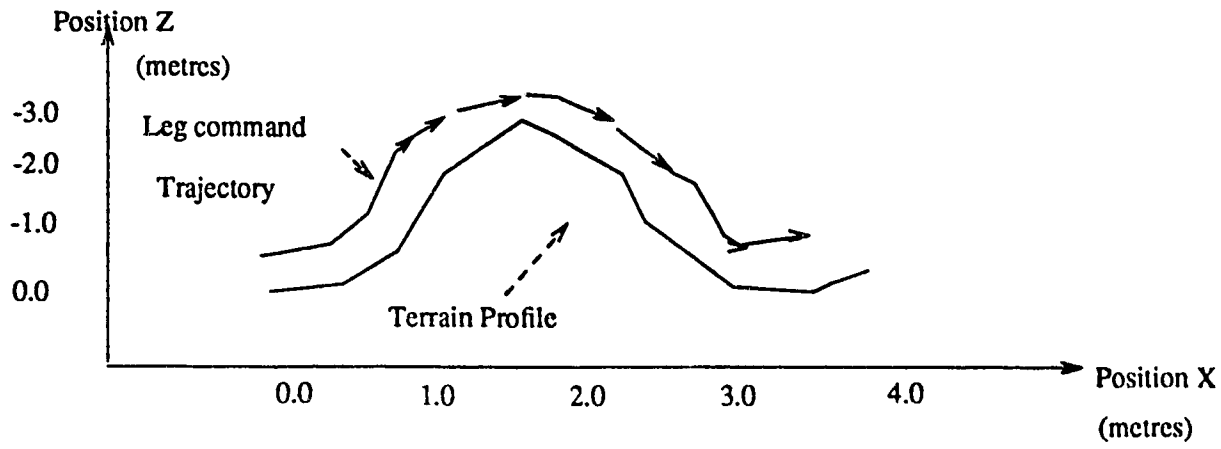

Figure 5.3 Ground Tracking Strategy

execute the edited torque profiles. This mechanism will provide control over the motion without disrupting the dynamic integrity of the resulting motion since the edited torque profile still has to drive the direct dynamics module in order to produce the motion. This mechanism represents a solution to the problem of controlling and coordinating the motion using torque profiles (developing dynamics-based skills). The torque control will be no longer unfamiliar to animators [AGL86]. The LC will build locomotive rules about the torques required to produce different kinds of locomotive skills, even under various difficult terrain and environmental conditions.

This is, to some extent, similar to work done at NYIT, where the animators have an interface that allows them to manipulate the animation database (at the frame level) produced by the BBOP keyframe animation system [Lun87] (this system was described in section 2.2). The BBOP provides the first approximation of the motion, and the animator uses a keyframe editor to correct it. The animation database is reused by the BBOP at any time for motion review and interactive modification. In other words, the editing techniques are used as post-processes to the BBOP animation system.

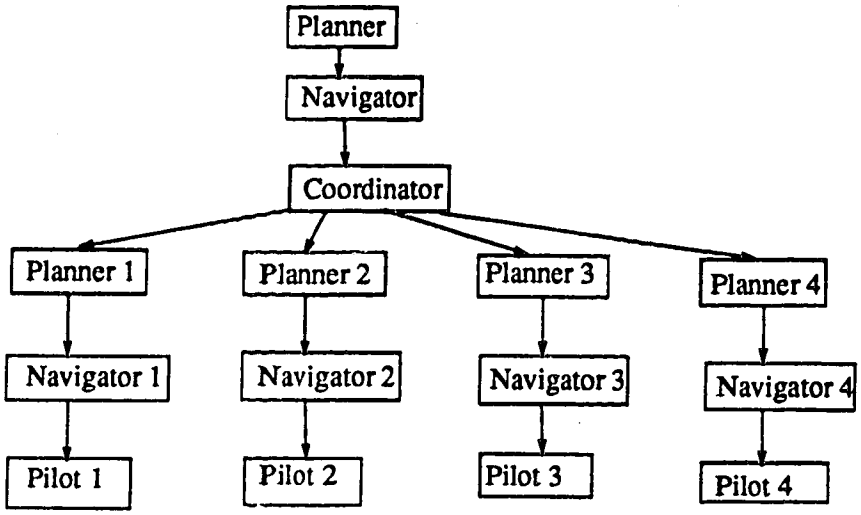

Figure 5.4 The Conceptual Structure of the Learning Controller

For example, Lundin used the BBOP system first to animate a vehicular model that was moved over a path, but the wheel motion was not performed. He then used a simulation program to read the BBOP database, perform the wheel motion, and write the modified transformation parameters back to the database. The effect of the simulation on the animation could then be viewed using the BBOP again. Further modifications were made by either changing the input parameters to the simulation algorithm or by changing the motion database with BBOP.[8]

With the skill transfer strategy described in this section, the LC will have the flexibility to achieve high performance and smooth locomotive motion through "tuning the dynamic behavior" of the robot. The objective here is the capability of specifying motions which combine the realism of dynamics simulation in real-time without removing control from the animator.

[8] Again here we would like to emphasize on the kind of animated simulations that the experimental work of this dissertation describes. What we are dealing with here is what is called "task level simulation", where the problem is to control the simulated robot by specifying a set of events and some constraints on its behavior, and letting it fill in the details as necessary.

5.4.2. SKILL DISCOVERY

Skill discovery describes how the LC can generate its own skills by refining and exploring those skills which have not been presented by the human animator in operational form. Skill discovery is done by the learning apprentice system (LAS) module. Learning Apprentice Systems (LASs) are knowledge-based consultant systems that directly confront the knowledge acquisition bottleneck by learning from interactions with an information-rich external environment (such as the human animator). Several previous LAS systems have been built in the field of machine learning in artificial intelligence. LASs represent autonomous learning systems that start with general inference rules and learning techniques and gradually acquire complex skills and knowledge through continuous interaction with a human expert. Learning Apprentice Systems are currently being developed in the domain of VLSI design [MMS85], well-log interpretation [WSK86], and medical diagnosis [WCB86].

All the rules that are learned by skill transfer in the previous section have a situation part (left hand side) that consists of formulae of conjunctive predicates (the feature set F_i^R along with their values), with truth values assigned to them. Skill discovery is achieved through rule-modification. The rules are modified because they contain faults, which can be of two types: (1) motion faults: a rule contains an action that calculates incorrect motion (incorrect right hand side); (2) control faults: the rules have undesirable control behavior when run in the consultation mode (incorrect left hand sides).

The main learning algorithm for the LAS is:

ALGORITHM A

Until all the rules are stabilized (see the definition of rule stabilization later):

(a) Identify a rule with a fault from the rule set.

(b) Modify the selected rule to remove the fault.

Following Smith et al. [SMC77], the module responsible for identifying faults is called the "critic". The module responsible for modifying the rules is called the "modifier".

The "critic" identifies faults by running the existing rule shells (incompletely-learned rules) on the current navigation task (see algorithm C later that describes the operation of the LC in consultation mode) and then analyzing the resulting rule trace. The analysis must identify where the rules behaved correctly, called positive training instances, and where they behaved incorrectly, called negative training (fault) instances. The positive instances are used to generalize the rules and the negative instances to correct them.

Negative fault instances can be of two types:

(a) Errors of commission: A rule is fired incorrectly, because it was insufficiently constrained.

(b) Errors of omission: A rule did not fire, either because it was incorrectly constrained, or the required rule simply does not exist.

The "modifier" requires three pieces of information on each instance:

(a) The type of instance: positive, negative-commission, negative-omission;

(b) The rule;

(c) The context, consisting of the variable bindings when the rule was fired.

A very common critic technique is the use of an "ideal trace",i.e., an account of what rules should have fired and in what sequence. The LAS takes the ideal trace as input from the animator when the animator uses the JCC and the NC to produce the details of the motion for a mission. The ideal trace is compared with the actual trace of the rules (the rule trace) to locate the first point at which the traces differ. This allows the faulty rules to be identified. In other words, the "critic" algorithm is:

ALGORITHM B

(a) Produce the rule trace by running the rule shells on the current navigation task (see algorithm C later).

(b) Compare the produced rule trace with the ideal trace and find the first place at which they differ.

(c) The rules that fired before this point fired correctly, so the associated contexts are positive training instances for those rules (see algorithm D).

(d) The LC rule which fired at the differing point caused an error of commission, so the associated context is a negative training instance for this rule (see algorithm E).

Once a fault has been located, the faulty rule can be modified. The modification module uses three rule modification techniques: rule ordering, left hand side learning, and right hand side learning.

(1) Ordering the rules, e.g. specifying that $S_i^R \rightarrow RUN$ should always be fired in preference to $S_j^R \rightarrow WALK$ if both are in the current conflict set during an LC's consultation mode. This technique is only appropriate for control faults and was used by Brazdil and Waterman [Bra78] [Wat70].

(2) Updating a rule's situation (left hand sides learning) to take account of new instances, e.g. transforming

$$S_i^R \rightarrow RUN \quad => \quad S_i^{R'} \rightarrow RUN \qquad 5\text{-}9$$

where $S_i^{R'}$ is derived from S_i^R by the left hand sides learning techniques (described in the next section). This is defined as the forming of a symbolic description of the target conditions from examples and non-examples and then using it for prediction -like those used by Winston [Win75].

(3) Modifying a rule's action (right hand sides learning) to correct the motion produced by the torque and force profiles used.

$$S_i^R \rightarrow A_i^R \quad => \quad S_i^R \rightarrow A_i^{R'} \qquad 5\text{-}10$$

where $A_i^{R'}$ is derived from A_i^R by right hand sides learning (see section 5.4.2.2).

5.4.2.1. LEFT HAND SIDE LEARNING

In order to understand the rule modification technique of 5-9, one has to learn about three concepts: the Situation Space, Generalization, and Discrimination.

(i) The Situation Space

The situation space tries to capture the notion of a partially-specified left hand side in which some situations are known to lie outside the target conditions, some inside, and some, in a grey area, are yet to be decided. The LAS works to reduce this grey area. Each rule's S_i^R part is represented by a data structure

called its situation space.

A situation space consists of a forest of feature-value trees (see Figure 5.5). Each tree corresponds to a conditional clause in the S_i^R of a rule. Each node of a tree is labelled with a feature-value, feature-values in the same tree being applied to the same arguments. The label of the root node is the feature value TRUE. These feature-values represent the robot's internal language vocabulary that are acquired by the LC during the skill transfer process. The labels of the sons of a node are mutually exclusive and exhaustive. The situation space of figure 5.5 shows some of the feature-value trees that might appear at the rules' left hand sides. Other feature-value trees concern the features of the sustained stable locomotion of the robot (these features were mentioned in section 3.2) such as: stability tree, combining locomotion tree, speed trees, obstacle avoidance tree, etc.

This situation space allows a partially-specified rule situation (left hand side) to be represented. During the course of rule learning, this partially formed situation is gradually firmed up until it is either stabilized or completely specified (see definitions next paragraph). The partial representation is achieved by placing two markers in each tree, an upper mark and a lower mark, as in Figure 5.3. The partially-specified rule is represented by the rule action (e.g., RUN) and the situation space together with its marks. This is what we call a rule shell.

The following definitions concern the manipulation of the trees in the situation space:

(a) A feature-value is said to be above a mark if it is outside the subtree dominated by that mark. Similarly, a feature-value is below a mark if it is in the subtree dominated by that mark. A feature-value is between the upper and lower marks if it is above the lower mark and below the upper mark.

(b) Any feature-value above the upper mark is outside the target left hand side. Any feature-value in the tree below the lower mark is inside the target left hand side. Any feature-value between the upper and lower marks is in a grey area, about which the LAS is not sure.

(c) A conditional clause is "firmed up" when the upper and lower marks coincide. The rule shell are firmed up into a rule when each of its conditional clauses are firmed up. Learning works by moving

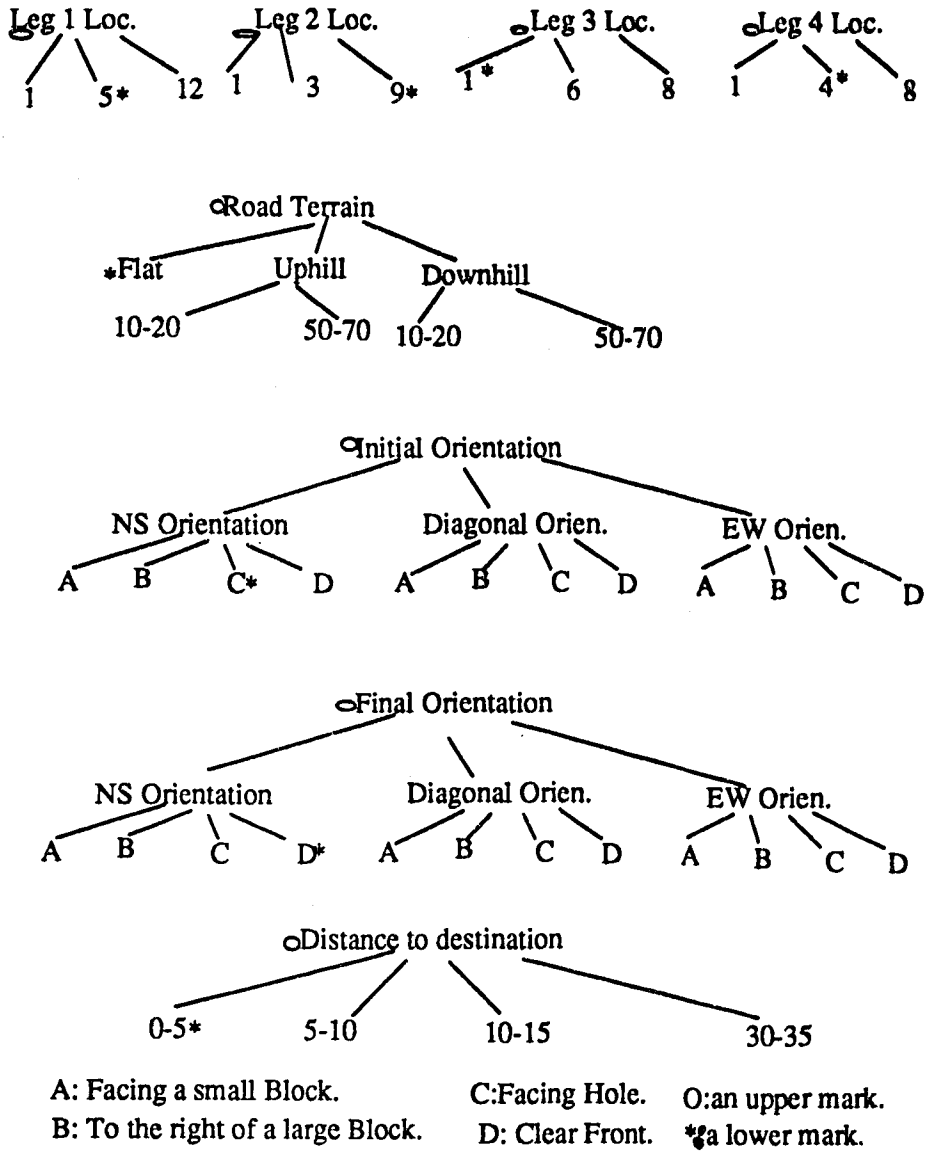

Figure 5.5 The Situation Space Consists of a Forest of Feature-value Trees

the upper marks down and/or the lower marks up, until they coincide.

(d) A rule shell is stabilized when the upper and lower marks in all tree forests freeze for a reasonable period of time during the learning process.

Since the rule shell partially specifies the rule, there is some ambiguity about what rule to use when forming rule traces (this is needed in step (a) of algorithm B above). Following Mitchell [Mit83], one can take two extreme views:

(1) the most general view: that the left hand side is specified by the conjunction of feature-values labelling its upper marks, which leads the rule to make errors of commission; and

(2) the most specific view: that the left hand side is specified by the conjunction of feature-values labelling its lower marks, which leads the rule to make errors of omissions.

Our technique for dealing with this predicament is to use the following algorithm:

Algorithm C (The LC consultation mode algorithm)

(a) All the corresponding trees in the forests of the various rule shells are merged into one forest keeping various marks of the different rule shells.

(b) Each working memory datum of the current context (coming from the working memory WM) is set into its designated tree in order to identify the new conflict set. The new conflict set consists of elements of the current conflict set where the working memory datum is located below their upper marks in the designated forest tree (i.e either within the current left hand side or in the grey area).

(c) Repeat (b) until all the working memory data that represent the current context are exhausted.

(d) The final conflict set is ordered according to the rule-ordering strategy (the priority ordering) , and the first rule on the list is applied.

This matching and rule selection algorithm is the inference engine of the KBC (see Figure 5.1). It is expected to work well in the process of forming rule traces for the following reasons:

(1) Only few rules in each rule type exist. In the experimental system of chapter 6, only 14 of them and their action parts implement the following locomotion skills: Climb, Descend, Walk, Turn-left, Turn-right, Walk-backward, Trot, Pace, Gallop, Stop, Run, Pronk, Walk-side-ways and Turn-in-place.

(2) Also few forest trees in the situation spaces : Leg_1 loc., Leg_2 loc., Leg_3 loc., Leg_4 loc., Road terrain, Initial loc., Final loc., Distance, Navigation goal, also speed range trees.

This means that the conflict set can be at most of size 14, and the number of forest trees in any situation space can be at most 12.

The partial representation of a rule provided by a rule shell is similar to the version space representation used by Mitchell et al. in the Candidate Elimination Algorithm [MUN81].

The raising of the lower marks of the Situation space forest trees is done when the critic provides a positive training instance of a rule. This is called generalization of the rule shell. Moving the upper marks from the root, down a tree, is done when the critic provides a negative training instance of the rule. This is called discrimination of the rule shell.

The learning process compares the current context of the environment (coming from the WM) with all previous contexts. This is possible because all previous contexts, both positive and negative instances, are summarized by the current positions of the upper and the lower markers in the feature trees. The learning process need only compare the current context with the current position of these marks. If the critic has provided the modifier with a positive training instance, then we will have a positive context and will apply the generalization algorithm (see algorithm D below). If the critic has provided the modifier with a commission error, then we have a negative error, and the modifier will apply the discrimination algorithm (see algorithm E below).

The situation space is initialized during the skill transfer process. For each tree in the situation space, exactly one of its tip feature-values will be specified by (i.e, be true in) the context of this instance. The lower mark is placed on this tip. The upper mark is placed on the root of the tree.

(ii) The Generalization Algorithm:

The generalization algorithm receives the context of a correct application of a rule and the situation space of the rule. The output consists of new lower marks for some of the trees. Each tree is considered in turn and the following steps are executed:

ALGORITHM D (The LAS generalization algorithm)

(a) For each of the feature-values labelling a tip node, determine its truth value in the context.

(b) Exactly one of these feature-values will be specified by the context. Label this node "the current node".

(c) Find the least upper bound of the current node and the current lower mark and make this the new lower mark.[9]

(iii) The Discrimination Algorithm:

The discrimination algorithm receives the negative context of an incorrect application of a rule and the situation space of the rule. The output consists of a new upper mark for exactly one of the trees. Note the lack of duality between the generalization and the discrimination algorithm. Since we are dealing with a conjunctive rule, all its conditions must be true for the rule to fire. Thus, making one condition false for an instance is enough to prevent the rule firing ("which one?" is a credit assignment problem). Each tree is considered in turn and the following steps are executed:

ALGORITHM E (The LAS discrimination algorithm)

(a) For each of the feature-values labelling a tip node, determine its truth value in the negative context.

(b) Exactly one of these feature-values will be true in the negative context, label its node, the current node. Note that the current node must lie below the upper mark, otherwise the rule could not have fired.

(c) If the current node lies below the lower mark, then mark the tree as a white tree.

(d) Otherwise, the current node must lie between the upper and lower mark. Mark the tree as a grey tree.

At least one of the trees must be grey, otherwise the rule shell application would be correct. If just one tree is grey, then we have a near miss. If more than one tree is grey, then we have a far miss.

[9]Despite changes to the lower marks, the rule shells used in the planning phase do not change form, because they are determined by the upper marks. However, generalization does have an effect on the rule-learning process, because the lifting of the lower marks can limit the choices available to discrimination.

Only one of the grey trees can have its upper mark lowered. We call this grey tree the discriminant. Far misses (a kind of credit assignment problem) introduces a choice, and therefore gives this discrimination algorithm a non-deterministic nature. In the case of far misses, an arbitrary grey tree in chosen.

(e) Once the discriminant has been picked, its upper mark is lowered, just enough to exclude the current node. This is done by setting the new upper mark to be the unique mark below the least upper bound of the current node and the lower mark that is still above the upper mark. (Also here, note lack of duality with generalization,i.e. we don't use the greatest lower bound of the upper and current mark).

Actually tree attributes and values are similar to generalization trees in concept learning systems [MMS85]. They are provided by the system designer. This process could be automated to some extend. For example, an inconsistency can be caused by an inadequate situation space. For instance, suppose the correct form of a "Gallop" rule is:[10]

$$Leg\,1\text{–}loc\,(X)\ .and.\ Leg\,2\text{–}loc\,(Y)\ .and.\ Leg\,3\text{–}loc\,(Z)\ .and.\ Leg\,4\text{–}loc\,(W)\ .and.....and.$$

$$[uphill\,(=a)\ .or.\ downhill\,(=b)] \rightarrow Gallop\,(X',Y',Z',W').$$

Since the road-terrain tree is ternary, positive instances for Gallop on an uphill and Gallop on a downhill will cause Generalization to move lower bound to the root node. This particular Gallop rule shell will then be:

$$Leg\,1\text{–}loc\,(X)\ .and.\ Leg\,2\text{–}loc\,(Y)\ .and.\ Leg\,3\text{–}loc\,(Z)\ .and.\ Leg\,4\text{–}loc\,(W).....\rightarrow Gallop\,(X',Y',Z',W').$$

Now a negative instance for Gallop on "Flat" surface will be below the lower bound and hence cause an inconsistency. The solution in this case is to manipulate the road-terrain tree into the form shown in Figure 5.6.

In summary, the characteristics of the learning technique of the LAS in dealing with the situation parts (left hand sides learning) of the rule shells are as follows;

---

[10]The =a and =b variables here means any bindings.

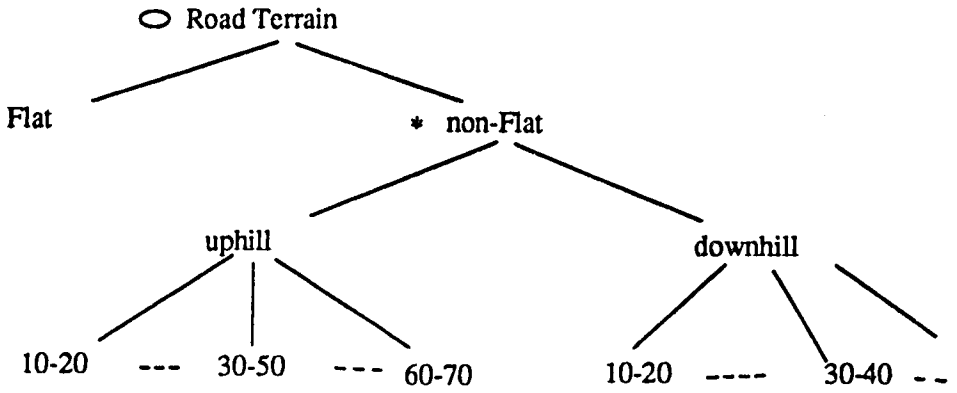

Figure 5.6 Manipulating a Feature Tree to solve an inadequancy in the Situation Space

(1) It combines generalization and discrimination in a clean manner. They are near dual, but discrimination is nondeterministic, whereas generalization is not.

(2) The firming up of a rule shell in the learning procedure provide guarantees that the learning process has terminated. No such guarantees are provided by generalization or discrimination used alone.

(3) The stabilization of a rule shell in the learning procedure guarantees that right hand sides' torque profiles will take place (see next section).

(4) Rule ordering is independent of the modification technique used for the left hand sides.

(5) Discrimination on far misses introduces choice and search into the modification process.

Finally, concerning the creation of new rules, one obvious technique that could be used is to treat the rules' absence as an error of omission and use the standard techniques to correct this error. The idea is to modify rules that have no conditions from the situation space, only a conclusion. Such rules may be called empty rules. An empty rule, together with its situation space, constitutes an empty shell.

5.4.2.2. RIGHT HAND SIDE LEARNING

In the LC, when the situation part of type I rules stabilizes (stability can be sensed when the markers stop moving around the trees of the situation spaces), the LAS substitutes the right hand sides of these rules with the torque profiles calculated by the NC. Motion computations by the NC require time-consuming calculations, and so it is desired that the motion experienced in the past be executed without complicated processing (easier and faster abilities of skills reproductions).

The torque profiles of the various degrees of freedom of the legs depend on two factors: the desired joints trajectories and the desired body's trajectory. Type II rules combine these two and remember the torque profile calculated by the NC that produced the motion. The torque profile is a 13 column look-up table indexed by the four legs tips' locations. Figure 6.9 show a torque profile for a complete walking cycle. For each leg, three joint torque profiles are stored in these look-up tables (each leg has three DOF)- see Table 6.2. Only one cycle for each skill is stored in a look-up table. This will be repeated as many times as required (see Chapter 6).

In this context, it is important to recognize that each leg's torque profile is considered in the context of the other legs' torque profiles. Because of the inter-dynamic coupling between the links of the body, a torque profile for one leg alone does not say much about the motion of the robot's body. This is the main reason why the granularity of the torque profile lookup table cannot be brought down to a link or even a leg level.

The teacher detects motion bugs during the motion execution on the screen when the LC takes over the control of the robot locomotion (consulation mode- see Chapter 6). There are two types of bugs: illegal operation, and failed steps. An illegal operation is one that is considered impossible in the environment. For instance, it is illegal to walk through an obstacle or for the leg to go under the ground. A failed step is one that does not achieve its goal for the designated time span. The LC verifies that at all times the goals intended by the animator have actually been met. These two methods of detecting motion bugs state that a navigation plan must be executed legally, achieve all intended goals and subgoals, and also be correct. The animation halts whenever one of these problems is identified by the animator (see Chapter 6). A trace of the

driving torque profiles or joint angles is presented to the animator upon request to allow him to fix it and correct the motion.

The micro-details of the roads are presented from the "micro map" frames described in Chapter 6 to WM at the appropriate time instances to show the robot what it is "facing" at any given instant of time. The legs, in turn, will edit their torque profiles on the fly to avoid or overcome the "seen" obstacles. It is important to realize that the robot has no control over what the environment will throw its way before actual motion execution. "Micro map" information represents sensory information that is supposed to be provided by sensory devices such as cameras.

The approach used in solving the problem of the legs' perception of the environment is simplified by using a set of standard geometric shapes to describe it. We avoid modeling sensory input more realistically in order to concentrate on the control problem. This is what we called the reduction to geometric primitives in section 5.3. Perception is then reduced to recognizing these situations and determining the values of the characteristic parameters. According to the recognized obstacle and the LC's experience in handling previous similar obstacles, it modifies the prototypical torque profile that performs the appropriate locomotive skill. New operators for torque modifications are created if the user has intervened and modified the torque profile himself. A corresponding rule for this newly recognized obstacle is created as a type-III rule. The rule will relate the conditions under which the motion editing has occurred and the modifications that have been made by the animator. The difference, between the original torque profile (before editing) and the edited torque profile, is stored as the action part of the rule.

The difference torque profile is a four column table that contains a column for each DOF of a leg (see Chapter 6). The first column is an index to the leg's cycle. These values will be added to their corresponding torque profile values for this particular leg.

In summary, the torque profile modifications must be carefully monitored in order to take into account unexpected changes in the environment. Therefore, the LC will have two functions: (1) a real-time response to stimuli from the local legs' navigators reporting a particular obstacle, and (2) a decision-making

capability to select which torque modification operator to execute.

5.5. DISCUSSION

The goal of this Chapter was to explore methods by which the human-synthesized robot locomotion skills may be captured by the robot's LC in order for them to be reproduced easier and faster. The LC is meant to have a long-term learning experience unlike conventional learning artificial intelligence programs that are turned off after running a few minutes and acquiring their desired concept. As such, learning is meant to be gradual, incremental, and subject to periodic refinement.

One of the benefits of the idea of skill acquisition is that, it may turn out that by analyzing the role of automatic mechanisms in the execution of functional motions by AI methods, one can get a better insight into the operation of neural networks and the evolution of learning processes. Research towards such an objective has been undertaken in [MoA88a].

It should be also kept in mind that skill-based AI systems are not limited just to the skill acquisition problem. Very interesting and instructive learning problems can be conceived in this area of research as well. For example, assuming that rule based forward walking control is available, one can formulate the seemingly simple learning problem: what constraints and rule modifications must be introduced in order to generate the knowledge base for walking backward (similar problem could be formulated for stair climbing and descending). Other problems that might produce more insight into the evolution of the control system are investigated separately in other work [MoA88b] [MoA88c]. In [MoA88b], a methodology to measure the progress in the learning process is proposed. As indicated previously in Chapter 2, that learning, at our current level of knowledge, cannot be measured directly, as the processes leading to changes in behavior are internal and usually not available for direct examination. Rather, one must infer that learning (the process) occurred on the basis of the changes in behavior that can be observed. The work in [MoA88b] answers the questions; how can one know if the robot control system has evolved into a control system that is in some sense intelligent and how can one measure the LC progress? (see Section 6.5).

In [MoA88c] a unifying expert animator model for animating a group of articulated robots (such as the one in this research) in a three-dimensional environment, has been proposed. In a sense, the multi-robot simulation is a natural extension to the single robot simulation.[11] However, the multiple robot extension brought up two research issues that do not appear in the single robot problem:

(1) The method used for planning the motion for the single robot is based on the assumption of a static environment (see section 6.2), and so it cannot be used in the multiple robots case because each of the robots is in a dynamic environment consisting of other moving robots;

(2) The technique used to animate a single robot's motion is based on calculations of the dynamic equations of motion. Producing the motion dynamically for the multiple robot case would need greater processing capability than is currently available.

In order to overcome these two problems and to produce "convincing" animations (as opposed to simulations) for the group of robots without pressing the user to become overly involved in the mechanisms of producing the motion , an expert animator agent for each robot has been proposed. These agents have a computational understanding of motion and its semantics in a way that each individual articulated robot would handle its motion autonomously. Each agent integrates knowledge engineering approaches, namely object-oriented programming and rule-oriented programming [Rob81] [Ste85], with computer animation approaches. The object-oriented approach plays a key role in the modeling of the robots' inter-relationships, whereas the intelligent functions of each expert animator agent are transparently programmed in rule-oriented programming style. In producing animations for the various robots, each robot is considered to be an object in the environment and handles its motion and interactions with other robots, as well as with the animator, autonomously.

These problems are challenging ones especially at this early stage of the development of skill based AI systems. Hopefully, the study of learning problems at the skill level will produce more insight into the evolution of learning processes. More comprehensive skill acquisition methodology will require extensive

[11]One can examine here the possibility of transplant learning from one robot to another and also the possibility of group learning in the dynamic environment.

multidisciplinary cooperation in which experts in biomechanics, neurophysiology, motor functions, biology, evolutionary psychology, etc., must be involved.

CHAPTER 6

EXPERIMENTAL RESULTS AND FUTURE DIRECTIONS

6.1. INTRODUCTION

In the previous three chapters the elements of the design of the hybrid numerical/knowledge based locomotion control system for the four-legged articulated robot have been developed. These include: (1) the development of the Joint Coordination Control algorithms (JCC) for the synthesis of body and leg trajectories under animator control, (2) the development of the Numerical Controller algorithms (NC) for the generation of torque profiles for joints of the robot that are needed to produce the desired motion, and (3) the development of the Learning Controller algorithms (LC) for capturing the human-synthesized robot locomotion skills so they can be reproduced easier and faster, and to deal with the unpredictability of the environment.

To explore the feasibility of the proposed system a prototypical simulation has been developed. The purpose of the simulation experiments is that we want to see if the LC can, indeed, traverse the kinds of terrains it was intended for. We would like to place it in a simulated environment and finds out if it works. The simulation has facilities to display the motion of the four-legged robot using four different locomotion control techniques: (1) Programmed-in, (2) Kinematic (using the JCC), (3) Dynamic (using the NC), and (4) High-level skills (using the LC). In this chapter the details of the prototypical simulation are described. Then the results of the various motions' performance evaluation are analyzed. Finally, conclusions and extensions for further work are presented.

The organization of this chapter is as follows. In section 6.2, the experimental system organization is described. Then section 6.3 describes the implementation development phases. The system operation is described in section 6.4. System evaluation and conclusions are described in section 6.5. Finally, the research contributions and future work and extensions are presented in section 6.6 and 6.7 respectively.

6.2. SYSTEM ORGANIZATION

The hybrid locomotion control system has been simulated on an IRIS-2400 graphics workstation. The simulation was written in C. Listings and descriptions of the routines may be found in a separate technical report [Moh88].

The modules that make up the simulation are shown in Figure 6.1. At the core of the simulation is the robot model, a data structure describing the robot at a particular instant of time. Display routines use the robot model to formulate a display description for the IRIS workstation. Modifications to the robot model

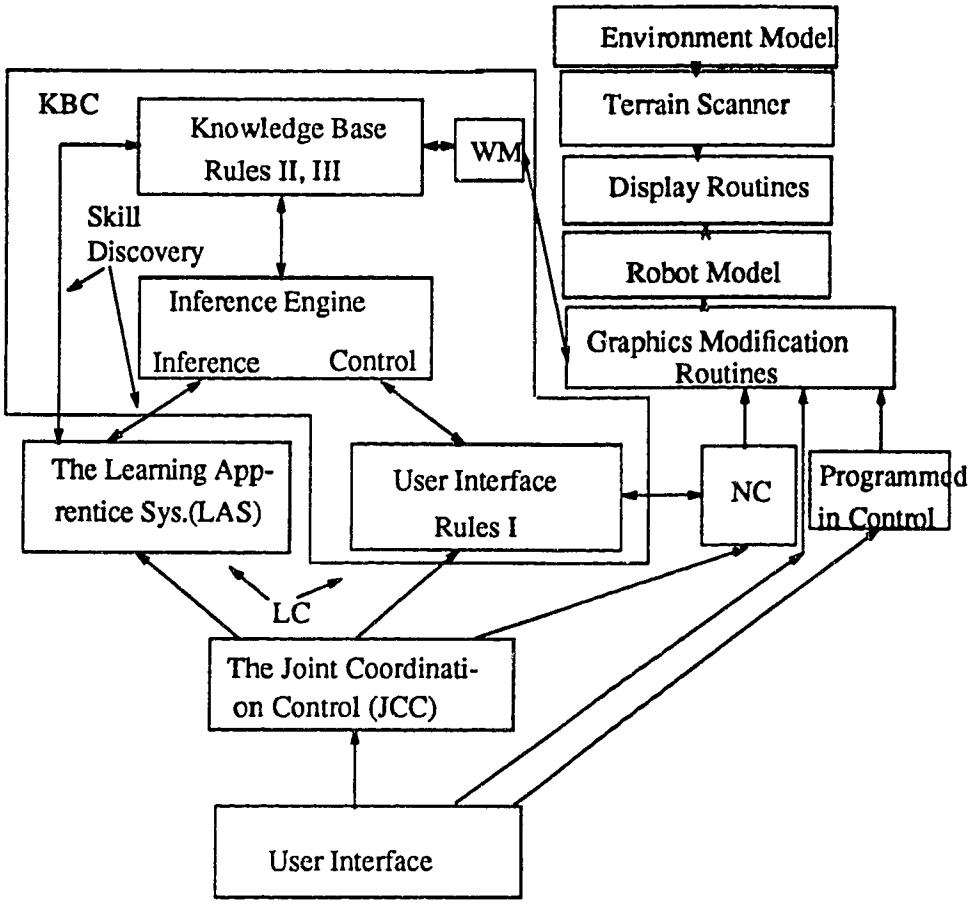

Figure 6.1 The Experimental System Modules

are automatically accompanied by a repositioning of the display figure. User-interface routines allow the user to peruse the present environment , to invoke the various motion control routines, as well as to use the graphics commands such as projection view, camera position, moving or fixed camera, zooming in/out, multi-views, etc.

Programmed-in motion routines send sequences of precisely programmed modifications directly to the modification routines. These routines implement walking on a flat surface, for example, as continuous increments of the joint angles at the appropriate walking swing and stance phases. The translation motion of the robot body is implemented to make the motion look natural. Each locomotive skill is implemented in a separate routine. These routines are not based on any motion equations, they just graphically position the robot joints at the appropriate times to animate the motion.

The kinematic routines implement the JCC algorithms of chapter 3: automatic leg positioning and automatic body height, pitch, and roll regulation. These routines send a sequence of joint positions to the modification routines which then update the robot configuration. The animator provides the leg cycles of the four legs, the locomotion phases of the motion, and the desired robot body trajectory. The JCC solves for the joint angles.

The dynamics routines implement the NC single processor algorithms of Chapter 4 (the distributed implementation is discussed in section 6.7). These routines can be driven either by the kinematic JCC routines or the LC's skill routines. These routines send torque profiles to the modification routines. As in the kinematic motion control, the animator provides the leg cycles of the four legs , the locomotion phases of their motion, and the desired body trajectory. The NC solves for the torque profiles that are needed to produce the desired motion trajectory.

The high-level skills routines implement the LC and its skill acquisition algorithms of chapter 5. These routines can either drive the NC routines or the modification routines directly. In the case of driving the NC routines, they send leg cycles of the four robot legs and the desired body trajectory to the NC. In this case the NC solves for the required torque profiles that are needed to produce such motion and sends them to the modification routines. In the case of driving the modification routines directly, the LC sends

torque profiles to the modification routines which uses them to update the robot motion.

The modification routines can receive either torque profiles or position and velocity profiles. In the first case, the modification routines use the direct dynamics algorithm of chapter 4 (based on Armstrong's equations) and solves for the robot motion. In the second case, the positions and velocities profiles are used directly to produce the robot motion.

The animator can interactively define the robot environment. Several types of environments are available for the animator. For example, Figure 6.2[1] shows an environment that consists of a flat surface

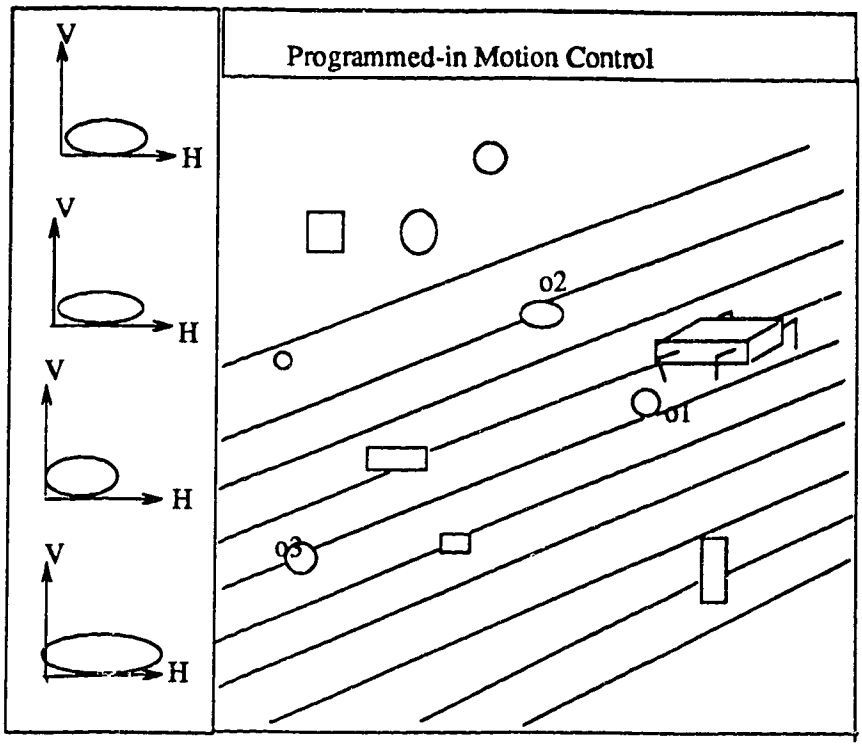

Figure 6.2 A Flat Surface Environment Beset with Obstacles

[1]In the figure V & H are the vertical and horizontal distances. The ellipses at the left side of the figure represent the trajectories of the robot feet viewed from the side (Cartesian space) expressed in the body coordinate frame (see Figure 3.5). The diagonal lines just show that the terrain is without holes (see figure 6.3).

beset with obstacles. The environment contains road blocks of two types: large obstacles that the robot can not step over and has to avoid and small obstacles that the robot can step on top of. Figure 6.3[2] shows a different type of environment that consists of flat surface beset with holes as obstacles. In this environment the robot is to avoid stepping over any of the holes. Finally, Figure 6.4 shows a terrain that is modeled as a continuous surface using cubic surfaces (Cardinal Spline, and Bezier). The surfaces could be adjusted by changing the values of the control points. By adjusting these points, the terrain could be made flat, or sloping in different regions.

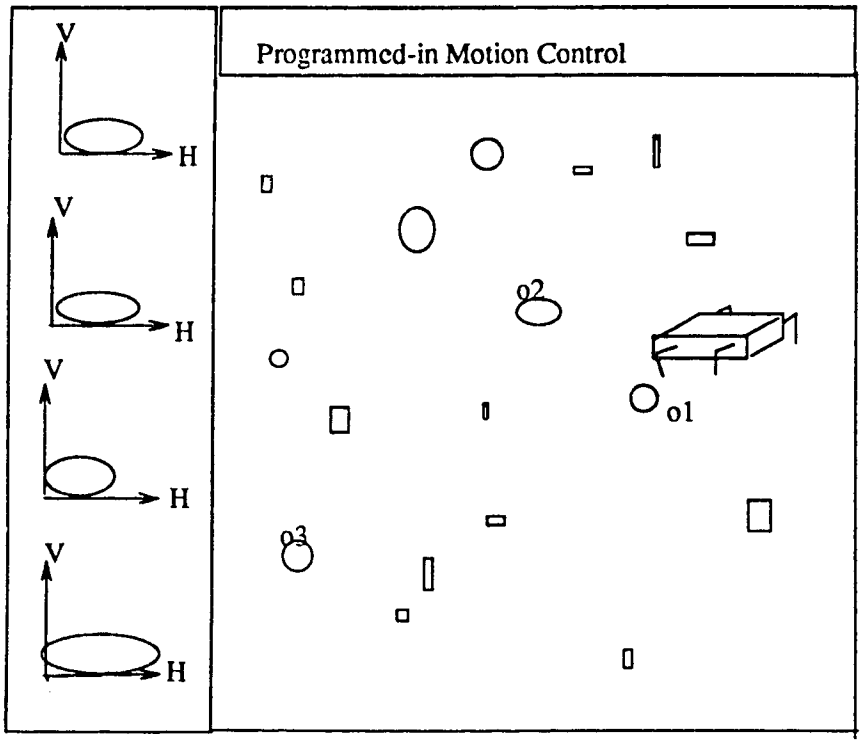

Figure 6.3 A Flat Surface Environment Beset with Holes

[2]V & H are as in Figure 6.2.

The experimental system provides three types of navigation strategies: pre-programmed, animator controlled, and robot controlled. In the case of programmed-in motion control, the navigation strategy is pre-programmed in the motion control program (e.g., walk to point A, then turn-left, then trot to point C, etc.) and it is fixed. In the case of both kinematic and dynamic control of the motion, the animator controls the navigation of the robot using the speed, heading, and gait control commands described in chapter 3.

Finally, in the case of high-level skill motion control, the navigation strategy uses an A^* search algorithm. This navigation system is a single initial state and a single goal state system. That is, it makes no difference whether a navigation mission is solved in the forward or backward direction. The approach that has been taken here is a bidirectional search algorithm similar to the one in [Ric83]. A symbolic description of the algorithm can be found in the appendix. The navigation system incrementally builds a terrain map by integrating the information about the paths traversed so far. The robot has a simulated on-board camera fixed on top of it. This camera regularly feeds to the navigation system the local obstacles that the robot faces. The initial traversals of the robot are based on a local navigation strategy that uses the on-board camera information. At any stage in the navigation, the terrain is characterized by a partially built world map. This world map is updated from time-to-time by incorporating information from the on-board camera. The robot employs a global navigation strategy that uses the learned world map in the regions it is available, and resorts to local navigation in the regions the learned map is not available.

The navigation system incrementally builds a terrain map by integrating the information about the paths traversed so far. The terrain map is actually a graph (V,E) where V is the set of vertices that represent the obstacles' edges in the environment, E is the set of edges such that $(v_i, v_j) \in E$ iff there is an unobstructed path (i.e. with no large obstacles) between vertices v_i and v_j. Initially the map is empty. The robot uses its on-board camera to measure the distance to any obstacle edge (v_k) in any specified direction. The robot's strategy is to navigate arbitrarily close to the obstacle's edges (alongside the edges). It only leaves the side of an edge if a new direction is suggested to it. In this case it traverses along this suggested direction till it reaches a point on the edge of another obstacle and then navigates alongside its edges, etc. To go from an initial point (S) to a destination point (D), the robot moves along the $S \rightarrow D$ direction till it

gets to the nearest obstacle at a point (say x) at one of its sides. The robot decides then either to circumnavigate or overcome this obstacle using a local planning strategy. The technique is then recursively applied to reach D from the intermediate point. When the robot reaches the point x, it then has one of two choices:

(1)If the faced obstacle is a small obstacle then the robot will try to overcome it by stepping on top of it, adjusting its body orientation, and maintaining its balance without changing its heading direction.

(2)If the obstacle faced is a large one or a big hole, then the robot will select one vertex on the edge of this obstacle on which the point x is located, and go towards it in order to circumnavigate the obstacle's corner.

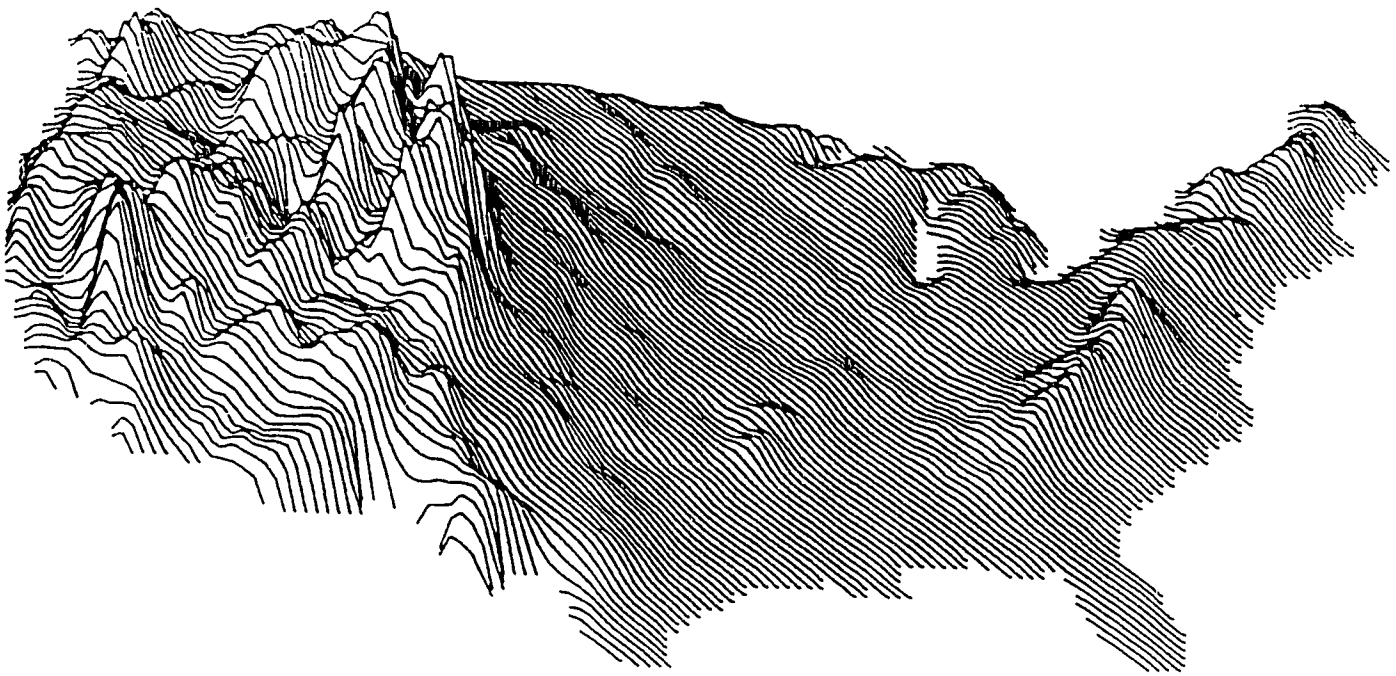

Figure 6.4 A Continuous Surface Built using Parametric Cubic Surfaces

Whenever the robot reaches a new vertex v_i, this vertex (an obstacle corner) is added to the current map. From this vertex, the robot uses its on-board camera in the direction of all the *existing* vertices of the current map. The edge (v_i, v_k) is added to the current map for each vertex v_k in the current map that is visible from v_i. This navigation strategy always yield a path if one exists (a proof could be found in [MoA88a]). This strategy is used to allow the robot to navigate an unexplored terrain. Note that the navigation paths are not necessarily globally optimal. However, the extra work carried out in the form of learning the map is inevitable because of the lack of information about these obstacles. In the regions where the navigation map is available, the optimal path can be found by computing the shortest path from the source point to the destination point on the map using an A^* search algorithm (see the appendix).

What we have, then, is not only a navigation system which operates in a dynamic microworld, but also a navigation system which can in some sense learn from its past behavior. This gives the robot the high agility that might be required to quickly move from one point to another on a structure under construction or needing repairs in an emergency. The strategy to test the navigation system has been to invent a particular microworld, pre-store several routes through the map, then iteratively produce plans, execute them and preview the updated map (see Section 6.5).

A module called the "terrain scanner" (see Figure 6.1) is used to provide terrain preview data which are used by the automatic leg positioning and body regulation algorithms to predict foothold locations and to determine average slope and elevation of the terrain for use in adjusting body attitude and altitude. Furthermore, this module maintains the following information:

(1) the location of each supporting foot;

(2) the polygon whose vertices consist of the vertical projection of the supporting feet;

(3) the predicted polygon of (2) that would result if a foot being in a swing phase were immediately lowered;

(4) the vertical projection of the center of gravity;

(5) the reachable area of each foot;

(6) any critical feet (A critical foot is defined as a foot which, if lifted, would cause the body to be statically unstable);

(7) the pitch and roll attitude of the robot body.

6.3. SYSTEM IMPLEMENTATION DEVELOPMENT

The development of the simulation went through various stages:

(1) The display routines were first tested using a programmed-in motion control. Various programmed-in locomotion skills on flat surface were tried (see figure 6.2 and 6.3). The robot navigation strategy was programmed-in, that is, that robot walked from position O_1 (x_1,y_1,z_1) to position O_2 (x_2,y_2,z_2) then turned left, then trotted to position O_3 (x_3,y_3,z_3), and so on. The produced motion was far from realistic, although smooth transitions between locomotive skills were achieved. Under this mode of motion control, it was easy to maintain the robot orientation and balance (artificially of course). Control over the robot's velocity and ability to avoid obstacles was also simple to achieve. On the other hand, under this mode of motion control, the robot had no ability to deal intelligently with the environment. Putting some obstacles in the way of the robot without modifying the motion programs, the robot went through the obstacles as if they were not there. In all current experiments the animator was able to change the graphical projection view, set the viewing camera, zoom-in and out, make the camera move on top of the traveling robot, and have several windows that displayed the traveling robot from different points of views. Projection transformation define the mapping from the eye coordinate system to the screen. The viewing transformation place the viewer and the eye coordinate system in the world space. A viewport, which specify a screen area to display the projected image, is associated with each projection transformation. The IRIS workstation has a window manager system (called "mex"). With it, one can create several independent graphics displays, or windows on the screen.

(2) The JCC algorithms were then tested. In Figures similar to 6.2 & 6.3 (with the top area of the screen indicating that the robot is "kinematically controlled"), the robot traveled under the animator control over a flat surface. The animator input the robot leg cycles, their locomotion phases, and the desired body trajectory. The JCC algorithms computed the robot joint angles and their rates. Table 6.1[3] shows a typical result, of the JCC algorithms, for a portion of a walking cycle. The stride length was set at 3 metres while the robot was moving $\frac{1}{6}$ metres/sec in a straight line. The size of the robot is $5.0 \times 2.0 \times 0.5$ metres- which is $2a \times 2b \times 2c$ (where a,b,c are the Coordinates of the hip socket in the body fixed coordinates). l_1=2.0 metres, l_2=3.0 metres. Under this mode of motion control, it was more difficult to maintain the robot's smooth transitions between different locomotive skills. The animator had control over the robot orientation, velocity, and the locomotive skill that is used at any time during the terrain navigation. Although the motion produced looked more realistic than the programmed-in motion control case, the robot had no way to intelligently respond to any environment changes.

As an example of a relatively difficult JCC control problem, in one simulated environment the robot was made to navigate a stairway. The stairs rose at a 45 degree angle to the horizontal, so that the robot could not climb them without tilting its body. This experiment tested the automatic pitch regulation of the JCC. The robot legs were extended and the position of any abnormally extending leg was monitored to insure it is never extended beyond the kinematic limits. With the automatic body regulation algorithm, if this extending leg reaches the kinematic limits, then the robot body automatically reacts to move in such a direction as to accommodate the desired motion of that individual leg. Of course, the positions of all of the support legs were monitored during this accommodation movement to insure that the body movement does not extend a support leg beyond its kinematic limits. Also, the robot stability was monitored to insure that the body is not shifted to a

[3]In the table entries; (x pos., y pos., z pos) are the components of the position vector of the center of gravity of the robot body relative to the inertial frame. $\psi[legi]$,$\theta_h[legi]$,$\theta_k[legi]$ are leg i joint angles. $\psi[legi]$ is the azimuth angle. It is the rotation angle around the Z_{hi}-axis. $\theta_h[legi]$ is the hip elevation angle. It is the rotation angle around the Y'_i-axis. $\theta_k[legi]$ is the knee elevation angle. It is the rotation angle around the Y"-axis. The four lines under the joint angle columns represent the values for the four legs 1 through 4 respectively.

position where the robot is unstable.

(3) The NC algorithms were then tested. These experiments also involved the JCC and the direct dynamic routines. The robot moved, under the operator's control, over a flat surface using the NC algorithms. Similar to (2) the animator input the robot leg cycles, their locomotion phases and the desired body trajectory. The NC algorithms computed the robot joint torques required to produce this desired motion. Table 6.2[4] shows a typical result of the NC algorithms for a portion of a trotting cycle. The stride length was set at 3 metres while the robot is moving $\frac{1}{8}$ metres/sec in a straight line. Figures 6.5 and 6.6 give graphs for the 12 joint torques for the complete walking and trotting cycles (both are using the same speed $\frac{1}{8} metres/sec$).

In implementing the solution of the dynamics equations of motion two types of checks were built in the NC code. First the difference between the recursive and the closed form equations of the direct dynamics calculations were monitored (the computed values of τ_k, $J^r \dot{\omega}^r$, and f^r should agree). The

Table 6.1 A Typical Result of the JCC Calculations for a Walking Skill

x pos.	y pos.	z pos.	$\theta_k[legi]$	$\theta_h[legi]$	$\psi[legi]$
-35.000000	0.000000	-2.500000	4.763295	0.382518	-0.334318
			1.578295	0.382518	-0.334318
			4.763295	0.382518	-0.334318
			1.558295	0.382518	-0.334318
-34.984886	0.000000	-2.500000	4.697274	0.382499	-0.334338
			1.585907	0.382499	-0.334338
			4.697274	0.382499	-0.334338
			1.585907	0.382499	-0.334338
-34.969662	0.000000	-2.500000	4.689609	0.382476	-0.345732
			1.593572	0.382476	-0.334318
			4.689609	0.382476	-0.334318
			1.593572	0.382476	-0.334318

[4]In the table entries (Trq_x, Trq_y, Trq_z) is the net torque acting on the robot body, expressed in the inertial frame. $\tau_k[legi]$,$\tau_\theta[legi]$,$\tau_\psi[legi]$ are the joint actuator torques in leg i. The four lines under the joint torque columns represent the values for the four legs 1, 4, 2, and 3 repectively.

second check is the "closure" (i.e. the produced trajectory should agree with the desired kinematic trajectory).

The way we supported the robot (made it stand up in its natural configuration without collapsing) is by attaching ground reaction forces to each of its four feet.[5] The amount of these forces were one fourth of the robot weight and directed upwards in the $-Z_I$ direction. To represent the kinematic limits of the joints, restorative torques at the joints (see Figure 6.7) were exerted at all times. This has decreased the tendency of the limbs to oscillate and keeps the robot from collapsing. These restorative torques maintained the joint angles at their natural orientations. The computation time for the NC varied with the present state of the robot. Much of the NC's computation time was involved in solving the linear programming problem with a computation time typically varying between 0.5 and 1.5 seconds per complete dynamics cycle (not locomotion cycle). Usually 15 to 30 iterations within the linear programming routine itself were needed to find the optimal solution. Most of these iterations involved finding a feasible solution (eliminating the artificial variables) for the joint torques.

Table 6.2 A Typical Result of the NC Calculations for a Trotting Skill

Trq_x	Trq_y	Trq_z	$\tau_\theta[leg\ i]$	$\tau_k[leg\ i]$	$\tau_\psi[leg\ i]$
0.000000	0.033646	0.000005	80.869167	6.271701	8.713732
			87.869370	4.271897	-8.713579
			91.098686	17.861366	0.064282
			91.098495	17.861319	-0.064332
-0.000002	0.037382	0.000005	82.699998	5.029613	9.655461
			82.698509	5.029601	-9.655068
			90.629852	17.862061	0.063670
			90.631615	17.862549	-0.061985
-0.000004	0.041225	0.000005	100.000000	3.771859	10.612420
			71.657433	10.251559	-15.761279
			71.667000	17.855803	0.000000
			100.000007	11.368942	-4.924666

[5]These reaction forces are not applied when the robot starts to move.

Under this mode of motion control, the motion produced looked more realistic than the previous two techniques, but still the robot had no ability to deal intelligently with any changes in its environment. A number of points may be noted from analyzing the curves of figures 6.5 and 6.6.

The figures summarize the flexor/extensor patterns during the stance and swing phases of the walking and trotting strides. The patterns are approximated from data that display the curve values every $\frac{1}{8}$ sec. The curves are computed during a constant speed on flat ground of $\frac{1}{8}$metres/sec, β=0.5 in the case of trotting and 0.8 in the case of walking. Stride length was 3.0 metres in both cases. The knee angles fluctuated around the 90°. The hip azimuth angle fluctuated around the 0°, and the hip elevation angle fluctuated around the 10°. For walking, a phase difference of approximately one-quarter of a cycle between successively moving legs was maintained. That means that each leg is always about one-half of a complete cycle out of phase with its fellow leg on the other side of the body. Each leg swings about one fourth of the walking cycle, and remains on the ground the rest of the cycle. The swing and stance phases are shown on the curves. During swinging the hip elevation torque alternates between flexion and extension, and mainly remains in flexion during the stance phase to keep the foot on the ground. The mechanism for swinging is as follows: (1) the τ_θ changes from flexion to extension to make the legs' feet leave the ground; (2) during this time the τ_k extends the knee and the τ_ψ changes from flexion to extension to drive the hip forward. The mechanism for the stance phase is as follows: (1) the τ_θ remains in flexion all the time to force the feet on the ground; (2) τ_k and τ_ψ alternate between flexion and extension to facilitate pushing the robot body. In the walking curves the sequence of leg lifting is 3-2-4-1 in that order. Torque is in Newton.metres and time is in seconds. The curves were computed in the middle of the robot motion (not starting from rest conditions).

The characteristic feature of a trot is a progressive reduction in the phase difference between a fore leg and the contralateral hind leg and a corresponding increase between a fore leg and the ipsilateral hind leg. In a fully synchronized trot a fore leg and the contralateral hind leg are completely in phase, and the body is supported on the two diagonal pairs of limbs alternately. In other words, during trotting there is always one-half cycle difference in phase between any leg and its contralateral fellow. Each leg is on the

ground for $\frac{1}{2}$ of a complete cycle. Phase differences between diagonal legs (2,3) and (1,4) is nil. Phase difference between ipsilateral legs is $\frac{1}{2}$ cycle. In the trot figure, legs 1, 4 torque profiles are almost identical images of each other. So are the torque profiles for legs 2, 3. The hip elevation torque reverses (between flexion and extension) only during the swing phases, and remains flexion during most of the stance phases. The mechanism for swinging is as follows: (1)the τ_θ changes from flexion to extension to force the foot to leave the ground; (2)during this time, the τ_k extends the knee and the τ_ψ changes from extension to flexion to drive the hip forward. The mechanism for the stance is as follows: (1) the τ_θ always remains flexion to force the foot not to leave the ground; (2)τ_k and τ_ψ alternate between extension and flexion to allow for pushing the body.

(4) Finally, the LC algorithms were tested. Figure 6.8 shows the robot traveling under the LC control. Under this mode of motion control, the robot was able to navigate the environment intelligently in real-time and to produce quality motion similar to the ones produced in (3). All the sustained stable locomotion requirements of the robot's motion were achieved (see next section). In the experiments of (3) the robot was not controlled in real time, since the total loop computation time for the NC algorithms was approximately 1.5 seconds. This was two orders of magnitude slower than real time control. Since the direct dynamics module of the modification routines was able to process the motion torque profiles in real time, the LC controlled the robot's motion in real time.[6]

The teacher uses the lower screen area of figure 6.8[7] for editing the joint torque profiles to avoid obstacles. For example, in order to extend the reach of the robot leg in order to avoid a hole, the magnitude of the knee extensor torque profile was increased in some parts of the cycle. The joint torques are illustrated in figures 6.5 and 6.6. Given the form of the produced motion it was

[6]Apparently there is no constraint that the learning process itself be performed in real-time.

[7]The V and H ellipses in the figure are the same as in figures 6.2 and 6.3.

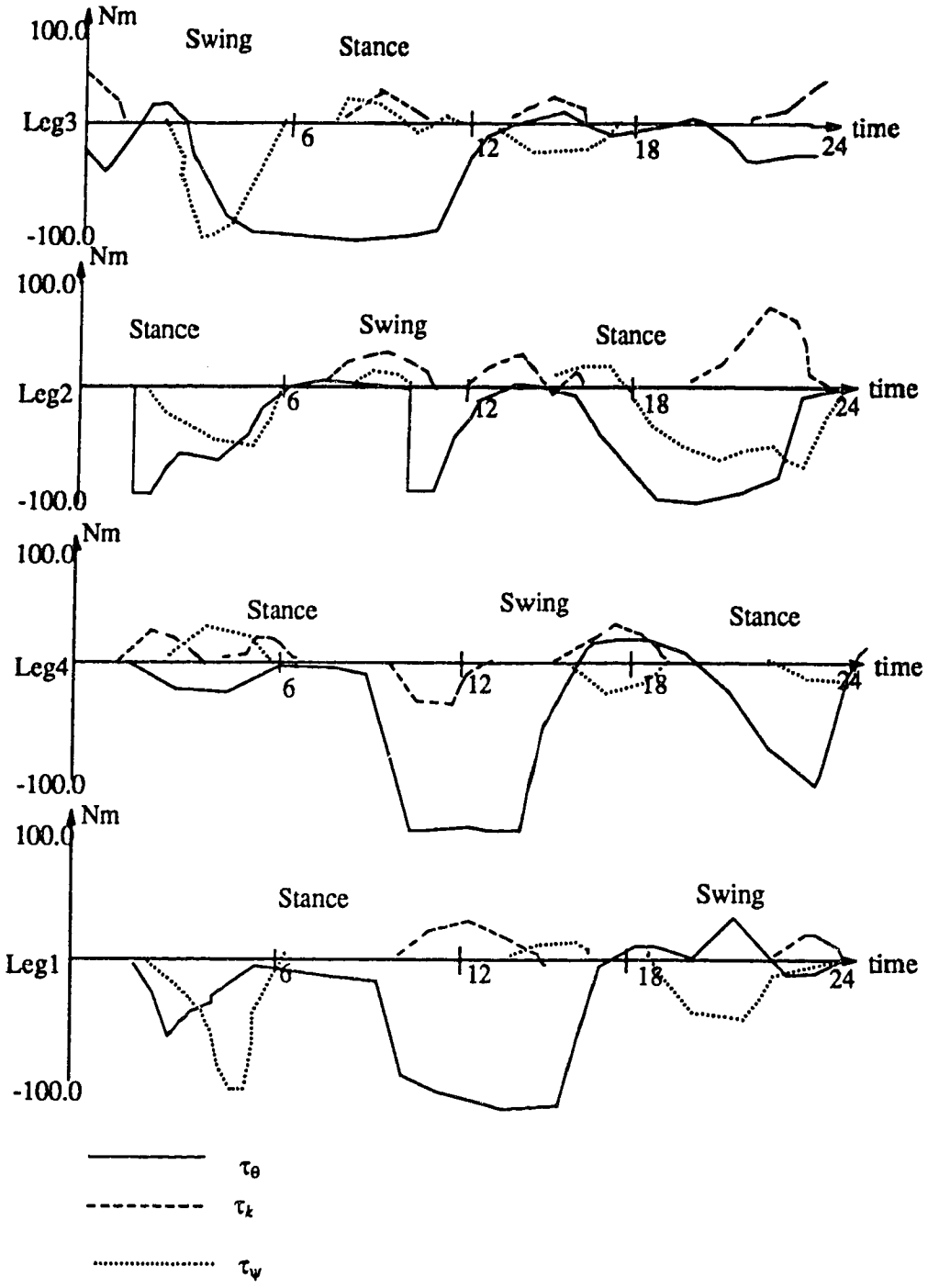

Figure 6.5 Graphs for the 12 joint torques for a complete walking cycle

recognized that modifications to the hip torque were necessary. Modifying[8] the hip torque profile

[8]This modification is done by trial and error adjustment.

coupled with the previous changes to the knee torque produced the required extended motion. As mentioned before, the idea of editing the torque profiles directly comes from the work of Hollerbach [Hol78], Asaytryan [SZH79], Platt and Badler [PlB81], and Marshall [MJW85].[9] As it is found difficult to edit the torque profiles directly to correct the motion, a more convenient method of modifying the leg cycles and the joint angles is implemented in the system. The user interface provides an interactive graphical editor which allows the teacher to edit control curves. In order to edit a joint movement, the teacher selects the robot's joint (using the mouse); the curves (angles vs time and torques vs time) for all the degrees of freedom at this joint are displayed for editing. Using point modifications, the curves can be modified.

6.4. SYSTEM OPERATION

The experimental system presented the animator with four screens. In one the programmed-in motion control is in charge of the control of the actions of the robot. In the second the JCC is in control of the actions of the robot. In the third the NC is in charge of the control of the actions of the robot. And finally, the LC is in charge of the control of the actions of the robot. The screen of figure 6.8 is the most general. It consists of five areas: the area showing the agent in control of the robot actions, the command area, the locomotive skills area, the small obstacle area, and the terrain area. The command area is used to define a mission for the robot; the animator can abort the robot's motion at any time by moving the cursor to the command area where he or she can order the robot to go faster or slower, change the displayed views, request the display of more than one view (several windows), or zoom in/out, etc.

The human teacher has control of the motion control algorithms' alternations of control over the robot's actions in order to produce sustained stable locomotion all the time and in any environment. Sustained stable locomotion requirements were identified in section 3.2, and they include:

---

[9] See Section 3.2.

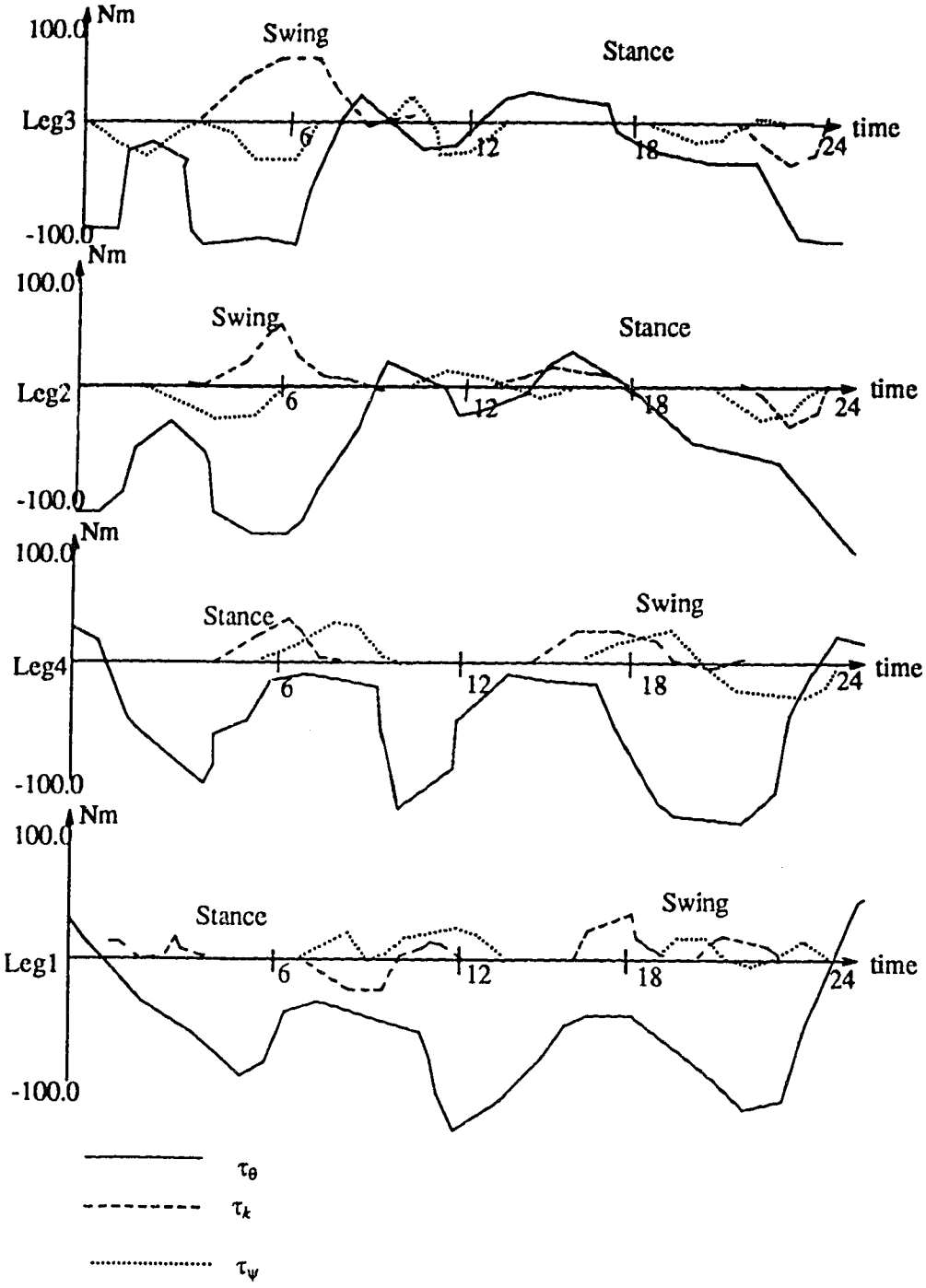

Figure 6.6 Graphs for the 12 joint torques for a complete trotting cycle

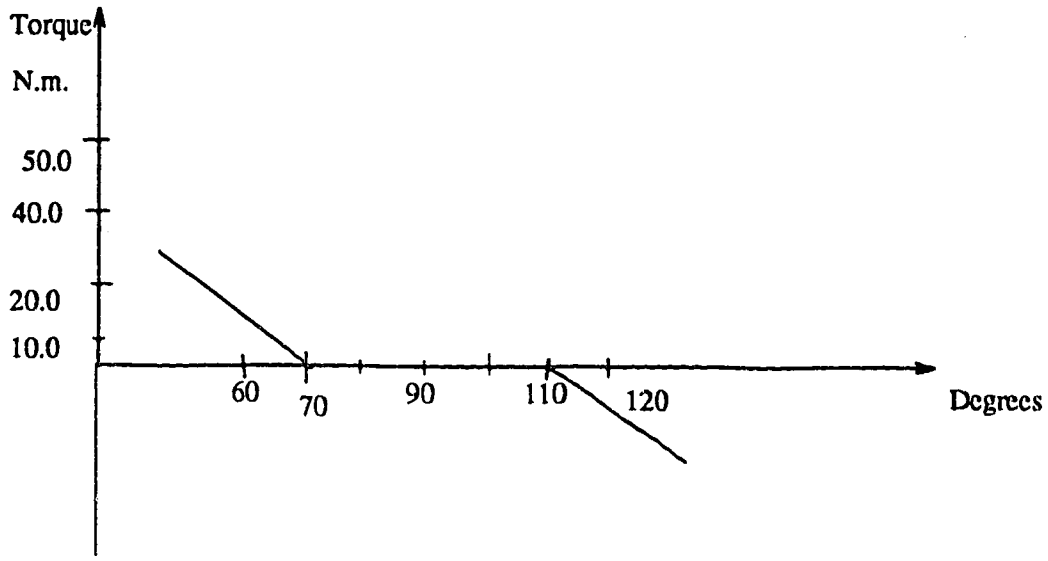

Figure 6.7 Knee Joint Restorative Torque

(1) maintaining the robot orientation, having control over its velocity, and avoiding obstacles,

(2) choosing the most appropriate locomotive skill at any point during the robot's navigation (e.g, walk, trot, climb, etc.),

(3) dealing intelligently with the environment,

(4) maintaining the robot's static and dynamic stability,

(5) ability to combine different locomotion skills (e.g, turning while running),

(6) achieving smooth transitions between different locomotive skills,

(7) preferring the paths the robot has traveled on before,

(8) reducing total energy consumption in executing the robot's missions.

When the LC is in charge of the robot actions, it uses the three types of rules that are created by the learning apprentice system (LAS). The current active rules that are producing the current robot motion are

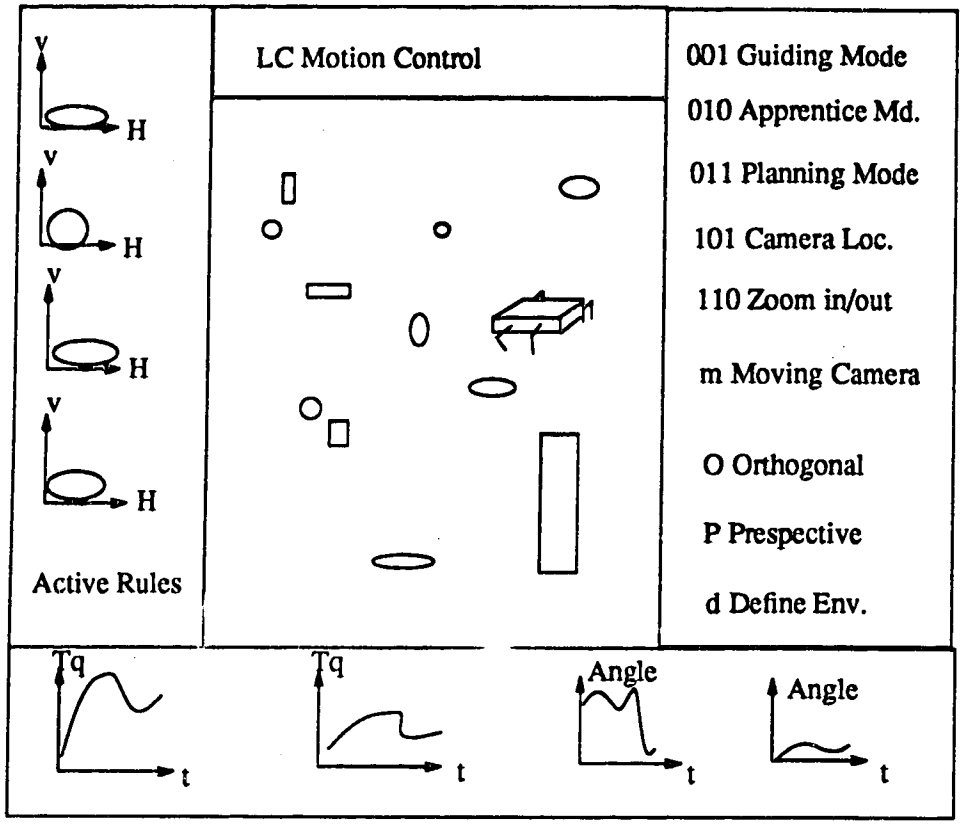

Figure 6.8 The Learning Controller Controlling the Robot Motion

displayed in both the locomotive skills and small obstacles areas of the screen. These areas are always reflecting the current active rules (see Figure 6.8).

The system of Figure 6.8 has three modes of operation:

(1) Dynamics guiding mode: In this mode the teacher "composes" new movements interactively using the JCC and NC. Through trial and error (use "Guide Mode" then "Playback" from the command area), he or she must first determine that the motion is satisfying the sustained stable locomotion requirements before deciding to let the LC learn it.

(2) Apprentice mode: Here the user plays back the motion to be learned by the LC. The learning apprentice system observes the actions and creates/modifies production rules in the appropriate rule

set. In other words, the system in this mode acquires rules and builds/updates its knowledge base.

(3) Planning mode: Here the LC produces plans for navigating the environment in order to reach a given destination. The inference engine uses the three rule sets to execute such plans and tailor a motion for the currently perceived obstacles.

It is important to note that there is no single procedure to be followed by the human teacher (animator) to get the robot to navigate its environment and produce locomotion according to the motion requirements all the time. The idea of human-machine mix (LAS's idea of "information rich external environment") that is present in the proposed system was possible because of the partitioning of any mission task that is performed by the robot into a number of intermediate decisions, between which the human teacher can alter the LC decisions. The learning apprentice system has to take advantage of this teacher's feedback to improve the LC's knowledge base and therefore performance in subsequent missions.

The teacher will detect "motion bugs" for the simulated robot in two forms: illegal operations, and failed steps. An illegal operation is one that is considered impossible in the current environment. For instance, it is illegal to walk through an obstacle or for the leg to go underground. A failed step is one that does not achieve its goal for the designated time interval. The teacher verifies that at all times the goals intended by the planner have actually been met. These methods of detecting "motion bugs" provide a performance standard for the robot, which states that a plan must execute legally, achieve all intended goals and subgoals, and also be purposefully correct. The teacher will halt the motion of the simulated robot whenever one of these problems is identified by moving the cursor to the appropriate screen area (when the cursor is in any area other than the terrain area, the robot motion is interrupted). Notice that the source of trouble will be shown on the appropriate screen area which is continuously changing to reflect what is going on underneath the animation subsystem. A trace of requested torque profiles and joint angles will be available to the teacher upon request to allow the torques to be fixed interactively. When the teacher intervenes, he or she moves the cursor to either edit one of the local terrain motion enhancement rules or to define (or modify) a prototypical torque profile rule for a particular locomotive skill rule. The rules that are fired at this point by the LC fired incorrectly, so the associated contexts are negative training instances for

them. Furthermore, the newly defined or modified rules take these same contexts as positive training instances. Negative training instances are used for rule discrimination (see algorithm C in the previous chapter), whereas positive training instances are used for rule generalization (see algorithm B in the previous chapter).

6.5. SYSTEM EVALUATION AND CONCLUSIONS

In this section we first evaluate three specific capabilities of the control system; namely its obstacle handling capability, its capability of learning the environment map and using it, and finally its capability to generalize and discriminate. Then a methodology for evaluating the overall system is described, followed by a methodology for measuring the learning progress in the system.

In order to analyze the capability of the robot to deal with obstacles, the feet positions (in Cartesian coordinates) and the data from the terrain scanner module were stored for an off-line analysis. Tests were performed for several different combinations of simulated terrain. The data stored from the test was used for generating the terrain map (off-line). Based on the foot position and actual foot height, the level of the surface in Cartesian coordinates was computed. By connecting a series of surface points, a 2D terrain map was generated on the graphics screen. Figures 6.9 and 6.10 show the trajectory of the front left foot while the leg adapted to the terrains of Figures 6.2, 6.4 during straight line walking and climbing strides. The terrain curve was generated off-line based on the information from the terrain scanner module. The other curves show the trajectory of the feet.

In Section 6.2 a navigation system that learns from its past behavior was described. A methodology to test its ability to learn has been to invent a particular microworld, pre-store several routes through the map, then iteratively produce plans, execute them and preview the updated map. Figure 6.11 shows how the iterative execution of navigation plans updates the robot's map. In the figure, initially the terrain is unexplored and the map is empty. A sequence of four paths is undertaken in succession by the robot. The figure illustrate the various paths traversed and the corresponding maps (the terrain is shown on the left side

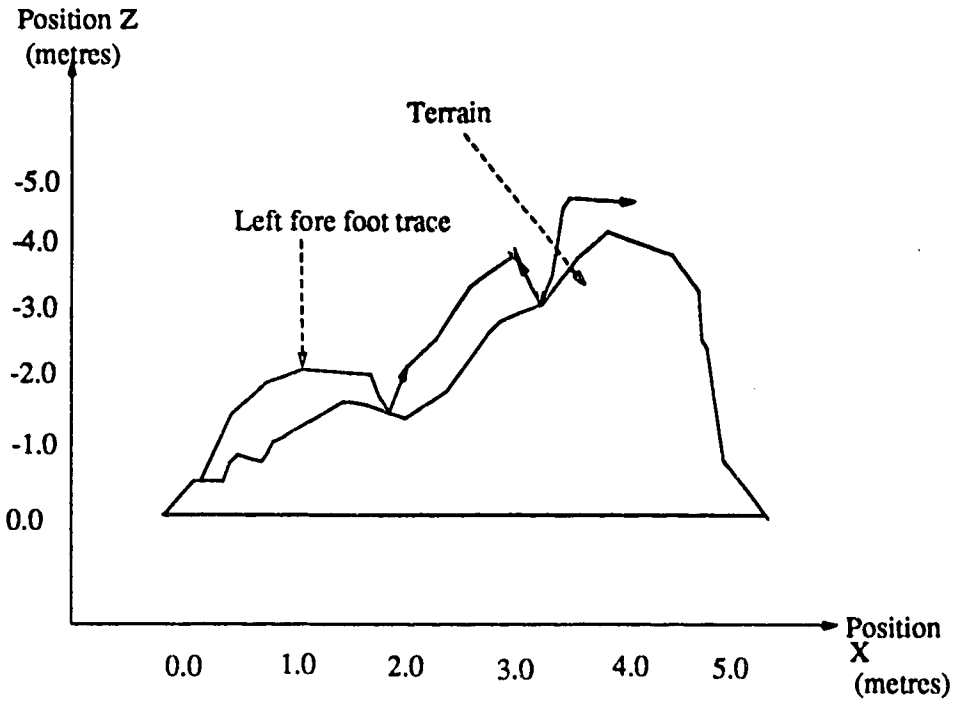

Figure 6.9 Obstacle Avoidance- Off-line Analysis

of the figure whereas the map is shown on the right side). Initially, during the motion from 1 to 2 the robot learnt four edges. In the next traversal, seven more edges of the terrain are learnt, and so on. Consequently, the terrain is guaranteed to become completely learnt, when the complete map of the entire obstacle terrain is built. After this stage the robot traverses along the optimal paths.

In order to test the LC's capability to generalize and discriminate, an explanation based subsystem that interacts with the teacher to display the status of various rule shells is implemented. At any time the user can request the system to display the current status of any rule shell, its situation and action spaces with both lower and upper mark locations. The example provided in Section 5.4.2.1 was generated using this facility.

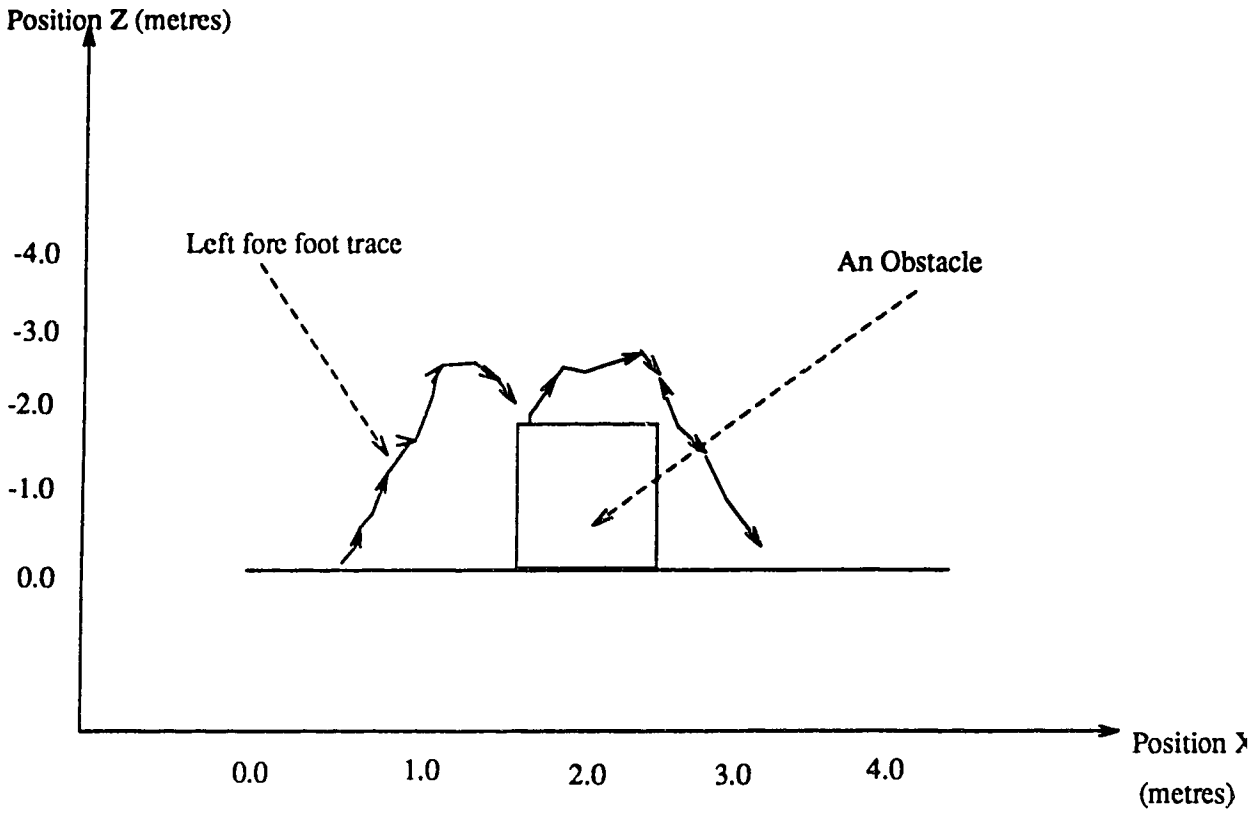

Figure 6.10 Stepping ontop of Obstacles- Off-line Analysis

In order to evaluate the overall system capabilities, a methodology for doing that is described in the next few paragraphs. As of now, the LC has proved that it can, indeed, traverse the kinds of terrains it was intended for. The important questions that we had to answer were: "How can one judge if we have succeeded or not?", "How can one know if the robot control system has evolved into a control system that is *intelligent?*" In other words, how can one measure our progress?

In 1950, Alan Turing [Tur63] proposed what is known as the "Turing test" to answer a similar question in the context of machine intelligence. In this section we propose a similar test to answer the previous questions. To perform the test, an interrogator is to assign tasks to two versions of the locomotion control system. The first version is exactly similar to what has been described in this dissertation. The other version does not have the learning controller (LC) component. The interrogator does not know which of

them is which. The interrogator knows them only as A and B, and aims to determine which of them is (NC+LC) and which is NC. If the interrogator succeeds to distinguish them on the basis of agility in dealing with the dynamics of the environment (either in producing sensible motion or in navigation), then one can conclude that the (NC+LC) has succeeded to acquire such agility through learning. If the interrogator succeeds to distinguish them on the basis of natural movements appearance, then one can conclude that the (NC+LC) has succeeded to enable the robot to solve in real-time complex problem of interaction with the environment. If the interrogator succeeds to distinguish them on the basis of response times for reasoning and decision making, then one can conclude that the internal motion representation of the LC is better manifestation of motion knowledge[10] with respect to processing time and performance requirements.

Actually the methodology to perform these tests is presently immature since we have not figured out a way to keep the teacher's intervention in the (NC+LC) version hidden from the interrogator. Meanwhile, since the Turing test has yet to be passed by AI systems in any of the human intelligence aspects after close to thirty years of research, one must not underestimate the important epistemological and methodological goals that we have to satisfy in order to build a satisfactory knowledge-based locomotion control system.

Another test is suggested in [MoA88b] to measure learning progress in autonomous robots in general as opposed to teleoperated robots. To conduct the proposed test two versions of the locomotion control system are proposed: a learning apprentice version (LAV) and an autonomous intelligent version (AIV). The learning apprentice version is similar to the system described in this dissertation. Its knowledge-based

[10]Although knowledge representation and processing are "hot" topics in AI research, very few attempts have been made to help understand motion, which means to build up a model of "motion knowledge". To make this point about the need for motion knowledge clear, one should compare it to music which is a complex phenomenon similar to motion from many viewpoints. The situation is quite different for music. Music notations were successfully able to discriminate which aspects of music to represent explicitly and which to represent implicitly. The notation was able to capture the essential structure of music (what in AI terms could be called "music knowldge"). It is easy to notice the abstractness and functionality of the notation. For example, the instruments are not shown in the notation, nor the way in which a performer plays a specific instrument, and the individual style of performance. This adequacy of the music notation symbolism to express the essential structure of music is clearly demonstrated by its ability to survive this computer era. Music notation were easily expressed in computer terms and were directly used to drive computer music synthesizers. For dance and movements, on the contrary, the picture is quite different. Dance and movement notation (such as Labanotation, effort-shape, etc.) did not find the same success as music notation, and computer techniques directly applied to them do not seem to pass the basic test, which is: the ability to generate from the representation a fluent, natural and convincing motion performance.

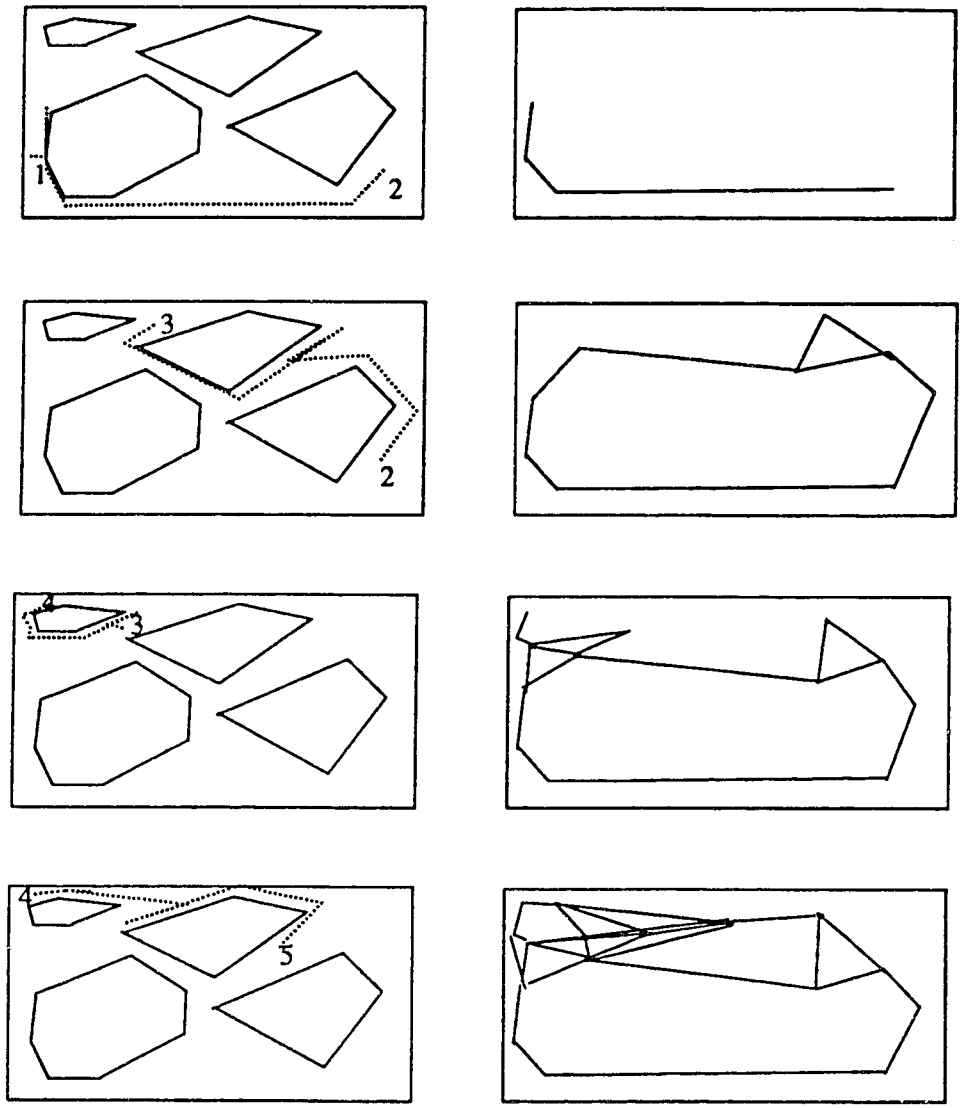

Figure 6.11 Iteratively executing navigation planes and previewing the updated map

system directly confronts the problem of knowledge acquisition by learning from interactions with an information-rich external environment (a teacher).

On the other hand, the autonomous intelligent version (AIV) replaces the LAV's teacher by experimentations. It starts with an initial set of rules (its knowledge-base) which support the search for a

solution to any navigation task. These initial rules are correct in that they propose only legal movements,[11] but they lack conditions for distinguishing good movements from bad ones. As a result, AIV makes many poor movements on its first pass (to execute an assigned navigation mission), and is forced to back up whenever it reaches a dead end. However, AIV does eventually solve the assigned navigation task in a trial-and-error way. At this point, it attempts a second pass, but this time it has the initial solution to guide its search. When one of its rules is incorrectly applied, AIV knows its mistake immediately. It tries to find a difference between the two situations and modify its rule set. AIV continues to learn in this fashion, constructing more conservative rules when errors are made, stengthening rules when they are relearned and generalizing rules when they are successfully applied under new conditions, until it traverses the entire solution path with no mistakes.

As such the AIV belongs to a class of learning systems that are called "non-supervised learning systems" [And83]. Accordingly, AIV should be responsible for deciding who should take control of the actions of the robot (the NC or the LC) at any time instant (it automates the NC-LC alternation of control). It should also be responsible for generating and experimenting with different leg cycles and measuring the motion quality based on an evaluation function (see later for the measure of desirability). One possible suggestion is that the AIV can use methods described in [MoA88b], to construct torque patterns from combinations of old ones.

Actually both LAV and AIV start with the same set of rules (the knowledge base). The difference between them is that LAV relies on a learning methodology called "explanation-based learning" [Mit83] [DeM86] [Seg87] that requires interactions with an external teacher. For that reason, we characterize it as a pseudo-teleoperated learner system. On the other hand the AIV relies on a "procedural learning approach" [Nev78] [Anz78] that is based on learning by making mistakes.

We propose a learning progress measure test for measuring the learning progress in autonomous robots. The test is to answer the following question: "Does the autonomous learning mechanism work as well as the teleoperated learning mechanism?" To perform the test, an interrogator is to assign tasks to

[11]Movements that satisfy the sustained stable locomotion requirements.

either LAV or AIV, but does not know which of them is which. The interrogator knows them only as A and B, and aims to determine which of them is LAV and which is AIV. The goal of AIV is to fool the interrogator into believing it to be the LAV. If the AIV succeeds at this, then one can conclude that we have succeeded in implementing intelligence in the AIV. Of course both the experimentations of the AIV and the teacher's intervention in LAV are hidden from the interrogator.

The interrogator judges the two intelligent locomotion control versions by measuring the desirability of the robot's actions (i.e. performing more adequately a list of motion requirements). The interrogator uses the following list of requirements: (1) produce sustained stable locomotion, i.e. maintain the robot's orientation, have control over its velocity, and avoid obstacles, (2) choose the most appropriate locomotive skill at any point during navigation (e.g. walk, trot, climb, etc.), (3) deal intelligently with the environment, (4) maintain the robot's static and dynamic stability, (5) possess ability to combine different locomotion skills (e.g. turning while running), (6) execute smooth transitions between different locomotive skills, (7) prefer the paths the robot has traveled on before, (8) reduce total energy consumption in executing the robot's missions.

A measure for the "desirability of a situation" is formed as a heuristic weighted function of these performance measures:

$$D(t_n - t_{n+m}) = \sum_{i=n}^{n+m} \sum_{j=1}^{8} a_{ij} f_j$$

where f_j is a performance measure (one of the eight mentioned above) and a_{ij} is a weighting coefficient. The summation over m control steps tends to enhance the evaluation accuracy. $D(t_n - t_{n+m})$ is the desirability of a situation from time step t_n to time step t_{n+m}. This takes a value between 0 and 1.

Interestingly, our concepts of LAV and AIV find evidential support in behavioral psychology research: motor skills development in children takes place over months parallel with their ability to control their limbs properly. During such development, errors are adjusted iteratively in the next instances of the movements either by the child itself (an AIV resemblance) or by the help of its parents (an LAV resemblance). Through the ability to remember a previous experience of solving a specific or analogous problem, the child's motor system has the ability to learn and improve its performance. As such, we expect

to have both similarities and differences in the performance of the LAV and AIV. We do not yet, however, know whether the similarities are going to be generalizable, or artificially produced.

In conclusion, the simulated system experiments showed how the robot acquires locomotion rules for the control of its dynamic system through cognition: learning the connection between perception and action from experience. Since the robot was able to realize new desired motions by using the knowledge of motion acquired by learning, one can conclude that this model of locomotion control makes the robot become more capable of dealing with its environment. Actually it might be safe to conclude that the proposed model proceeds along lines in accordance with the *modus operandi* of the neuromusculoskeletal system [SZH79], viz; the central nervous system generates a motor programme based upon sensory inputs and motor "memories", and motor outputs are directed to the muscles. The muscles then provide the joint torques which in turn provide the segmental angular velocities and displacements. In response to changes in the control and feedback patterns, the joint torque-time functions are altered and segment movements proceed accordingly. The proposed model incorporates neuromuscular control, in that it provide the LC with the opportunity to experiment with joint torque modifications to produce different movement outcomes in much the same way as the central nervous system adapts to changing environmental and task demands.

At present, problems involving producing oscillations and inappropriate speeds sometimes occur. This is due to the numerical instabilities that arise as a result of using the Euler numerical integration method in the direct dynamic module. It is expected that stabilization may be achieved through the use of more sophisticated numerical integration techniques (e.g. Runge-Kutta method [WiF88]).

When high speed gaits were attempted, instability resulted and the simulated system "crashed". Preliminary indications are that stabilization of high speed locomotion may require some type of control over the sampling rate. The sampling intervals used by the NC varied between .01 and .001 seconds. Sampling rates were chosen to be the maximum for realistic motion because, obviously, the less often one need sample, the faster the NC will run. When sampling was done too rarely, the NC sometimes found itself unable to solve the dynamic equations.

Another problem with infrequent sampling is that the links of the robot moved into noticeably unrealistic positions (e.g. legs below the floor) between sampling intervals. It is found that the frequency of sampling needed for realistic motion increases with the rapidity of the motion. The time step used in the dynamic computation is too fine for the graphics display. Currently, a frame is displayed every 0.05 seconds of simulation time. This rate is sufficient for normal animation frame rate (25-30 frames per second). In some cases, the legs were slipping. The reasons for that are due to the simplification assumptions that are used in the NC, which include: the assumption of ball-and-socket joints, the neglecting of the swinging legs effect on the body motion, and not including the friction nor the air viscosity in the model.

6.6. RESEARCH CONTRIBUTIONS

The feasibility of the proposed hybrid locomotion control system has been demonstrated by the results presented in this chapter. While considerable work remains to be done, it is felt that the current simulation experiments allowed the opportunity of examining the effectiveness of merging the methodologies of computer numerical control with the methodologies of motor learning. The result, demonstrated in this chapter, showed that both methodologies can act together to provide an autonomous locomotion control system that can improve its performance through practice and experience.

Several of the specific contributions that have been made to articulated body locomotion will be discussed in the next few paragraphs.

On the macroscopic level, a vital link has been completed between "motor control" and "motor learning". This was a necessary link that brought together man and robot at equal levels and allowed them to successfully communicate.

Several other contributions were achieved:

(1) The organization of the proposed hybrid controller. The partitioning of the problem of locomotion control into functional modules: programmed-in, Kinematic, Dynamic, and skill acquisition.

(2) The system represents a prototypical example of a knowledge-based system that is coupled with numerical computing. The coupling is of the shallow type.

(3) The kinematic motion control algorithms of chapter 3, that involved setting the kinematic model of the robot and developing the algorithms for automatic leg positioning as well as automatic body height, pitch, and roll regulation over undulating terrain, represent a novel kinematic treatment of the locomotion control problem.

(4) The NC organization. The partitioning of the dynamic analysis to deal separately with the legs that are in swing phase and the ones in stance phase. This has allowed the easier handling of the rather complicated sequence of the closed kinematic chains that are formed and broken many times in the course of the robot navigation.

(5) The use of linear programming to generate the torque profiles for the joints.

(6) The inversion of Armstrong's equations and extending them to the case of closed kinematic chains.

(7) The parallel pipelining NC computation model for near real-time solution of the locomotion control of the four-legged robot. Although this was not implemented in the simulation system, it is expected that its implementation would follow the same methodology presented in [AMO87].

(8) The LC organization and the new model for dynamics-based motion skill acquisition.

6.7. FUTURE WORK AND EXTENSIONS

A number of research problems have become increasingly clear as the phases of this research have been completed. For example, the skill based AI systems suggested in section 5.5 might produce more insights into the evolution of the control and learning phases. Figure 6.12 shows one of these systems that represents multiple robots animation research [MoA88c] and measuring learning progress research [MoA88b]. Other possible extensions are:

(1) Implementing the distributed pipeline version of the NC that is described in chapter 4. The distributed direct dynamics paper by Armstrong, Marsland, Olafsson and Schaeffer might provide the

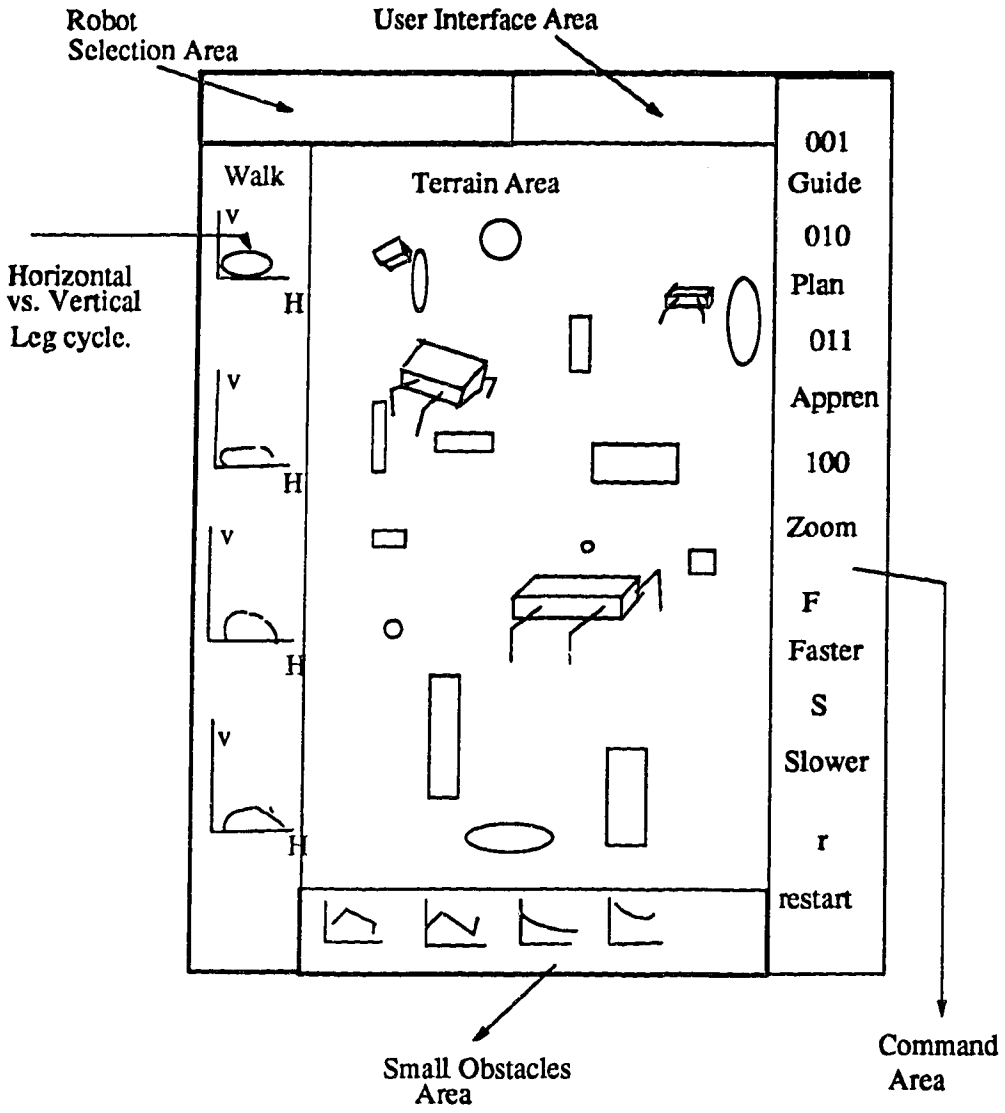

Figure 6.12 The Multi-Robot Research

basis for such an extension.

(2) Enhancing the LC learning capabilities by allowing the LAS to add completely new trees into the forests of the feature trees of the LC. This extension is related to the work in machine learning entitled "constructive induction" [WCB86].

(3) Designing a Balance Maintenance Subsystem (BMS) that would work like Hock's system [Hoc66], for maintaining the robot balance under undulating terrain conditions.

(4) Designing an Explanation Based Subsystem (EBS) that is to be attached to the LC and be able to interact with the animator to explain the locomotion behavior of the robot. A similar explanation-based learning paradigm could be found in [DeM86].

References

[ABC86] M. Abernathy, M. Bernabe, L. Chandler, M. Wilkins and M. Wolfe, A Conceptual Design of an Intelligent Front-End to Scientific Application Programs, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[Ada71] J. Adams, A Closed-Loop Theory of Motor Learning, *Journal of Motor Behaviour, 3*, 1971.

[Ada76] J. Adams, in *Learning and Memory: An Introduction*, Homewood, IL: Dorsey, 1976.

[Akk79] N. Akkas, *Progress in Biomechanics*, in progress in Biomechanics, 1979.

[Alb75] J. Albus, A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC), *Journal of Dyn., Sys., Meas., Control*, 1975.

[Alb79] J. Albus, Mechanisms of Planning and Problem Solving in the Brain, *Mathematical Biosciences, 45* , 1979.

[Alb81] J. Albus, Brain, Behavior, and Robotics, *BYTE books*, 1981.

[And83] J. Anderson, in *The Architecture of Cognition*, Harvard University Press, Cambridge, Mass., 1983.

[And88] R. Anderson, in *A Robot Ping-Pong Player: Experiment in Real-time Intelligent Control*, The MIT Press, Cambridge, Massauchusetts, 1988.

[And85] in *Applications in Artificial Intelligence*, S. Andriole (ed.), Petrocelli Books, Inc., Princeton, N.J., 1985.

[Anz78] Y. Anzai, Learning Strategies by Computer, *proceedings of the second National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978.

[Arb85] M. Arbib, Local Organizing Processes and Motion Schemas in Visual Perception, *Machine Intelligence*, 1985, Ellis Horwood Ltd, Chichester.

[Arm79] W. Armstrong, Recursive Solution to the Equations of Motion of an N-Link Manipulator, *Proceedings Fifth World Congress on the Theory of Machines and Mechanisms American Soc. of Mech. Eng.*, 1979.

[ArG85] W. Armstrong and M. Green, The Dynamics of Articulated Rigid Bodies for Purpose of Animation, *Proceedings of Graphics Interface'85*, 1985.

[AGL86] W. Armstrong, M. Green and R. Lake, Near-Real Time Control of Human Figures Models, *Graphics Interface-86*, 1986.

[AMO87] W. Armstrong, T. Marsland, M. Olafsson and J. Schaeffer, Solving Equations of Motion On a Virtual Tree Machine, *SIAM J. Sc. Stat. Comput., 8*, 1987.

[BaB78] N. Badler and R. Bajcsy, Three Dimensional Representations for Computer Graphics and Computer Vision, *Computer Graphics, 12*, 1978.

[BMW87] N. Badler, K. Manoochehri and G. Walters, Articulated Figure Positioning by Multiple Constraints, *IEEE Computer Graphics and Applications, 7(6)*, 1987.

[BeP81] A. Bejczy and R. Paul, Simplified Robot Arm Dynamics for Control, *Proceedings IEEE Decision and Control*, 1981.

[BiH86] E. Binder and J. Herzog, Distributed Computer Architecture and Fast Parallel Algorithms in Real-time Robot Control, *IEEE Trans. on Systems, man, and*

Cybernetics, SMC-16(9), 1986.

[Bra78] P. Brazdil, Experimental Learning Model, AISB/GI, *Society for the Study of Artificial Intelligence and Simulation of Behaviour* , 1978.

[BrC89] A. Bruderlin and T. Calvert, Goal-Directed, Dynamic Animation of Human Walking, *Submitted to SIGGRAPH89*, 1989.

[BuW76] N. Burtnyk and M. Wein, Interactive Skeleton Techniques for Enhancing Motion Dynamics in Keyframe Animation, *Comm. ACM 19(10)*, 1976.

[CCP80] T. Calvert, J. Chapman and A. Patla, The Integration of Subjective and Objective Data in the Animation of Human Movement, *Computer Graphics, 14*, 1980.

[CCP82] T. Calvert, J. Chapman and A. Patla, Aspects of the Kinematic Simulation of Human Movement, *IEEE Computer Graphics And Applications*, 2, 1982.

[CaO86] S. Campbell and S. Olson, WX1- An Expert System for Weather Radar Interpretation, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[Cha69] D. Chaffin , A Computerized Biomechanical Model-Development and Use in Studing Gross Body Actions, *Journal of Biomechanics, 2* , 1969.

[Cha86] K. Chalfan, An Expert System for Design Analysis, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[CoF81] in *The Handbook of Artificial Intelligence*, P. Cohen and E. Feigenbaum (ed.), HeirisTech Press, Stanford Calif., 1981.

[CoK86] D. Cooper and J. Kornell, Combining Symbolic and Numeric Methods for Automated Induction, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[DeM86] G. DeJong and R. Mooney, Explanation-Based Learning: An Alternative View, *Machine Learning,1-2*, 1986.

[Doo82] M. Dooley, Anthropometic Modeling Programs- A Survey, *IEEE Computer Graphics And Applications*, 2, 1982.

[DrS66] N. Draper and H. Smith, in *Applied Regression Analysis*, John Wiley & Sons, New York, 1966.

[EsA84] B. Espiau and G. Andre, Sensory-based Control for Robots and Teleoperators, *Proc. of the 5th IF-TOMM Symposium on Theory and Practice of Robots and Manipulators*, 1984, Italy, June.

[Fet82] W. Fetter, A Progression of Human Figures Simulated by Computer Graphics, *IEEE Computer Graphics And Application (2)*, 1982.

[FrV69] A. Frank and M. Vukobratovic, On the Synthesis of a Biped Locomotion Machine, *8th International Conference on Medical and Biological Engineering*, 1969, Evanston, Illinois.

[Fu70] K. Fu, Learning Control Systems- Review and Outlook, *IEEE Transactions on Automatic Control, AC-15, no. 2*, 1970.

[GiM82] C. Ginsberg and D. Maxwell, Graphical Marionette: A Modern-Day Pinocchio, Tech. Rep. , 1982, Architecture Machine Group, MIT.

[GiM85] M. Girard and A. Maciejewski, Computational Modeling for the Computer Animation of Legged Figures, *Computer Graphics 19:3* , 1985.

[GlK86] N. Gladd and N. Krall, Artificial Intelligence Methods for Facilitating Large-Scale Numerical Computations, *Workshop on coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[Gom84] J. Gomez, TWIXT: A 3D Animation System, *Proceedings of EUROGRAPHICS'84*, 1984.

[GGS81] V. Gurfinkel , E. Gurfinkel, A. Shneider, A. Lenskey and L. Shitilman, Walking Robot with Supervisory Control, *Mechanics and Machine Theory, 16*, 1981.

[Hat77] H. Hatze, A Complete Set of Control Equations for the Human Musculo-skeletal System, *Journal of Biomechanics, 10*, 1977.

[Hee73] Remotely Manned Systems- Exploration and Operation in Space, *Proceedings of the First National Conference*, Pasadena, CA, 1973, California Inst. of Technology.

[Her78] D. Herbison-Evans, NUDES2: A Numeric Utility Displaying Ellipsoid Solids, *Computer Graphics, 12*, 1978.

[Her80] D. Herbison-Evans, Rapid Raster Ellipsoid Shading, *Computer Graphics, 13*, 1980.

[Hoc66] R. Hock, Control of a Legged Vehicle, *Research Report, Pacific Northwest Lab*, 1966, Richland, Washington.

[HoR78] J. Holland and J. Reitman, Cognitive Systems Based on Adaptive Algorithms, in *Pattern Directed Inference Systems*, D. Waterman and F. Hayes-Roth (ed.), Academic Press, 1978.

[Hol78] J. Hollerbach, A Study of Human Motor Control through Analysis and Synthesis of handwriting., *Ph.D. dissertation*, 1978, MIT.

[Hol84] J. Hollerbach, Dynamic Scaling of Manipulator Trajectories, *Transactions of the ASME- 106E*, 1984.

[HoM65] W. Hooker and G. Margulies, The Dynamical Attitude Equations for an n-Body Satellite, *Journal Astronaunt Science, 4-123*, 1965.

[HPH76] R. Huston, C. Passarello, R. Hessel and M. Harlow, On Human Body Dynamics, Annals Biomedical Engineering, 4, 1976.

[IsC87] P. Isaacs and M. Cohen, Controlling Dynamic Simulation with Kinematic Constraints Behavior Functions, and Inverse Dynamics, *Proceedings of Graphics Interface'87*, 1987.

[KaS70] T. Kane and M. Scher, Human Self-Rotation by Means of Limb Movements, *Journal of Biomechanics, 3*, 1970.

[KBB82] D. Kochanek, R. Bartels and K. Booth, A Computer System for Smooth Keyframe Animation, Tech. Report, 1982, University of Waterloo.

[KoB82] J. Korein and N. Badler, Techniques for Generating the Goal-Directed Motion of Articulated Structures, *IEEE Computer Graphics And Applications*, 1982.

[Kro86] B. Kroyer , Animating with a Hierarchy, *Proceedings Siggraph-86*, 1986, in Seminar on Advanced Computer Animation.

[LGI86] P. Langley, J. Gennari and W. Iba, Hill-Climbing Theories of Learning, in *Machine Learning: A Guide to Current Research*, T. Michell, J. Corbonell and R. Michaleski (ed.), 1986.

[Lat85] R. Lathrop, Parallelism in Manipulator Dynamics, *The International Journal of Robotics Research, (4-2)*, 1985.

[LeC86] C. Lee and P. Chang, Efficient Parallel Algorithm for Robot Inverse Dynamics Computations, *IEEE Transactions on Systems, man, and Cybernetics, SMC-16(4)*, 1986.

[LiM68] R. Liston and R. Mosher, A Versatile Walking Truck, *Proceedings of the Transportation Engineering Conference*, 1968.

[LoT86] P. Lounamaa and E. Tse, The Simulation and Expert Environment, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[LWP76] J. Luh, M. Walker and R. Paul, On-line Computational Scheme for Mechanical Manipulators, *Journal of Dyn., Sys., Meas., Contr., 102*, 1976, (Errata Correction September 1980).

[Lun86] D. Lundin , *Simulation*, From Reynolds, Advanced Computer Animation, Siggraph'86, 1986.

[Lun87] R. Lundin, Motion Simulation, *in Advanced Computer Animation, Siggraph-87*, 1987.

[MJW85] R. Marshall, R. Jensen and G. Wood, A General Newtonian Simulation of An N-segment Open Chain Model, *J. Biomechanics, 18-5*, 1985.

[McG88] T. McGeer, Powered Flight, Child's Play, Silly Wheels, and walking Machines, *Canadian Conference on Electrical and Computer Eng., Nov.*, 1988.

[McG68] R. McGhee, Some Finite State Aspects of Legged Locomotion, *Mathematical Biosciences, 2-1*, 1968.

[McK72] R. McGhee and K.Jain , Some Properties of Regularly Realizable Gait Matricies, *Mathematical Biosciences, 13*, 1972.

[McO76] R. McGhee and D. Orien, A Mathematical Programming Approach to Control of Joint Positions and Torques in Legged Locomotion Systems, *Proceeding of Symposium on Theory and Practice of Robots and manipulators*, Warsaw, Ploand, 1976.

[McG80] R. McGhee, Robot Locomotion with Active Terrain Accommodation , *Proceedings of National Science Foundation Robotics Research Workshop*, 1980, University of Rhode Island.

[McG83] R. McGhee, in *In Advances in Automation and Robotics*, G. Saridis (ed.), JAI Press, 1983.

[MMS81] D. Meno, J. Mansour and S. Simon, Analysis of Human Swing Leg Motion During Gait and its Clinical Applications, *Journal of Biomechanics of Sport*, 1981.

[Mey85] A. Meystel, Baby-robot: On the Analysis of Cognitive Controllers for Robotics, *IEEE 1985 Proceedings of the International Conference on Cybernetics and Society*, 1985, Tucson, Arizona.

[Mea86a] A. Meystel and et al., Hierarchical Algorithm for Suboptimum Trajectory Planning and Control, in *Recent Trends in Robotics*, M. Jamshidi, J. Luh and M. Shahiupooz (ed.), North-Holland, Elsieve Science Publ., New York, 1986.

[Mea86b] A. Meystel and et al., Nested Hierarchical Controller for Intelligent Mobile Autonomous System, *Procceding of the Intl. Congress on Intelligent Autonomous Systems*, 1986, Amsterdam.

[Mea87a] A. Meystel and et al., Navigation Algorithm for a Nested Hierarchical System of Robot path Planning among Polyhedral Obstacles, *Proceeding of IEEE Intl. Conference on Robotics and Automation*, 1987, Raleigh-Durham.

[Mea87b] A. Meystel and ei al., Planning Minimum Time Trajectories in the Traversability Space of a Robot, *Proceeding IEEE Intl. Symposium on Intelligent Control*, 1987, Philadelphia.

[MUN81] T. Mitchell, P. Utgoff, B. Nudel and R. Banerji, Learning Problem-solving Heuristics Through Practice , *in Proceeding 7th IJCAI* , Vancouver, B.C., 1981.

[Mit83] T. Mitchell, Learning and Problem Solving , in *in Proceedings 8th International Conference on Artificial Intelligenece*, A. Bundy (ed.), Karlsruhe, F.R.G., 1983.

[MMS85] T. Mitchell , S. Mahadevan and L. Sterinberg, LEAP: A Learning Apprentice For VLSI Design, *Proceeding of IJCAI-85*, 1985.

[MiS84] H. Miura and I. Shimoyama, Dynamic Walk of a Biped, *International J. Robotics Research, 3*, 1984.

[MSM85] H. Miura, I. Shimoyama, M. Mitsuishi and H. Kimura, Dynamical Walk of Quadrupped Robot (Collie), *Second International Symposium on Robotics Research*, 1985, Cambridge: MIT Press.

[MoA88a] A. Mohamed and W. Armstrong, An Experimental Autonomous Articulated Robot that Can Learn, *Proceeding of the International Conference on Computational Intelligence 88*, 1988, Chapter of ACM, September 26-30.

[MoA88b] A. Mohamed and W. Armstrong, A Hybrid Numerical/Knowledge-Based Locomotion Control System for A Multi-Legged Manipulator Robot, *Proceeding of the 8th International Workshop on Expert Systems and their Applications*, 1988, Avignon, France, June 1-5.

[MoA88c] A. Mohamed and W. Armstrong, Towards A Computational Theory for Motion Understanding: The Expert Animator Model, *Proceeding of the 4th International Conference on Artificial Intelligence for Space Applications (AISA'88), Sponsored by NASA.*, 1988.

[Moh88] A. Mohamed, Description of the Hybrid Numerical/Knowledge Based Locomotion Control System, *Technical Report- Dept. of Computer Sc., The University of Alberta*, 1988.

[MoA88a] A. Mohamed and W. Armstrong, A Learning Network for Rule-Based Control of Motion, *Working paper*, 1988.

[MoA88b] A. Mohamed and W. Armstrong, Measuring Learning Progress in Intelligent Autonomous Robots, *Volume 5.6, December 1988 of the Journal of Robotic Systems*, 1988.

[Mor83] H. Moravec, The Stanford Cart and the CMU Rover, Report of the Robotic Institute, Carnegie-Mellon University, Pittsburgh, PA, 1983.

[NAS78] NASA, Large Space Systems Technology, *NASA Conference Publication 2035*, Washington, DC, 1978, Scientific and Technical Information Office, NASA.

[NGS86] P. Nachtsheim , W. Gevarter, J. Stutz and C. Banda, A Knowledge Based Expert System for Scheduling of Airborne Astronomical Observations, *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[NaT74] K. Narendra and M. Thathachar, Learning Automata- A Survey, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-4*, 1974.

[NWM77] K. Narendra, E. Wright and L. Mason, Application of Learning Automata to Telephone Traffic Routing and Control, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-7*, 1977.

[Nev78] D Neves, A Computer Program that Learns Algebric Procedures by Examining Examples and Working Problems in a Textbook, *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978.

[Nor80] D. Norman , Twelve Issues for Cognitive Science, *Cognitive Science, 4*, 1980.

[OkP73] D. Okhotsimski and A. Platonov, Control Algorithms for a Walker Climbing over Obstacles, *Proceedings 3rd IJCAI*, 1973.

[OGA74] D. Okhotsimski, Y. Golubev and L. Alekseeva, A stabilization Algorithm for an Automatic Walking Machine, *Proceedings of 4th IFAC Symposium on Automatic Control in Space*, 1974, Tsakhadzor, Armenian USSR.

[OrM73] D. Orin and R. McGhee, An Interactive Computer-Control System for a Quadruped Robot, *Proc. of Symposium on theory and practice of Robots and Manipulators (ROMANSY)*, 1973, Italy.

[OTM84] F. Ozguner, J. Tsai and R. McGhee, An Approach to the Use of Terrain-Preview Information in Rough-Terrain Locomotion by a Hexapod Walking machines, *International J. Robotic Research, 3*, 1984.

[PaF73] W. Park and K. Fegley, Control of A Multi-Legged Vehicle, *Proc. of V IFAC Symposium on Automatic Control in Space*,, 1973, Genoa, Italy.

[PlB81] S. Platt and N. Badler, Animating Facial Expressions, *Siggraph'81*, 1981.

[PoB78] A. Polit and E. Bizzi, Processes Controlling Arm Movements in Monkeys, *Science, 201*, 1978.

[PrM79] T. Procyk and E. Mamdani, A Linguistic Self-Organizing Process Controller, *Automatica*, 1979.

[Rai77] M. Raibert, Analytical Equations Versus Table Look-up: A Unifying Concept, *Procceding IEEE Conf. Decision and Control*, 1977.

[Rai81] M. Raibert, Dynamic Stability and Resonance in a One-legged Hopping Machine, *Fourth Symposium on Theory and Practice of Robots and Manipulators*, 1981, Warsaw Polish Scientific Publishers.

[RaS83] M. Raibert and I. Sutherland, Machines that Walk, *Scientific American, 248*, 1983.

[RBM84] M. Raibert, H. Brown and S. Murthy, 3D Balance Using 2D Algorithms?, *First International Symposium of Robotics Research*, 1984, Cambridge MIT Press.

[Rai84] M. Raibert, Special Issue on Legged Systems, *International J. Robotics Research, 3*, 1984.

[Rai85] M. Raibert, Four-Legged Running With One-Legged Algorithms, *Second International Symposium on Robotics Research*, 1985, Cambridge MIT Press.

[Rai86] M. Raibert, in *Legged Robots that Balance*, The MIT Press, Cambridge, Massachusetts, 1986.

[Ric83] E. Rich, in *Artificial Intelligence*, Reading, 1983.

[Rob81] D. Robson, Object-Oriented Software Systems, *BYTE, 6, no. 8, 74/86*, 1981.

[RuN82] D. Rumelhart and D. Norman, Simulating a Skilled Typist: A Study of Skilled Cognitive-motor Performance, *Cognitive Science, 6*, 1982.

[SaO78] G. Savage and J. Officer, CHOREO: An Interactive Computer Model for Dance , *Intl. Journal of Man-Machine Studies, 10*, 1978.

[ScW72] R. Schmidt and J. White, Evidence for an Error Detection Mechanism in Motor Skills: A Test of Adam's Closed-Loop Theory, *Journal of Motor Behaviour*, 1972.

[SZH79] R. Schmidt, H. Zelarnik, B. Haukins, J. Frank and J. Quinn , Motor Output Variability: A Theory for the Accuracy of Rapid Motor Acts, *Psychological Review 86*, 1979.

[Seg87] A. Segre, Explanation-Based Manipulator Learning: Acquiring Robotic Manufacturing Schemata, *Ph.D. Thesis, Dept. of Electrical and Computer Engineering*, 1987, University of Illinois at Urban-Champaign.

[SeA75] A. Seireg and R. Arkivar, The Prediction of muscular Load Sharing and Joint Forces in the Lower Extremities During Walking, *Journal of Biomechanics, 8*, 1975.

[ShZ81] D. Shapiro and R. Zernicke, Evidence for Generalized Motor Programs Using Gait-pattern Analysis, *Journal of Motor Behaviour, 13*, 1981.

[SBS82] J. Simons, H. Brussel, D. Schutter and J. Verhaert, A Self-Learning Automaton with Variable Resolution for High Precision Assembly by Industrial Robots, *IEEE Transactions on Automatic Control, AC-27, no. 5*, 1982.

[SMC77] R. Smith, T. Michell, R. Chestek and B. Buchanan , A Model for Learning Systems, *Proceeding 5th IJCAI*, Cambridge, MA, 1977.

[SoW87] S. Song and K. Waldron, Geometric Design of a Walking Machine for Optimal Mobility, *Journal of Mechanisms, Transmissions, and Automation in Design, 109/21*, 1987.

[Ste85] M. Stefik, Rule-Oriented Programming in LOOPS, *Symposium on Knowledge Engineering and Artificial Intelligence, Information Processing Society of Japan, 65/71*, 1985.

[Ste82] R. Stein, What Muscle Variable(s) does the Nervous System Control in Limb Movements?, *Behavour Brain Science, 5*, 1982.

[STM83] R. Stein, J. Taylor and P. Murphy, Discharge Patterns of Dynamic and Static Gamma Motoneurones during Locomotion in the Premammillary cat and their Implications for Control of Locomotion, *Soc. Neurosci.; Abstr., 9*, 1983.

[SMT85] R. Stein, P. Murphy and J. Taylor, Phasic and tonic modulation of impulse rates in 'gamma' motoneurones during locomotion in the premammillary Cat and their implications for Control of Locomotion, in *The Muscle Spindle*, I. Boyd and M. Gladden (ed.), 1985.

[StB85] S. Steketee and N. Badler, Parametric Keyframe Interpolation Incorporating Kinetic Adhustment and Phrasing Control, *Computer Graphics, 19*, 1985.

[Stu84] D. Sturman, Interactive Keyframe Animation of 3-D Articulated Models, *Proceeding Graphics Interface-84*, 1984.

[Stu87] D. Sturman, A Discussion on the Development of Motion Control Systems , Course Notes- Siggraph-87, 1987.

[TEC86] S. Talukdar, A. Elfes, E. Cardozo, R. Joobboni, M. Rychener and R. Benares, A System for Distributed Problem Solving , *Workshop on Coupling Symbolic and Numerical Computing in Expert Systems*, 1986.

[Tom81] R. Tomovic, Active Modular Unit for Lower Limb Assistive Device, in *in Advances in External Control of Human Extremities*, 1981.

[ToB86] R. Tomovic and G. Bekey, Robot Control by Reflex Actions, *Proceeding IEEE Conference on Robotics and Automation*, 1986, IEEE Computer Society Press.

[TuP87] S. Tupling and M. Pierrynowski, Use of Cardan Angles to Locate Rigid Bodies in three-dimensional Space., *Med. and Biol Eng. and Comput., 25*, 1987.

[Tur63] A. Turing, Computing Machinery and Intelligence, in *in Computers and Thought*, E.A. Feigenbaum and J. Feldman (ed.), McGraw-Hill, New York, 1963.

[Unm86] M. Unmeel, Knowledge Based Systems for Computational Aerodynamics and Fluid Dynamics, in *in Knowledge Based Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J., 1986.

[Vuk73] M. Vukobratovic, Analysis of Energy Demand Distribution Within Anthropomorphic Systems, *Transactions ASME, Journal of Dyn., Meas, and*

Control, 1973.

[WVP84] J. Waldron, J. Vohnount , A. Pery and R. McGhee, Configuration Design of the Adaptive Suspension Vehicle, *International J. Robotic Research, 3*, 1984.

[Wat70] D. Waterman, Generalization Learninbg Techniques for Automating the Learning of Heuristics, *Artificial Intelligence, 1* , 1970.

[Wat75] D. Waterman, Adaptive Production Systems, *Proceedings of the 4th IJCAI*, 1975.

[WaH78] D. Waterman and F. Hayes-Roth, An Overview of Pattern-directed Inference Systems, in *in Pattern Directed Inference Systems*, Academic Press, 1978.

[WiB85] J. Wilhelms and B. Barsky, Using Dynamic Analysis for the Animation of Articulated Bodies such as Human and Robots, *Proceedings of Graphics Interface'85*, 1985.

[Wil86] J. Wilhelms , Virya- A Motion Control Editor for Kinematic and Dynamic Animation, *Proceedings of Graphics Interface'86*, 1986.

[WiF88] J. Wilhelms and D. Forsey, Techniques for Interactive Manipulation of Articulated Bodies Using Dynamics Analysis, *Graphics Interface 88*, 1988.

[WCB86] D. Wilkins, W. Clancey and B. Buchanon, Overview of the ODYSSEUS Learning Apprentice, in *in Machine Learing: A Guide to Current Research*, T. Michell, J. Corbonell and R. Michaleski (ed.), 1986.

[WiS78] R. Williams and A. Seireg, Interactive Modeling and Analysis of Open or Closed Loop Dynamics Systems with Redundant Actuators, *ASME, 42*, 1978.

[WiB78] N. Winkless and I. Browning, in *Robots On Your Doorsteps*, Robotic Press, Portland, OR, 1978.

[Win75] P. Winston, Learning Structural Descriptions from Examples, in *The Psychology of Computer Vision*, P. Winston (ed.), McGraw-Hill, New York, 1975.

[WSK86] H. Winston, R. Smith, M. Kleyn, T. Mitchell and B. Buchanon, Learning Apprentice Systems Research at Schlumberger, in *in Machine Learning: A Guide to Current Research*, T. Michell, J. Corbonell and R. Michaleski (ed.), 1986.

[WiL85] J. Winter and S. Lawrence, Analysis of Fundamental Human Movement Patterns Through the Use of In-Depth Antagonistic Muscle Models, *IEEE Journal of Biomedical Engineering*, 1985.

[Wit77] J. Wittenburg, in *Dynamics of Systems of Rigid Bodies*, Teubner, Stuttgart, 1977.

[Wit87] J. Wittenburg, Multibody Dynamics: A Rapidly Developing Field of Applied Mechanics, *MECCANICA- 22*, 1987.

[Zel82] D. Zeltzer, Motor Control Techniques for Figure Animation, *IEEE Computer Graphics And Applications 2(9)*, 1982.

APPENDIX A

Bidirectional Search Algorithm for Macro/Strategic Navigation

In this appendix we present a road map production system that uses a bidirectional search algorithm. It enables the robot to navigate and execute high-level commands such as "go from location x to location y". The search algorithm yields the shortest path solution. In the event of unexpected occurrences such as obstruction of the shortest path by a new obstacle, the algorithm would enable the robot to quickly search for the shortest alternative path from its current position to its destination. The system can readily adapt to a dynamically changing map.

The following are the terms used in the algorithm:

(a) OPENT1 is the set, initially set to the initial location of the robot s, of locations currently to be processed or expanded using production rules in the forward direction generating subsequent robot locations. These locations are guaranteed to be reachable from s by the existence of sequences of production rules.

(b) CLOSEDT1 is the set, initially set to empty, of ancestors of the elements of OPENT1.

(c) OPENT2 is the set, initially set to the initial location of the robot s, of successors of OPENT1 which are not in CLOSEDT1, i.e, ancestors of OPENT1. This is required to prevent the generation of loops. OPENT2 becomes OPENT1 (OPENT1 = OPENT2) in the next level of processing.

(d) CLOSEDT2 is the set, initially set to empty, of processed elements of OPENT1. This set would become ancestors or members of CLOSEDT1 (CLOSEDT1 = CLOSEDT1 + CLOSEDT2) in the next level of processing.

(e) OPENB1 is the set, initially set to goal location g, of goal locations currently to be processed or expanded using the production rules in the backward direction (in the navigation problem forward and backward production rules are the same) generating subsequent goal locations or subgoals. The subgoals are guaranteed to reach g by the existence of sequences of production rules.

(f) CLOSEDB1 is the set, initially set to empty, of successors of the elements of OPENB1

(g) OPENB2 is the set, initially set to goal location g, of ancestors of OPENB1 which are not in CLOSEDB1, i.e, successors of OPENB1. This is required to prevent the generation of loops. OPENB2 becomes OPENB1 (OPENB1 = OPENB2) in the next level of processing.

(h) CLOSEDB2 is the set, initially set to empty, of processed elements of OPENB1. This set would become successors or members of CLOSEDB1 (CLOSEDB1 = CLOSEDB1 + CLOSEDB2) in the next level of processing.

(i) Type(x) is a function that generate the road type of x, that is either EW (East-West) or NS (North - South) type.

(j) ChangeType(g) is a procedure that change the type of g by selecting the junctions of g closest to the initial location s.

The Algorithm

```
Begin
 if Type(s) <> Type(g) then ChangeType(g);
 OPENT1 := s;
 OPENT2 := s;
 CLOSEDT1 := empty;
 CLOSEDT2 := empty;
 OPENB1 := g;
 OPENB2 := g;
 CLOSEDB1 := empty;
 CLOSEDB2 := empty;
 start := .TRUE.;
 While OPENT1 <> empty and OPENB1 <> empty do
 begin

   While not start do
```

```
    begin
      start := .FALSE.;
      While OPEN1 <> empty do
      begin
        Initialize OPENT2 and CLOSEDT2 to empty;
        Select a node n on OPENT1, remove it from OPENT1;
        Put it on CLOSEDT2 (next set of ancestors);
        Expand node n, applying production rules,
        generating the set N of its successors that
        are not ancestors (elements of N) to point back to n;
        Put N into OPENT2 (next level of state nodes);
        Tag each element of N to point back to n. That is
        if x is in N, then x(n) denote that x was generated
        by n, or n is the ancestor of x.
      end;

      While OPENB1 <> empty do
      begin
        Initialize OPENB2 and CLOSEDB2 to empty;
        Select a node m on OPENB1, remove it from OPENB1;
        Put it on CLOSEDB2 (next set of successors);
        Expand node m, applying production rules,
        generating the set M of its ancestors that
        are not successors (element of CLOSEDB1) of m;
        Put M into OPENB2 (next level of goal nodes);
        Tag each element of M to point back to m. That is
        if y is in M, then y(m) denote that y was generated
        by m, or m is a successor of y.
      end;

      CLOSEDT1 := CLOSEDT1 + CLOSEDT2;
      CLOSEDB1 := CLOSEDB1 + CLOSEDB2;
    end;
    While OPENT2 * OPENB2 <> empty do
    begin
      For each element of OPENT2 * OPENB2 that is connected:
      Trace the path to s and g;
      Retain the minimum distance path;
      Delete all connected elements of OPENT2 * OPENB2 from OPENT2, and OPENB2;
    end;
    OPENT1 := OPENT2;
    OPENB1 := OPENB2;
  end;
end;
```