Virtual Model Control of a Hexapod Walking Robot

by

Ann L. Torres

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

Bachelor of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author . . . .
Department of Mechanical Engineering
May 10, 1996

Certified by . . . .
Gill A. Pratt
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Professor Peter Griffith
Chairman, Department Committee on Undergraduate Theses

Virtual Model Control of a Hexapod Walking Robot

by
Ann L. Torres

Submitted to the Department of Mechanical Engineering
on May 10, 1996, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Mechanical Engineering

Abstract

Since robots are typically designed with an individual actuator at each joint, the control of these systems is often difficult and non-intuitive. This thesis explains a more intuitive control scheme called Virtual Model Control. This thesis also demonstrates the simplicity and ease of this control method by using it to control a simulated walking hexapod.

Virtual Model Control uses imagined mechanical components to create virtual forces, which are applied through the joint torques of real actuators. This method produces a straightforward means of controlling joint torques to produce a desired robot behavior. Due to the intuitive nature of this control scheme, the design of a virtual model controller is similar to the design of a controller with basic mechanical components. The ease of this control scheme facilitates the use of a high level control system which can be used above the low level virtual model controllers to modulate the parameters of the imaginary mechanical components.

In order to apply Virtual Model Control to parallel mechanisms, a solution to the force distribution problem is required. This thesis uses an extension of Gardner's Partitioned Force Control method which allows for the specification of constrained degrees of freedom.

This virtual model control technique was applied to a simulated hexapod robot. Although the hexapod is a highly non-linear, parallel mechanism, the virtual models allowed text-book control solutions to be used while the robot was walking. Using a simple linear control law, the robot walked while simultaneously balancing a pendulum and tracking an object.

Thesis Supervisor: Gill A. Pratt
Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

To misquote a famous scientist,

If I have accomplished anything, it is because I have stood on the shoulders of giants.

Thanks to my advisor, Gill Pratt, for his assistance and enthusiasm for my work in the lab.

Also thanks to all the members of the Leg Lab for always making my time in lab interesting and exciting.

Special thanks goes to Jerry Pratt whose previous hard work and continued support made this thesis possible. Most of my work is built on Jerry's Masters thesis and his answers to my endless barrage of questions. Also, much thanks to Karl Leeser for his many suggestions and invaluable lessons.

Most importantly, thanks to Evan Wies for his endless support and encouragement.

This research was supported in part by the Office of Naval Research, Grant #N00014-93-1-0333

Contents

List of Figures

Chapter 1

Introduction

Due to design constraints, robots typically have actuators at the joint level. This produces an actuation space which is non-intuitive. Because of the non-linearities at the joint level, generally caused by sine and cosine functions which are used when describing motion in joint space, the control of a robotic system is often difficult to understand and visualize. For example it is often not clear how to specify torques at the ankles, knee, and hip joint of a legged robot in order to produce smooth motion in the body. However, it is easier to just specify the motion of the body and have the non-linearities of joints abstracted away.

A control scheme dubbed Virtual Model Control was developed by Pratt [5] in order to make the control of robotic systems more simple and intuitive. In Virtual Model Control, virtual mechanical components are used to describe the desired behavior of a robot. The motion of the robot is altered by modulating the parameters of the virtual mechanical components. For example, if one desired a robot to stand at a certain height, a virtual spring from the ground to the robot's body could be connected. The height at which the robot stands could be changed by modulating the rest length of the spring. The virtual components are used to transform the desired behavior of the robot into a generalized virtual force on the robot.

The generalized virtual force produced by the virtual components is mapped to joint torques using virtual model math. When these torques are applied to the joints, the robot will behave exactly as if the virtual components were really there. Virtual Model Control is a powerful tool in robotic design because it abstracts away the complexity of control in the joint space and only requires an intuitive understanding of the desired behavior of the robot.

This thesis proves the validity and usefulness of Virtual Model Control by using it to enable a simulated hexapod robot to walk. In order to control the hexapod, two virtual model controllers were developed. One controller regulates the motion of the body in stance and the other regulates the motion of the legs in swing. The development of the controllers involved deriving the virtual model math for each set of linkages. The mathematics developed allowed the control of the joints to be ignored and the control of the robot's body to be dictated by desired motions. Appropriate virtual mechanical components were also chosen for each controller to induce the desired motion of the body.

By developing these two virtual model controllers, a higher level controller was implemented in order to make the robot walk. A simple walking algorithm regulated the virtual controllers. The walking algorithm chose the appropriate tripod for the task and set the parameters of the virtual components based on the desired behavior of the robot. Using this method, the robot was induced to walk while requiring only three user inputs. These user inputs specified the robots speed, the direction of the robot's travel and the turning rate of the robot's body. The robot was also able to perform more complex tasks such as balancing a pendulum on its back while walking and tracking an object.

Simulations of the walking robot can be found at:
http://www.ai.mit.edu/projects/leglab/simulations/hexapod/hexapod.html

1.1 Summary of Thesis Contents

This thesis is organized in the following manner:

Chapter 2 gives a brief overview of Virtual Model Control and how it is implemented. This chapter develops a robotic boxer example in order to illustrate Virtual Model Control and its advantages.

Chapter 3 describes the general structure of the hexapod robot. This chapter also gives an overview of how Virtual Model Control was used to enduce the robot to walk.

Chapter 4 explains the mathematics used to relate the generalized virtual forces of the hexapod at the body and the feet to actual joint torques.

Chapter 5 describes the actual simulation of the robot. This chapter also details the development of the generalized virtual forces using virtual components and describes the walking control algorithm.

Chapter 6 presents the results of implementing Virtual Model Control on the hexapod in order to make it walk. This chapter also describes how the robot is able to walk with a pendulum on its back and gives supporting experimental results.

Chapter 7 discusses the lessons learned in the implementation of Virtual Model Control on a Walking Hexapod.

Chapter 2

Virtual Model Control Overview

By developing virtual model controllers, the behavior of the robot can be controlled using only the state of the virtual components and the current state of the robot (i.e. the joint positions and velocities). This approach is much more intuitive than other control schemes which attempt to control the motion of the robot through the specification of individual joint torques or positions in joint space. The use of Virtual Model Control is particularly helpful in controlling complex tasks such as walking, runing, dancing, or swimming. In order to implement virtual model controllers, a complex task must be broken up in to subtasks which are snapshots of the overall task. For example, walking can be divided into two sub tasks, stance (double support) and swing (single support). Once virtual model controllers are developed to control the motion of the robot during these subtasks, a higher level controller can be used to coordinate the virtual model controllers in order to successfully perform the overall task. The number and type of virtual model controllers developed for a given task depends on the complexity of the overall task and the desired complexity of the higher level controller. Choosing the correct subtasks is crucial in the success of the controller. Intuition in the design of these controllers comes with experience in mechanical and robotic design.

2.1 Developing a Virtual Model Controller

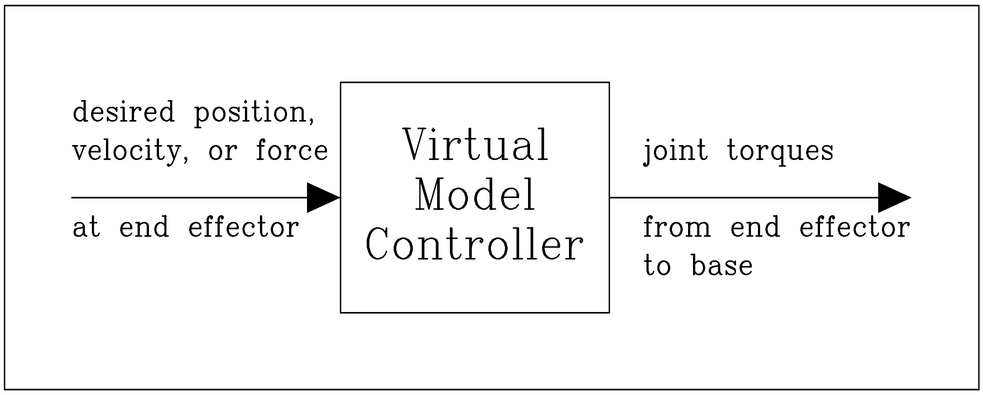

Figure 2-1: Block diagram of simple virtual model controller. This schematic illustrates that given the desired behavior, a position, velocity, or force, of the end effector, a virtual model controller produces the torques on the joints between the end effector and the base.

The formulation and implementation of virtual model controllers are relatively straightforward. Once a subtask is devised the physical base and the end effector of the desired motion must be identified. Again this is a crucial step in the development of an effective controller, and the insight into the proper design comes from experience and intuition. The base of the subtask can be thought of as the part of the robot which remains stationary relative to an external reference frame during the desired motion of the subtask. The end effector is the part of the robot which moves in order to achieve the desired motion. For example, if it is desired that the robot draw a small picture on a chalk board, the end effector would be the hand and the base would be the center of mass. The end effector should be chosen so that the desired task to be controlled can be described by the desired values of the end effector. If developed correctly, virtual model controller given the desired position, velocity, or force at the end effector will output the joint torques required to achieve the motion of a desired task.

Virtual model controllers have two important elements. One element is the virtual components and the other is the virtual model math. Virtual model components are used to translate the desired variables describing the behavior of the end effector into a generalized virtual force acting at the end effector. These virtual model components can be, but are not limited to, linear mechanical elements such as virtual springs and dampers as well as virtual force sources. The choice of which virtual components to use depends on the desired motion of the end effector. This motion dictates the parameters of the virtual components which in turn produce a generalized virtual force at the end effector.

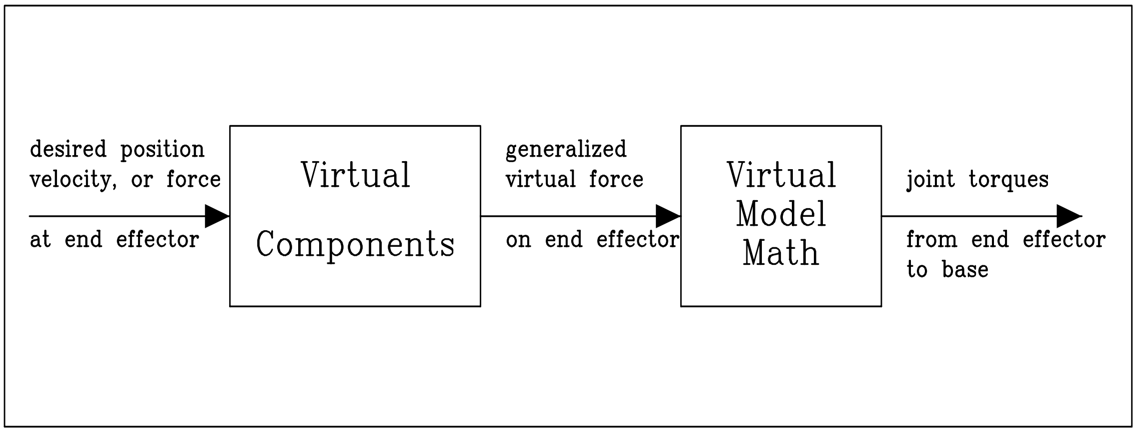

Figure 2-2: This expanded block diagram shows two essential elements in a basic virtual model controller. The virtual components are imaged mechanical components which translate the desired behavior of the robot at the end effector into a generalized force at the end effector. The virtual model math then maps this force to the joint torques between the end effector and base.

The generalized virtual force is then converted to correspond to actual joint torques using virtual model math. The mapping of a generalized virtual force to joints torques is done by deriving the kinematics of the robot starting from the base to the end effector. The Jacobian of the linkage from the base to the end effector is then calculated. The transpose of the Jacobian serves as the mapping between the virtual force and the actual joint torques. The derivation of the virtual model math for a serial linkage is relatively straightforward once the appropriate base and end effector are chosen.

2.2 Robotic Boxer Example

The following example is developed in order to give a better explanation of the formulation and implementation of virtual model controllers. This section suggests a possible control scheme for a robotic boxer. The example presented in this section give an extended explanation of the example mentioned in [5]. Although it is not obvious how each of the joints of the robotic boxer should be moved in order to produce a desired motion, it is more obvious how the robotic boxer should move its fist and body in order to throw a punch and maintain balance. Two virtual model controllers are discussed in this section. One is a controller for a jab and the other is a controller for balance. Figure

2-3 depicts the generalized forces at the fist and body as well as virtual components connected to the fist and body.

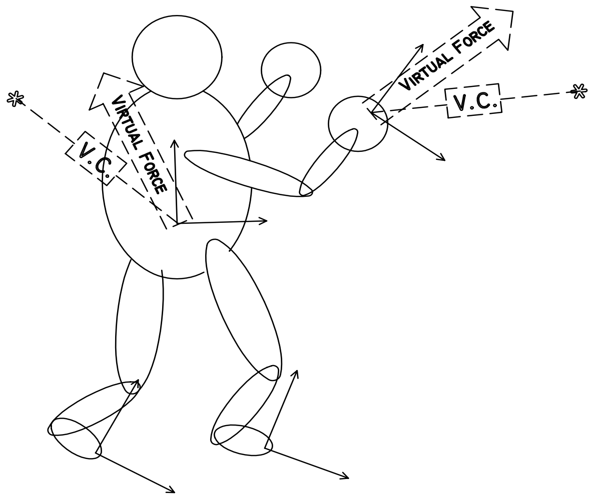

Figure 2-3: Modified version of the robotic boxer example by Pratt. In this diagram, the a generalized virtual force from the body to the fist is shown. A generalized virtual force from the feet to the body is also shown. Generalized virtual components, V.C., which produce the virtual forces, are illustrated as well.

2.2.1 Jab Controller

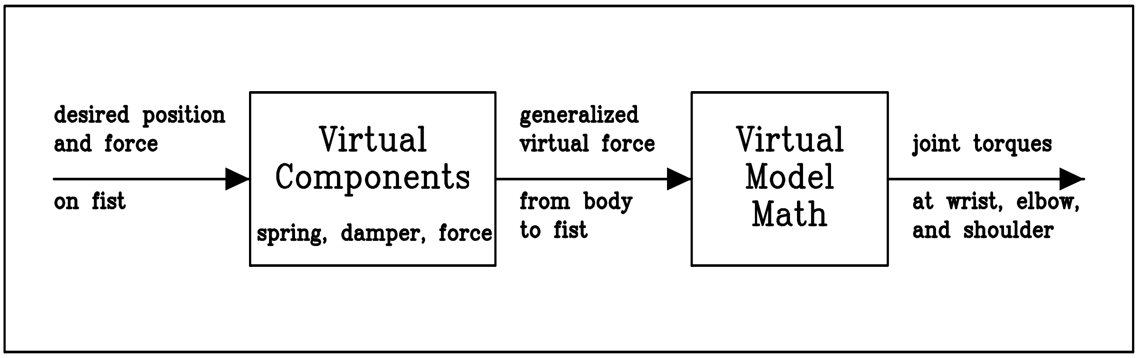

Figure 2-4: Block diagram of the Jab Controller used when a robotic boxer is throwing a punch. This demonstrates that virtual spring and dampers with virtual force sources can be used to create a virtual force on the fist given a desired position and force at the fist. The virtual force at the fist is then translated into the wrist, elbow, and shoulder joint torques through virtual model math.

In order to throw a punch, the motion of the fist, wrist, forearm, elbow, upper arm, shoulder, and body are involved. The purpose of developing a virtual model controller is to abstract away the control of each of these components, especially the joint control. The first step in developing the virtual model controller is to decide the end effector and the base. In this example, the fist is chosen as the end effector because the motion of a punch can be described by the desired motion of the fist. The base is the body. Virtual components are then used to translate the desired variables into a generalized virtual force. Since the boxer wants to hit an opponent's face as hard as possible, the desired variables will be a position and a force. In order to achieve these variables, a virtual spring and damper is used to emulate the force needed to position the fist to opponent's face and a virtual force source is used to emulate the striking force. These components translate the desired variables into a generalized virtual force. This generalized force is then mapped to actual joint torques which in turn produce the desired motion of the fist. The mapping of the generalized force to the joint torques is achieved by deriving the Virtual Model Math from the body to the hand.

2.2.2 Balance Controller

In order to maintain balance while throwing a jab the center of mass of the robotic boxer must be controlled. The robotic boxer uses its legs and feet to position its center of mass and maintain balance. In order to control the center of mass, a control scheme must be developed which, given a desired motion of the body, outputs the appropriate joint torques at the hips, knees, and ankles. As stated in the previous section, the development of this control scheme begins by defining the end effector and the base. In this example, the end effector is the center of mass of the body and the bases are the two feet. This example is different from the previous one because there are two bases instead of one. This adds an extra step in the control scheme.

In order to maintain balance, the center of mass of the body needs to be positioned. Also in order to hold up the body a force equal to the weight of the robot must also be applied. The desired variables for the robotic boxer are then a position and a force. The virtual components used in this control scheme are again a virtual spring and damper and a virtual force source. These virtual components produce the generalized virtual force which would need to be applied to the center of mass of the body in order to achieve the desired variables.

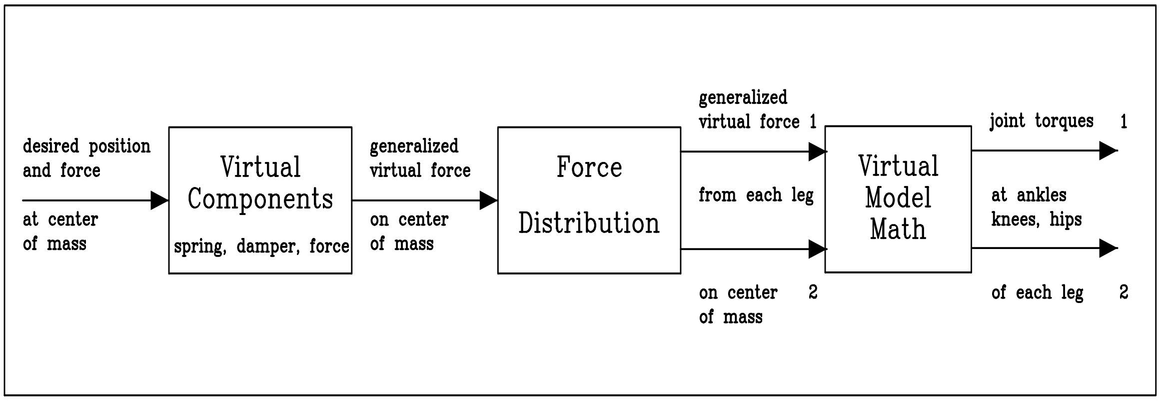

Figure 2-5: Block diagram of the Balance Controller used by the robotic boxer to maintain stability. Virtual components of springs and dampers along with forces sources are used to produce a virtual force on the center of mass given the desired position and force of the center of mass. A force distribution step is added to this controller since the two legs are parallel mechanisms. The virtual force on the body is partitioned into two generalized force contributions from each leg. Each of these forces is then translated into the ankle, knee, and hip joint torques through virtual model math.

Since there are multiple bases in this case, a force distribution operation must be added before the virtual model math can be used to map the virtual force to the actual joint torques. The force distribution operation partitions the generalized virtual force acting on the body into generalized virtual force components. These force components are the contribution of each serial link to the virtual force at the center of mass. Once the generalized force at the body is broken down into the virtual force components of each leg, each virtual force component is mapped to the appropriate joint torques for a given leg using the virtual model math. The virtual model math for each leg is derived using the kinematics and Jacobian from the foot to the center of the body for each serial link.

The development of these two controllers allows for the complete control of a robotic boxer throwing a jab while standing on two feet. Using these two controllers, the design of a higher level control algorithm is simplified since the inputs to the virtual model controllers are intuitive forces and positions. The use of Virtual Model Control is even more powerful when parallel linkages are involved. Often the distribution between parallel linkages is complex when the serial linkages are described in joint space. However, since Virtual Model Control abstracts away the complexity of the joint space and allows behavior to be described using virtual components, a force distribution function can be incorporated into the virtual model controller which partitions this force in to virtual force components from each serial linkage.

Chapter 3

Hexapod Robot Implementation

A hexapod robot was developed in order to test the usefulness of Virtual Model Control. Previous work involving Virtual Model Control includes its implementation on a foot simulation as well as on a planar biped simulation and robot by Pratt[5]. These examples, however, were only two dimensional (2D) implementations. A hexapod was chosen because it is a relatively simple three dimensional (3D) example. Although the hexapod is among one of the simplest 3D examples, its control is not a trivial task. Because of the large number of actuated joints, a control scheme at the joint level would be very complex. Virtual model controllers are used to abstract away this complexity and simplify the control of the hexapod which enables it to walk.

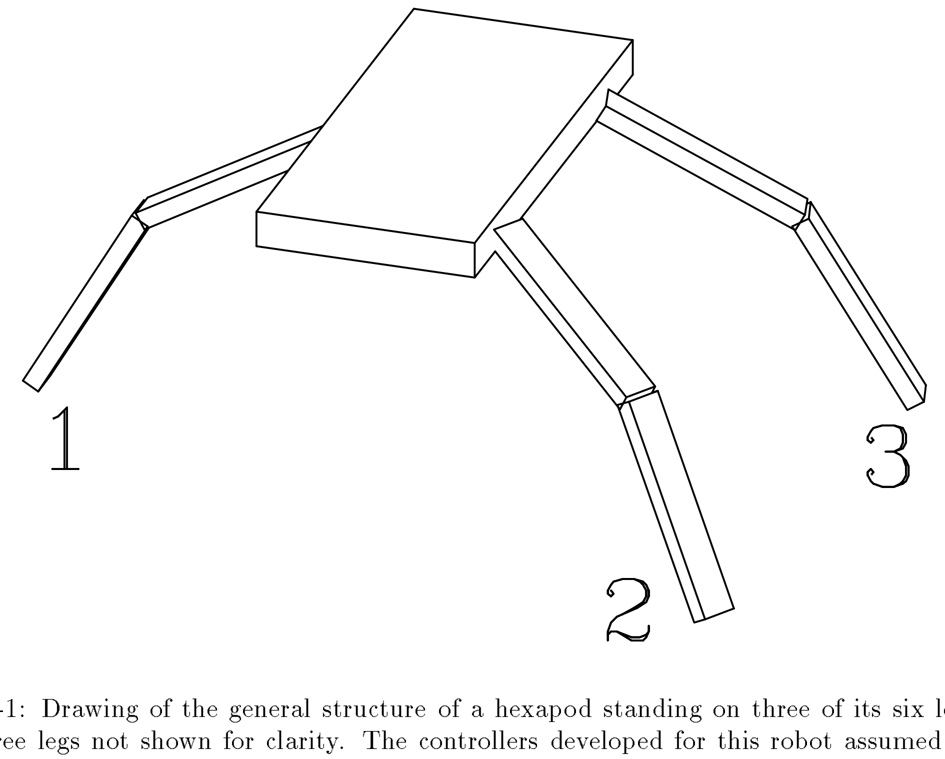

Figure 3-1: Drawing of the general structure of a hexapod standing on three of its six legs. The other three legs not shown for clarity. The controllers developed for this robot assumed that the robot would walk using a tripod gait.

The hexapod robot implements a tripod gait in which it is alternatively supported by a tripod composed of three of its six legs. Using three legs for support allows the hexapod to be statically stable. The hexapod has 18 actuated joints with two degrees of freedom in the hip and one degree of freedom in the knee for each of the legs. The general structure of the hexapod is shown in 3-1.

In order to induce the robot to walk, two virtual model controllers were developed. One controller was used to control the motion of the body while the support tripod was on the ground. The other controller was used to control the motion of the legs in the swing tripod. The next two section give a brief overview of each of the controllers.

3.1 Swing Tripod Controller

This controller is the simpler of the two because the legs are handled as individual serial linkages. The Swing Tripod Controller inputs the desired foot positions for each leg of the swing tripod and outputs the actual joint torques for each joint of the swing tripod. The Swing Tripod Controller can actually be broken down into three smaller controller as seen in Figure 3-2.

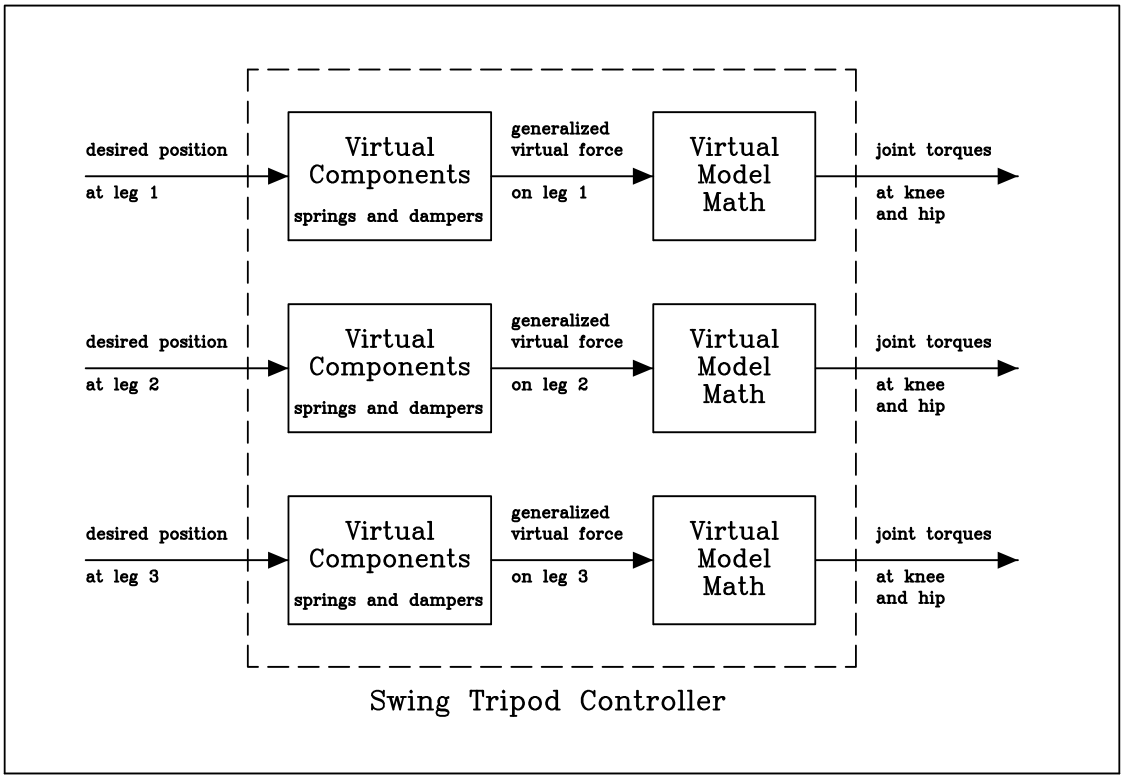

Figure 3-2: Block Diagram of Swing Tripod Controller. This controller is a conglomeration of three smaller virtual model controller. Each of the smaller controllers input the desired position of a given leg and return the joint torques for that leg.

The basic structure of this control scheme can be revealed by looking at one of the swing legs since the structure for each of the smaller controllers is similar. The foot or point at the end of the leg is considered the end effector. The base is the center of the body. In order to move the leg to a desired position a generalized virtual component is attached from the end of the foot to the desired position. Using virtual springs and dampers, a generalized virtual force at the foot is produced.

Using virtual model math derived from the body to the foot, the generalized virtual force is mapped to the actual joint torques of the leg needed to move the leg to its desired position.

Although the task of changing the desired positions of each foot into joint torques is independent between the legs, the Swing Tripod Controller lumps these smaller controllers into one larger controller. Since the motion of the legs are coordinated in the swing tripod, it is sensible to lump the smaller controllers of each leg into one controller which moves all three legs of the tripod.

3.2 Stance Tripod Controller

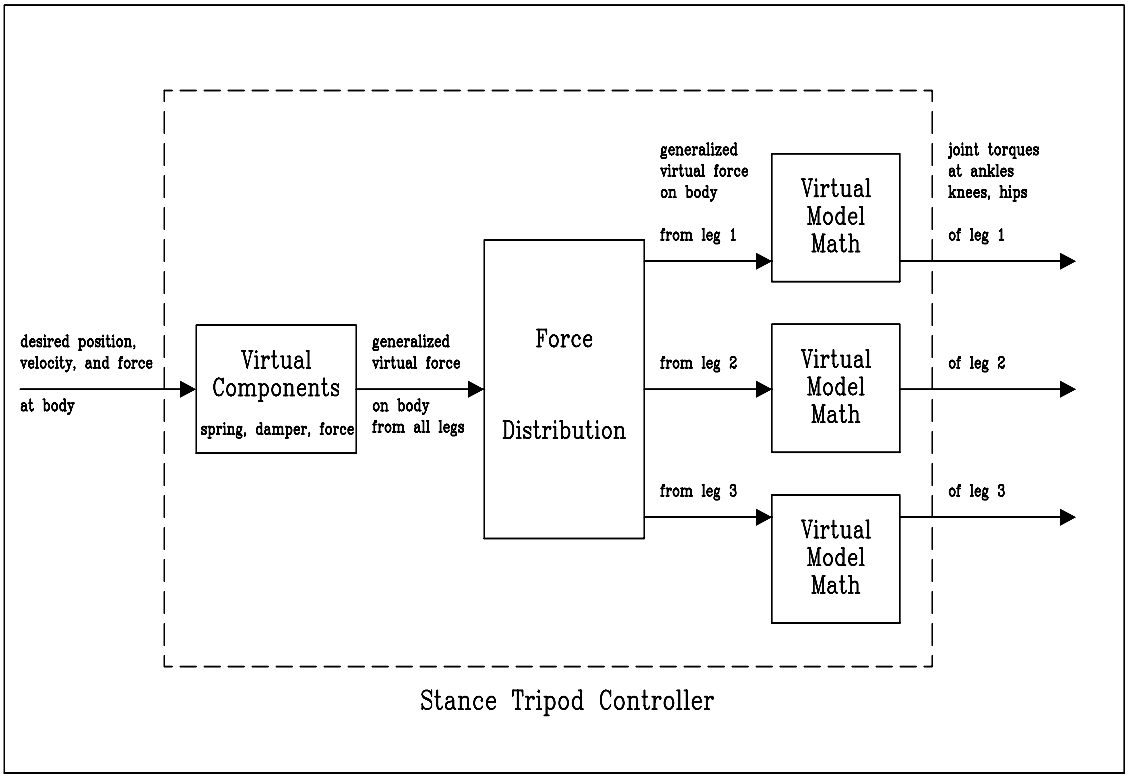

Figure 3-3: Block Diagram of Stance Controller. This virtual model controller in takes the desired position, velocity, and force of the body and produces the joint torques for all three legs of the tripod. Inside the Stance Controller is a force distribution operation which partitions the generalized virtual force on the body in to the virtual force contribution from each leg.

The Stance Tripod Controller is used when all three legs of the tripod are on the ground. During stance the robot advances in a desired manner by moving its body. The motion of the body is dependent on the coordination of the three legs and the joint torques of these three legs. The control and coordination of this task can be rather complicated in joint space due to the linkages between the feet and body. The Stance Tripod Controller induces the motion of the robot body using an intuitive set of inputs, position, velocity, and force. The structure of the Stance Tripod Controller is similar to the Swing Tripod Controller. However, there is an added step due to the fact that parallel linkages contribute to the motion of the body. In this case, the body is considered the end effector. Since each leg is a serial linkage which contributes to the motion of the body, each of the three feet is a base.

In order to move the body given desired variables, virtual components need to be attached from the center of the body to a given desired position. The desired motion of the body was achieved using virtual springs and dampers and force sources. These virtual components produced a virtual force on the body. Before virtual model math can be used to map this generalized virtual force to the joint torques, an added step of force partition is required. The generalized virtual force on the body must be broken up into virtual force components contributed by each leg. This operation is done using an extension of Gardner's Partitioned Force Set [2, 3]. Once the generalized virtual force on the body is partitioned among the other legs, virtual model math is used to map each of the virtual forces to the joint torques of the appropriate leg.

3.3 Overall Control Scheme

Once the controllers for the hexapod were developed, a higher level controller is used above them to switch between the appropriate controllers and set the desired inputs of each in order to induce walking. A simple walking control algorithm was developed which inputs three variables of velocity magnitude, velocity angle, and turning rate of the body and outputs the desired motion of the body and the swing legs. The simplicity of the design of the walking algorithm is contributed to the development of the Swing and Stance Tripod Controllers which abstract away the complexity of the control of each joint, leaving only the motion of the feet and body to be commanded.

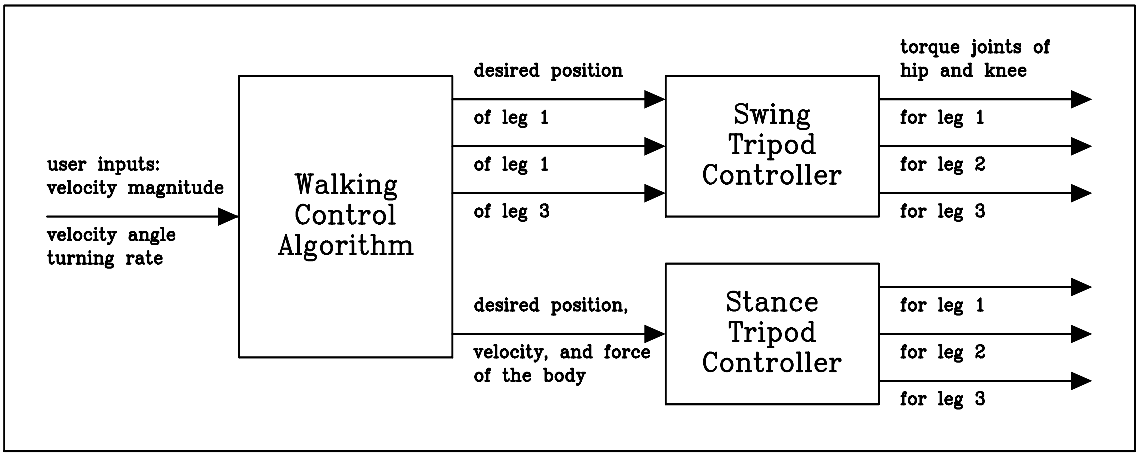

Figure 3-4: Block Diagram of Overall Control Scheme. This schematic demonstrates how the walking control algorithm coordinates the Swing and Stance Tripod Controllers by modulating the inputs into these controllers based on the user inputs of velocity magnitude, velocity direction, and turning rate of the body.

The above figure shows the overall design of the control scheme used to make the hexapod walk. The development of each of the block functions is described in detail in the following sections.

Chapter 4

Hexapod Virtual Model Math

In order to implement the virtual model controllers described in the previous section the virtual model math must be derived. In this chapter, the mathematics needed for the Swing and Stance Tripod Controllers is presented. The method in which the following equations were developed is detailed in Appendix A. This appendix combines work from [5] and [4]. This chapter will only briefly describe how virtual model math is derived for the hexapod.

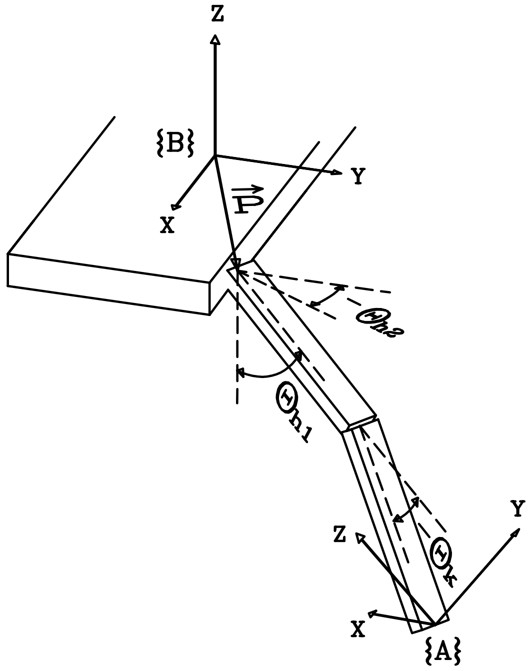

Figure 4-1: Diagram of a single leg of the hexapod example. There is one actuated degree of freedom at the knee and two at the hip. The knee angle is θ_k and the hip angles are θ_{h1} and θ_{h2}. The leg links are both of length L. The location of the leg with respect to frame {B} is (P_x, P_y, 0).

Figure 4-1 shows a close up view of one of the legs. There is one actuated degree of freedom at the knee and two at the hip. The knee angle is θ_k and the hip angles are θ_{h1} and θ_{h2}. The leg links are both of length L. The location of the leg with respect to frame {B} is (P_x, P_y, 0). {A}is a reference frame at the foot and {B} is at the center of the body. These specifications are needed in order to derive the virtual model math for the robot. As stated previously, virtual model math is a function which maps the generalized virtual force on a part of the robot to the joint torques needed to move that part.

4.1 Swing Tripod Math

In the Swing Tripod Controller, we wish to map the generalized virtual forces on the feet of the swing tripod to the joint torques of the legs. Since the legs are independent serial linkages, the virtual model math for each linkage can be derived separately. Since the body is the base and the foot is the end effector, the virtual model math must be developed from the body to the foot. The generalized virtual force ${}^{B}({}^{B}_{A}F_i)$ at leg i is generated from frame {B} to {A} and is with respect to {B}. The mathematics derived here is generalized so that it can be used for any leg i of the swing tripod.

The generalized force ${}^{B}({}^{B}_{A}F_i)$ has three components f_{x_i}, f_{y_i}, f_{z_i}. This force is mapped to the joint torques τ_{k_i}, τ_{h1_i}, τ_{h2_i} of leg i using the Jacobian transpose J^T from {B} to $\{A_i\}$. The virtual force to joint torque relationship is given by,

$$\begin{bmatrix} \tau_{k_i} \\ \tau_{h1_i} \\ \tau_{h2_i} \end{bmatrix} = J^T \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \end{bmatrix} \tag{4.1}$$

The Jacobian transpose is found by extracting elements from the Jacobian from {B} to $\{A_i\}$. The Jacobian transpose ${}^{B}({}^{B}_{A_i}J)$ is determined via an iterative computation of the joint velocity to cartesian velocity mapping, as described in Craig[1],

$$J = \begin{bmatrix} -L\,s_{h2_i}(c_{h1_i} - c_{h1_i}c_{k_i} + s_{h1_i}s_{k_i}) & -L\,c_{h2_i}(s_{h1_i} - s_{h1_i}c_{k_i} - c_{h1_i}s_{k_i}) & L\,s_{h2_i}(s_{h1_i}s_{k_i} - c_{h1_i}c_{k_i}) \\ L\,c_{h2_i}(c_{h1_i} + c_{h1_i}c_{k_i} - s_{h1_i}s_{k_i}) & -L\,s_{h2_i}(s_{h1_i} - s_{h1_i}c_{k_i} - c_{h1_i}s_{k_i}) & -L\,c_{h2_i}(s_{h1_i}s_{k_i} + c_{h1_i}c_{k_i}) \\ L\,(s_{h1_i}c_{k_i} + c_{h1_i}s_{k_i} + s_{h1_i}) & 0 & L\,(c_{h1_i}s_{k_i} + s_{h1_i}c_{k_i}) \end{bmatrix} \tag{4.2}$$

where $c_x = \cos(\theta_x)$ and $s_x = \sin(\theta_x)$.

4.2 Stance Tripod Math

The virtual model math in the Stance Tripod Controller maps the virtual force on the body to the joint torques of the three supporting legs. In order to map the virtual force on the body to the joint torques of each leg, an added step of force distribution must be performed. Because the three legs are parallel linkages working together to move the body, the generalized force on the body is partitioned into generalized force components. Each force component is a generalized virtual force contribution from a given leg to the body. This generalized virtual force for a given leg can then be mapped to the joint torques of that leg.

4.2.1 Virtual Force to Joint Torque Mapping

The generalized force on the body from the three legs of the stance tripod is ${}^{B}({}^{A^*}_{B}F)$. A^* denotes the multiple bases of the feet from which the force on the body at {B} are applied. Since we are concerned about the relative rotation between the feet and the body, the generalized force ${}^{B}({}^{A^*}_{B}F)$ has six components which are the forces and torques on the body in the x, y, and z directions.

$${}^{B}({}^{A^*}_{B}F) = \begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix} \tag{4.3}$$

After partitioning this force, we have three generalized forces ${}^{B}({}^{A^i}_{B}F_i)$ which also have 6 components which are the force and torque contributions from leg i to the body in the x, y, and z directions.

$$
{}^{B}({}^{A^i}_{B}F_i) = \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \\ n_{x_i} \\ n_{y_i} \\ n_{z_i} \end{bmatrix} \tag{4.4}
$$

The virtual force ${}^{B}({}^{A^i}_{B}F_i)$ from the ith leg to the body is mapped to the joint torques of the ith leg using the relationship,

$$
\begin{bmatrix} \tau_{k_i} \\ \tau_{h1_i} \\ \tau_{h2_i} \end{bmatrix} = J^T \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \\ n_{x_i} \\ n_{y_i} \\ n_{z_i} \end{bmatrix} \tag{4.5}
$$

J^T is the Jacobian transpose from the foot to the body. The Jacobian from the foot to the body is found using methods detailed in Craig[1]. The Jacobian from the foot to the body is

$$
J = \begin{bmatrix} -L\, s_{h2_i} c_{h1_i} & 0 & -P_{y_i} \\ L\, c_{h2_i} c_{h1_i} & 0 & +P_{x_i} \\ L\, s_{h1_i} - P_{x_i}\, s_{h2_i} + P_{y_i}\, c_{h2_i} & -P_{x_i}\, s_{h2_i} + P_{y_i}\, c_{h2_i} & 0 \\ -c_{h2_i} & -c_{h2_i} & 0 \\ -s_{h2_i} & -s_{h2_i} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.6}
$$

where $c_x = \cos(\theta_x)$ and $s_x = \sin(\theta_x)$.

4.2.2 Force Distribution Method

In order to partition the generalized force ${}^{B}({}^{A^*}_{B}F)$ into three generalized force contributions ${}^{B}({}^{A^i}_{B}F_i)$ from each leg, Pratt's Force Distribution Method [4, 5] was employed. This method is an extension of Gardner's Force Distribution Method [2, 3] and is detailed in Appendix A.

Since no moments can be applied at the point feet, each leg has three natural constraints. Following the method in Section A.5.1 we have

$$
n = 6,\ p = 3,\ l = 3,\ d = 3,\ r = 3 \tag{4.7}
$$

To reduce the size of the matrix to be inverted, the individual components of the virtual forces are partitioned into the Minimum Force Set (6 elements), Redundant Force Set (3 elements), and Constrained Force Set (9 elements). For 3 design constraints, the horizontal forces of legs 2 and 3 are matched and the lateral force of leg 1 is matched with the sum of the lateral forces of legs 2 and 3,

$$
f_{x3} = f_{x2},\ \ f_{y3} = f_{y2},\ \ f_{y1} = 2f_{y2} \tag{4.8}
$$

These design constraints are written in the terms of Equation A.16 if the virtual forces are partitioned as follows,

$$
\begin{aligned}
MFS &= \{f_{x1}, f_{z1}, f_{x2}, f_{y2}, f_{z2}, f_{z3}\} \\
RFS &= \{f_{x3}, f_{y3}, f_{y1}\} \\
CFS &= \{n_{x1}, n_{y1}, n_{z1}, n_{x2}, n_{y2}, n_{z2}, n_{x3}, n_{y3}, n_{z3}\}
\end{aligned}
$$

The terms of the constraint equation (A.11) are now,

$$
f^a = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}, \quad f^b = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
$$

$$
D^{1a} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad D^{2a} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & -2 & 0 \end{bmatrix}, \quad D^{3a} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
$$

$$
D^{1b} = D^{2b} = D^{3b} = 0 \tag{4.9}
$$

The elements of the natural constraints, J^{ia} and J^{ib} are extracted from the Jacobian Transpose. The Minimum and Redundant Force Sets are acted on by J^{ia} where

$$
\begin{aligned}
[J^{ia}]_{1,1} &= -Ls_{h2_i}(c_{k_i+h1_i} + c_{h1_i}) \\
[J^{ia}]_{1,2} &= Lc_{h2_i}(c_{k_i+h1_i} + c_{h1_i}) \\
[J^{ia}]_{1,3} &= Ls_{k_i+h1_i} + Ls_{h1_i} + P_{y_i} c_{h2_i} - P_{x_i} s_{h2_i} \\
[J^{ia}]_{2,1} &= -Lc_{h2_i}(1 + c_{k_i}) - P_{y_i} s_{k_i+h1_i} \\
[J^{ia}]_{2,2} &= -Ls_{h2_i}(1 + c_{k_i}) + P_{x_i} s_{k_i+h1_i} \\
[J^{ia}]_{2,3} &= -P_{y_i} s_{h2_i} c_{k_i+h1_i} - P_{x_i} c_{h2_i} c_{k_i+h1_i} \\
[J^{ia}]_{3,1} &= Lc_{h2_i} s_{k_i} - P_{y_i} c_{k_i+h1_i} \\
[J^{ia}]_{3,2} &= Ls_{h2_i} s_{k_i} + P_{x_i} c_{k_i+h1_i} \\
[J^{ia}]_{3,3} &= P_{y_i} s_{h2_i} s_{k_i+h1_i} + P_{x_i} c_{h2_i} s_{k_i+h1_i}
\end{aligned}
$$

while the Constrained Force Set is acted on by J^{ib} where

$$
(J^{ib}) = \begin{bmatrix} -c_{h2_i} & -s_{h2_i} & 0 \\ s_{h2_i} c_{k_i+h1_i} & -c_{h2_i} c_{k_i+h1_i} & -s_{k_i+h1_i} \\ -s_{h2_i} s_{k_i+h1_i} & c_{h2_i} s_{k_i+h1_i} & -c_{k_i+h1_i} \end{bmatrix} \tag{4.10}
$$

The first six rows of the constraint matrix state that the sum of the virtual force and moment vectors for each of the individual legs equals the total virtual force and moment vector acting on the body at frame {B}. The next three sets of three rows define the natural constraints of zero torque at each foot. The last three rows are the design constraints chosen in Equation 4.8.

Note that the angles the foot makes with frame $\{A_i\}$ do not appear in any of the previous equations. In order to derive these equations, X-Y-Z Euler angles were used at the feet. The foot angles did not appear in the torque-force relationship (Equation 4.5). However, they did appear in the natural constraints, J^{ia} and J^{ib}. The natural constraint equations define a 3 dimensional subspace of the 6 dimensional virtual force space. This subspace is the space in which "admissible" virtual forces can be applied. It was verified that this subspace is the same for any foot angles. Therefore, the foot angles were arbitrarily set to zero.

An intuitive explanation for why the foot angles do not matter is that virtual forces are being applied from frame $\{A_i\}$ to frame {B} *with respect to frame* $\{B\}$. How we define frame $\{A_i\}$ therefore doesn't matter, as long as we can specify the foot angles in some way. Therefore, it is arbitrary what the foot angles are and we can set them to zero in order to eliminate them from our equations.

The submatrix, J^{ib}, happens to be orthonormal so that its inverse is simply its transpose,

$$
(J^{ib})^{-1} = (J^{ib})^T \tag{4.11}
$$

The Constrained Force Set is eliminated to get the following elements in Equation A.14,

$$
S^i = D^{ia}, \quad u_l = 0, \quad v_r = 0 \tag{4.12}
$$

$$
\begin{aligned}
[Q^i]_{1,1} &= 0 \\
[Q^i]_{1,2} &= Lc_{k_i+h1_i} + Lc_{h1_i} \\
[Q^i]_{1,3} &= P_{y_i} + Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) \\
[Q^i]_{2,1} &= -L(c_{k_i+h1_i} + c_{h1_i}) \\
[Q^i]_{2,2} &= 0 \\
[Q^i]_{2,3} &= Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{x_i} \\
[Q^i]_{3,1} &= -Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{y_i} \\
[Q^i]_{3,2} &= -Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) + P_{x_i} \\
[Q^i]_{3,3} &= 0
\end{aligned}
$$

The Redundant Force Set is eliminated, as in Equations A.17 and A.18, to get

$$
\begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & s_{4,2} & 0 & s_{4,4} & s_{4,5} & s_{4,6} \\ s_{5,1} & s_{5,2} & s_{5,3} & 0 & s_{5,5} & s_{5,6} \\ s_{6,1} & 0 & s_{6,3} & s_{6,4} & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{x1} \\ f_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ f_{z3} \end{bmatrix} \tag{4.13}
$$

where,

$$
\begin{aligned}
s_{4,2} &= P_{y_1} + Lc_{h2_1}(s_{h1_1+k_1} + s_{h1_1}) \\
s_{4,4} &= -2L\, s_{h1_1}s_{k_1} - Ls_{h1_2}s_{k_2} - Ls_{h1_3}s_{k_3} \\
&\quad +Lc_{h1_2}(c_{k_2}+1) + Lc_{h1_3}(c_{k_3}+1) + 2L\, c_{h1_1}(c_{k_1}+1) \\
s_{4,5} &= P_{y_2} + Lc_{h2_2}(s_{h1_2+k_2} + s_{h1_2}) \\
s_{4,6} &= P_{y_3} + Lc_{h2_3}(s_{h1_3+k_3} + s_{h1_3}) \\
s_{5,1} &= -Lc_{h1_1+k_1} - Lc_{h1_1} \\
s_{5,2} &= -P_{x_1} + Ls_{h2_1}(s_{h1_1+k_1} + s_{h1_1}) \\
s_{5,3} &= Lc_{h1_2}(-c_{k_2}-1) + Lc_{h1_3}(-c_{k_3}-1) \\
&\quad +Ls_{h1_2}s_{k_2} + Ls_{h1_3}s_{k_3} \\
s_{5,5} &= -P_{x_2} + Ls_{h2_2}(s_{h1_2}(c_{k_2}+1) + c_{h1_2}s_{k_2}) \\
s_{5,6} &= -P_{x_3} + Ls_{h2_3}(s_{h1_3+k_3} + s_{h1_3}) \\
s_{6,1} &= -P_{y_1} - Lc_{h2_1}(s_{h1_1+k_1} + s_{h1_1}) \\
s_{6,3} &= -P_{y_2} - P_{y_3} - Lc_{h2_2}(s_{h1_2+k_2} + s_{h1_2}) \\
&\quad -Lc_{h2_3}(s_{h1_3+k_3} + s_{h1_3}) \\
s_{6,4} &= 2P_{x_1} + P_{x_2} + P_{x_3} - 2Ls_{h2_1}(s_{h1_1+k_1} + s_{h1_1}) \\
&\quad -Ls_{h2_2}(s_{h1_2+k_2} + s_{h1_2}) - Ls_{h2_3}(s_{h1_3+k_3} + s_{h1_3})
\end{aligned}
$$

To solve for the Minimum Force Set, an inversion of the 6×6 matrix in Equation 4.13 is needed. The design constraints (Equation 4.8) are then used to solve for the Redundant Force Set in terms of the Minimum Force Set. Equation A.12 is used to solve for the Constrained Force Set in terms of the Minimum and Redundant Force Sets and finally Equation 4.5 is used to compute the joint torques.

Chapter 5

Hexapod Walking Robot

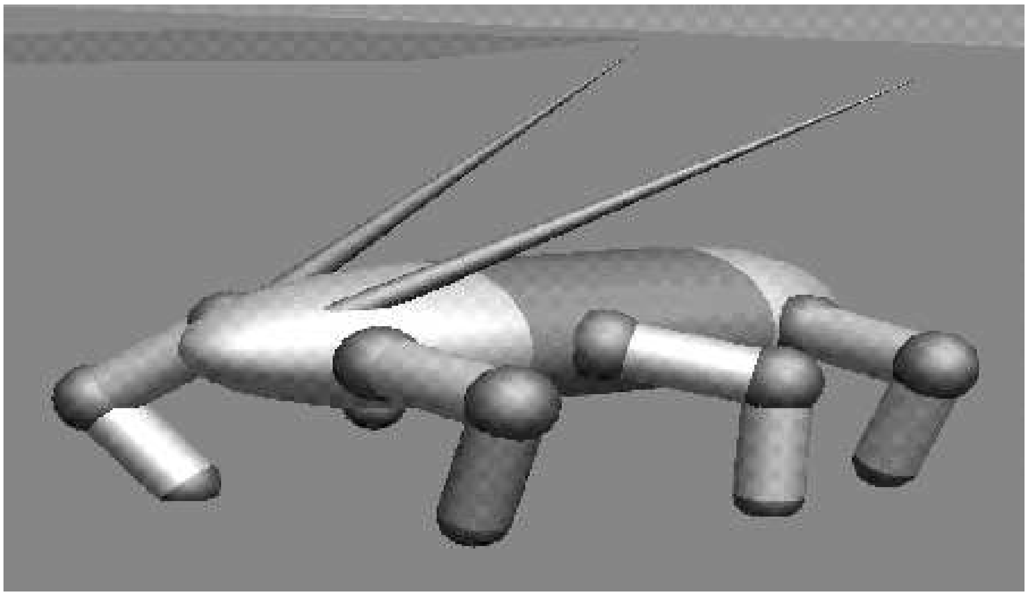

Figure 5-1: Graphical rendering of the hexapod robot simulation. The hexapod was simulated using software which emulates physically based dynamics.

A hexapod robot was built in simulation in order to implement Virtual Model Control. The mathematics described in the previous section was used in order to develop the virtual model controllers. The virtual model controllers were completed by implementing virtual components in the simulation. Once the development of the virtual model controllers was complete a high level walking algorithm was developed in order to regulate the virtual model controllers and induce walking.

The goal of the simulation was to achieve stable walking in a variety of different patterns. Stable walking was defined as the successful periodic cycle from one tripod to the other in order to move the hexapod. The only input to the hexapod is the speed and direction it should travel as well as the turning rate about the z axis.

5.1 Hexapod Simulation

A six-legged robot was created using simulation software which emulates physically based dynamics. The simulated hexapod has a total mass of approximately 3.8 kg. The body of the hexapod is a teardrop ellipsoid with a total length of 45.0 cm and width of 8.0 cm. Each leg is made of two links and has a total mass of 0.07 kg. Each leg link is 10.0 cm in length. At the hip of each leg, there is a gimble joint in the x and z directions. At the knee, there is a pin joint in the x direction. The body reference frame is set up so that the x axis points in a direction from the center of mass to the head, the y axis points to the robot's left side, and the z axis points to the sky.

The simulator was able to measure the position and velocity of the hexapod's body. The simulator also outputted the angle and angular velocity of all the joint angles. The position of each foot and the information stating whether it was contacting the ground was also provided by the simulator. These variables are mentioned because they were used in the control of the hexapod. If the hexapod were to actually be built, appropriate sensor technology would have to be employed in order to obtain this information. Such sensors might include gyroscopes, accelerators, and foot switches.

5.2 Virtual Components

As discussed previously, two virtual model controllers were developed in order to simplify the control of the hexapod. The next two sections describe the virtual components used in the simulation in order to complete the virtual model controllers. The various spring, damper, and force constants presented in the next two sections were chosen using design experience and insight. These were also found through trial and error in order to generate the desired behavior of the walking hexapod.

5.2.1 Stance Controller Virtual Components

Once the robot was "built" in simulation, virtual components were implemented in order to control the motion of the body. The virtual components were chosen based on the desired motion of the body. The virtual components input the desired motion of the body and produce the generalized virtual force on the body. The virtual components serve as simple linear controllers which regulate the motion of the body.

The following equations demonstrate the simple linear virtual components which were used to regulate the motion of the body in stance:

$$\begin{aligned}
F_x &= b_x(\dot{x}_d - \dot{x}_{act}) \\
F_y &= b_y(\dot{y}_d - \dot{y}_{act}) \\
F_z &= k_z(z_d - z_act) - b_z\dot{z}_{act} + f_{weight} \\
T_x &= k_{roll}(roll_d - roll_{act}) - b_{roll}\dot{roll}_{act} \\
T_y &= k_{pitch}(pitch_d - pitch_{act}) - b_{pitch}\dot{pitch}_{act} \\
T_z &= b_{yaw}(\dot{yaw}_d - \dot{yaw}_{act})
\end{aligned} \tag{5.1}$$

On the left are the six components of the generalized force vector on the body. The expressions to the right represent the relationship between the desired motion of the body, the state of the robot, and the virtual components. The first three equations are the virtual forces in the x, y, and z directions and the last three are the virtual torques in roll, pitch, and yaw directions.

Since the desired motion of the robot in the x and y directions is dictated by desired velocities, the x and y virtual forces are emulated using virtual dampers (velocity control). The virtual force in the x direction, F_x, is produced by taking the difference between the desired velocity in the x direction, $\dot{x}_d$, and the actual velocity in the x direction, $\dot{x}_{act}$, and multiplying it by a damping coefficient, b_x. The virtual force in the y direction F_y is produced in the same fashion.

During walking the height of the body should remain off the ground at a given height. In order to achieve this, the virtual force in the z direction is comprised of a number of virtual components.

The first is a virtual spring and damper which is used to set the height of the body. In this case, the desired z position of the hexapod was set to 15.0 cm. A virtual force, f_{weight}, was also added in the z direction in order to combat the weight of the robot. This virtual force component was set to 32.0 N, a value slightly below the weight of the hexapod.

During walking minimal motion in the roll and pitch direction is desired. The position of the body in these directions should be stable at zero during walking. In order to achieve this, the virtual torques in the roll and pitch directions are produced using virtual springs and dampers. These PD controllers are used to maintain a desired position in the roll and pitch directions.

In order to changed the facing of the robot, an angular velocity about the z axis is specified. The virtual torque, τ_x is produced using a virtual damper which takes the difference between the desired yaw angular velocity, $y\dot{a}w_d$, and the actual yaw angular velocity, $y\dot{a}w_d$, and multiplies it by a damping coefficient, b_{yaw}.

5.2.2 Swing Controller Virtual Components

The virtual components used to move the legs in the swing tripod produce a virtual force in each leg given the desired positions of the legs. Because it is desired to command the position of the foot of each leg during swing, virtual springs and dampers are implemented. Again simple linear controllers are used to regulate the motion of the legs.

The following equations implement the virtual spring and damper components which were used to control the motion of the legs during swing:

$$\begin{aligned} F_{x_i} &= k_{leg}(x_{d_i} - x_{i_{act}}) - b_{leg}(\dot{x}_{i_{act}}) \\ F_{y_i} &= k_{leg}(y_{d_i} - y_{i_{act}}) - b_{leg}(\dot{y}_{i_{act}}) \\ F_{z_i} &= k_{leg}(z_{d_i} - z_{i_{act}}) - b_{leg}(\dot{z}_{i_{act}}) \end{aligned} \tag{5.2}$$

On the left side are the three components of the generalized force vector on any leg i. The expression of the right side illustrate the relationship between the desired position of the of leg i, the actual state of leg i, and the virtual components. These three equations compute the virtual forces on leg i in the x, y, and z directions. There are only three virtual force components in the generalized force on the foot because only the position of the foot is important and the relative rotation of the foot with respect to the body can be disregarded. The virtual force on a foot in the x direction is given by the difference between the desired, x_{i_d}, and the actual, $x_{i_{act}}$, position multiplied by a gain coefficient, k_{leg}, minus the product of a damping coefficient, b_{leg}, and the actual velocity, $\dot{x}_{act}$, of the foot in the x direction. The virtual forces in the y and z directions on the foot are produced in the same manner. Given the desired x, y, and z positions of each of the feet in the swing tripod, the virtual components in the swing controller produce a generalized virtual force acting on each foot from the body.

5.3 Walking Control Algorithm

Having developed the virtual model math and the virtual model components for the swing and stance tripods, virtual controllers could be implemented in order to move tripods A and B. Tripod A consists of the legs shown in Figure 3-1 and tripod B consists of the remaining legs. As stated previously these controllers are able to direct the motion of the body in stance and the legs in swing given a set of desired velocities and positions. In order to achieve stable walking, the proper coordination of tripods A and B is needed. A walking control algorithm was developed in order to switch the controllers applied to tripods A and B and to give these controllers the appropriate input based on the three user inputs. Figure 5-3 depicts the walking control algorithm as a state machine.

In order to coordinate the motion of the three legs of the swing tripod, a virtual pod leg was developed. The virtual pod leg is located at the centroid of the triangle which is created when lines are drawn to connect the three feet of the tripod. The virtual pod is used to determine the desired positions for the feet of each of the three legs. Once the desired position of the the virtual pod foot is calculated, the desired positions of the three individual feet can be calculated since they are always a given offset away from the virtual pod foot. The virtual pod is also used to determine the relative distance of each tripod to the center of mass of the body. This information is used to trigger transitions out of some states in the walking algorithm.

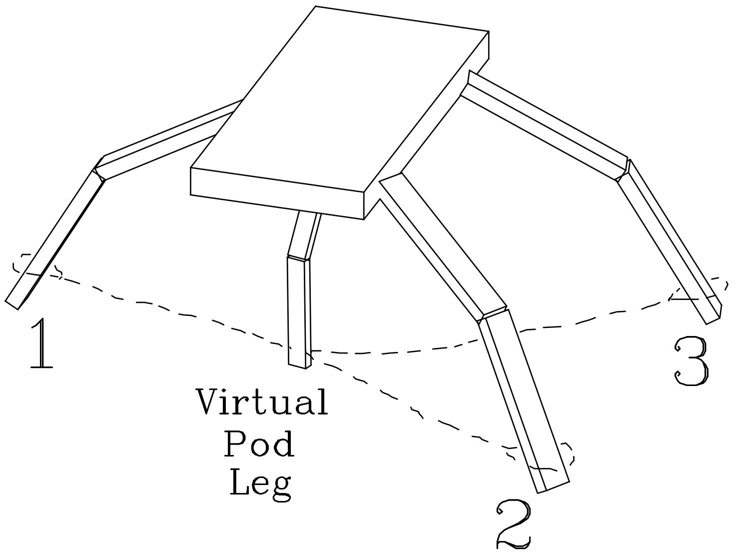

Figure 5-2: Illustration of the virtual pod leg used to position the swing legs and to determine the relative distance between the tripods and the center of mass. The virtual pod is located at the centroid of the triangle drawn by connecting the legs of the tripod. The location of the three legs in a tripod are always a fixed offset from that tripod's virtual pod leg.

In *Double Support*, tripods A and B are both controlled using the Stance Controller. In this state, the walking control algorithm calculates the desired x and y velocities of the body based on the user inputs of robot speed and direction of travel. The virtual force on the body is divided between tripods A and B. The transition from the *Double Support* state to single support on either leg is dependent on the relative distance between the center of mass of the body and the virtual pod foot of each tripod. As the body is moving on its given trajectory, the distance between the center of mass of the body and the projection of the virtual pod position onto that trajectory is calculated for both virtual pods A and B. When one of these distances reaches a threshold value, the virtual pod which is further away is lifted up.

For example, if the desired motion of the robot is away from the virtual pod leg B, the distance between the virtual pod leg B and the center of mass of the body would reach the lift threshold before the virtual pod leg A. The robot would therefore lift tripod B. The next actions of the robot would follow the right lobe of the walking state machine in Figure 5-3

After this transition, tripod A continues to support the body while tripod B is lifted off the ground. In the next three states, the supporting tripod A continues to move the body on the given trajectory while the swing tripod B moves through the air. Throughout these three states tripod A is controlled by the Stance Controller and tripod B is controlled by the Swing Controller. The walking control algorithm inputs desired x and y velocities, and turning rate to the stance controller based on the user inputs V_{mag}, V_{ang}, and Ω.

The desired motion of the swing tripod B is dictated by the motion of the virtual pod leg in the next three states. The motion of the virtual pod leg is set in each state and the walking algorithm calculates the desired x, y, and z positions of the actual legs in the swing tripod. In the *B Lift* state, the virtual pod is positioned to a given height. Once all the legs of tripod B are off the ground, the walking state machine transitions into the next state, *Swing B*. In this state, the virtual pod leg of the swing tripod B moves along the given walking trajectory while mirroring the virtual pod leg of the support tripod A. When the virtual pod foot of the swing tripod B is a specified down threshold distance away from the body's center of mass, the walking algorithm transitions in to the *B Down* state.

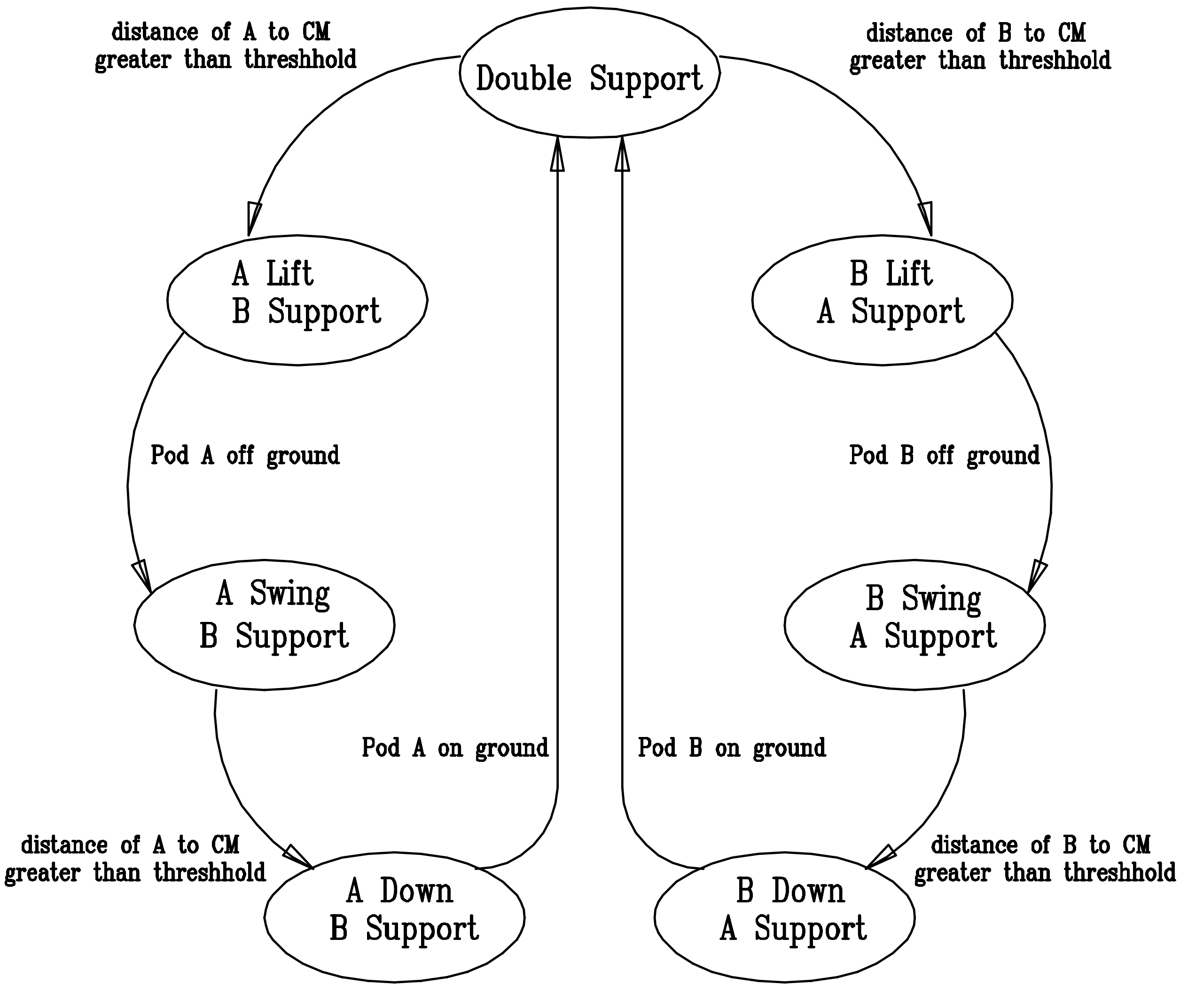

Figure 5-3: Walking algorithm used to control the robot. Each bubble is a state in which the virtual controllers were used to accomplish the task written in the bubble. The arrows are the transitions between each of these states. The trigger event for each transition is listed beside the arrows. In stable walking the robot continuously cycles from double support to single support while switch tripods A and B.

In the *B Down* state, the virtual pod foot of the swing tripod B is positioned on the ground. The support tripod A still continues to move the body. Once ground contact has been made by all the foot switches in the swing tripod B, the state machine returns to *Double Support*. Once in *Double Support*, the force required to move the body is slowly transferred from the support tripod to the swing tripod until each are sharing an equal load. The left lobe of the walking state machine mirrors the right lobe with the roles of tripods A and B switched. In stable walking, the hexapod periodically cycles through this state machine by altering between the right and left lobes.

Chapter 6

Experimental Results

Using virtual model controllers allowed a straightforward implementation of a relatively simple walking control algorithm. This setup facilitated the use of a minimal set of high level user inputs which commanded the robot to successfully walk different paths. The hexapod is able to trace out complex walking paths by specifying the magnitude and angle of the body velocity, V_{mag} and V_{ang}, and the turn rate of the body, Ω. Examples of various walking patterns given different user inputs is shown in Figure 6-1.

For straight line walking, the hexapod is able to walk at speeds up to 0.8 m/s. Figure 6-2 shows experimental data from the hexapod while walking in a straight line. In this case the only user input is a body speed and the other two inputs of velocity angle and yaw velocity are zero. The data show that the robot behaves in a desired fashion. From its starting position, the robot walks in a straight line. The speed of the robot increases in the same fashion as the user inputs velocity steps of 0.2 m/s, 0.4 m/s, and 0.5 m/s. The three graphs on the left of speed, angle, and turn rate show that the robot moves in a manner which is desired by the user with minimal fluctuations. The right three graphs show that the robot behaves appropriately in the z, roll, and pitch directions as well. The final graph in Figure 6-2 shows that the robot achieves stable walking by periodically cycling through its walk states. The minor fluctuations in the motion of the hexapod come from the discontinuities of each step. As the robot picks up and puts down its legs, slight disturbances in the walking pattern are attributed to legs colliding with the ground, shifts in center of mass during swing, and inertial forces during motion.

Having successfully achieved stable walking in a straight line, other more complicated patterns were developed in simulation. Figure 6-3 shows data from the hexapod walking in a circle. In this simulation, the robot traverses a circle while always facing the center of the circle. In order to achieve this motion, the user input V_{mag} is set to 0.2m/s, V_{ang} is 1.57 radians, and Ω is 0.3 radians. As the robot walks around the circle, it tracks the center of the circle so that it is always facing the center. In the data only a partial arc of the circle is shown for clarity. In order to walk the given path, the hexapod only stays in double support, state 0, for a split second. Although this is unusual, it is still able to maintain a stable walking pattern.

It is interesting to note that when the robot begins to walk in this simulation its steps are irregular until about 2 seconds. At this time, the robot recovers from its missed steps and continues on its path in a stable walking manner. In simulation when the robot is operating perfectly, it will continue to operate perfectly until a disturbance is encountered. In this case, the robot started in a non-ideal state from its desired path of motion. Although the robot stumbles a bit at the beginning of the simulation, it is able to successfully recover into a stable walking pattern.

6.1 Pendulum Walking

Virtual Model Control was also used to balance a pendulum on the back of the hexapod while walking. A pendulum of length 30 cm and mass 0.18 kg was attached to the hexapod via a pin joint which rotates in the y axis as seen in Figure 6-4. A simple LQR controller was implemented along with the virtual components in order to regulate the movement of the hexapod in the x direction and to stablize the pendulum. The equation which relates the virtual force in the x direction, F_x, to the desired position of the pendulum, the desired velocity of the bug, the state of the pendulum and the bug, and the virtual components, is

$$F_x = k_x(x_d - x_{act}) + b_x(\dot{x}_d - \dot{x}) + k1_{pend}(-x_{pend}) + k2_{pend} * (-\dot{x}_{pend}) \tag{6.1}$$

where

$$\begin{aligned} x_{pend} &= L_{pend} sin(x_{pend} + pitch) \\ \dot{x}_{pend} &= L_{pend} cos(x_{pend} + pitch)(\dot{x}_{pend} + pi\dot{t}ch) \end{aligned}$$

The preliminary gains of these components were found using a mathematical computation software, Matlab. These gains were then changed through an iterative process of trial and error in order to achieve desired performance.

Figure 6-5 shows data from a simulation in which the hexapod walked in a straight line. The input to the robot was V_{mag} set to 0.2 m/s with V_{ang} and Ω set to zero. The pendulum and hexapod exhibit the typical response of a non-minimum phase system. The robot begins by traveling in a direction opposite to the desired velocity in order to have the pendulum fall in the direction of the desired velocity. The robot then changes its velocity direction and takes approximately 1 second to ramp to the 0.4 m/s. After an initial peak in pendulum angle at 0.055 radians, the angle reduces to around 0.01 radians. The final plot of the walking state shows that the bug is able to balance the pendulum without disrupting its walking cycle.

Data from two other interesting simulations of pendulum walking are also included. Figure 6-6 shows the data from a simulation in which the robot walked in a diamond pattern. The user inputs of speed and turn rate are constant at 0.2 m/s and 0.0 radians respectively. The direction of travel is changed every three seconds in order to achieve the diamond pattern. From the speed and pendulum angle graphs it is observed that there is a discontinuity at six seconds. This discontinuity is due to the fact that the robot's travel in the x direction switches from being negative to positive.

In the final simulation, Figure 6-7, the robot walks a complex pattern consisting of a loop and a sinusoidal path. The robot is directed to walk in this pattern using the turning rate input. At 0.5 seconds, Ω, changes to 0.8 radians and remains constant until 14.0 seconds. This produces the circle in the robot's path. The wavy pattern is created by varying the input Ω in a sinusoidal fashion. At about 16.0 seconds the frequency of the input is doubled. The result of the sinusoidal input is a couple large squiggles and a few smaller squiggles following.

Animations of these simulations can be found at:
http://www.ai.mit.edu/projects/leglab/simulations/hexapod/hexapod.html

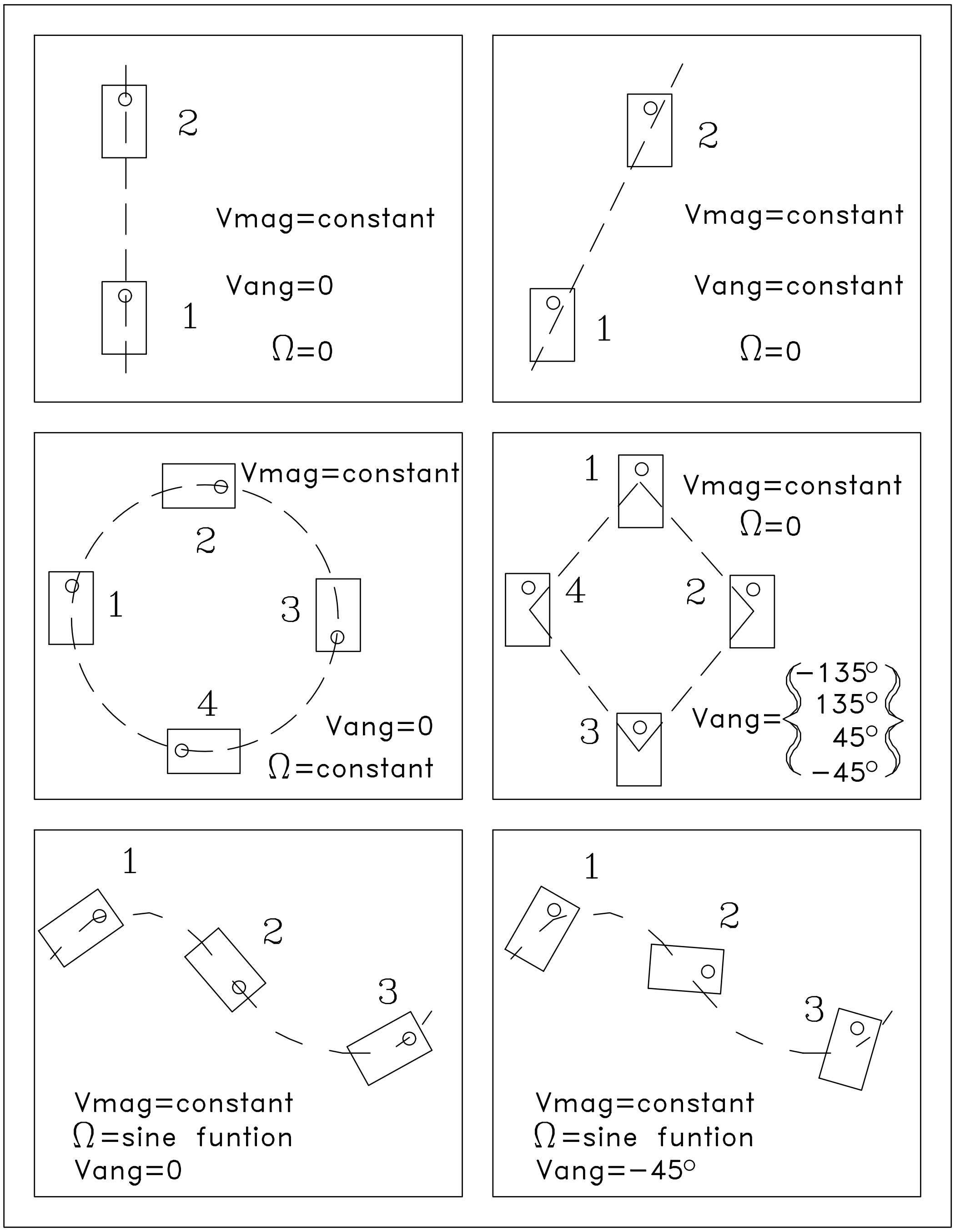

Figure 6-1: Diagrams of different walk paths of the robot as high-level user inputs are varied. V_{mag} is the speed of travel, V_{ang} is the direction of travel, and Ω is the rate of turning of the robot's body. The robot is illustrated as a small rectangle with a circle at it head. The dashed lines represent the path which the robot was able to walk given the three user inputs listed within its box.

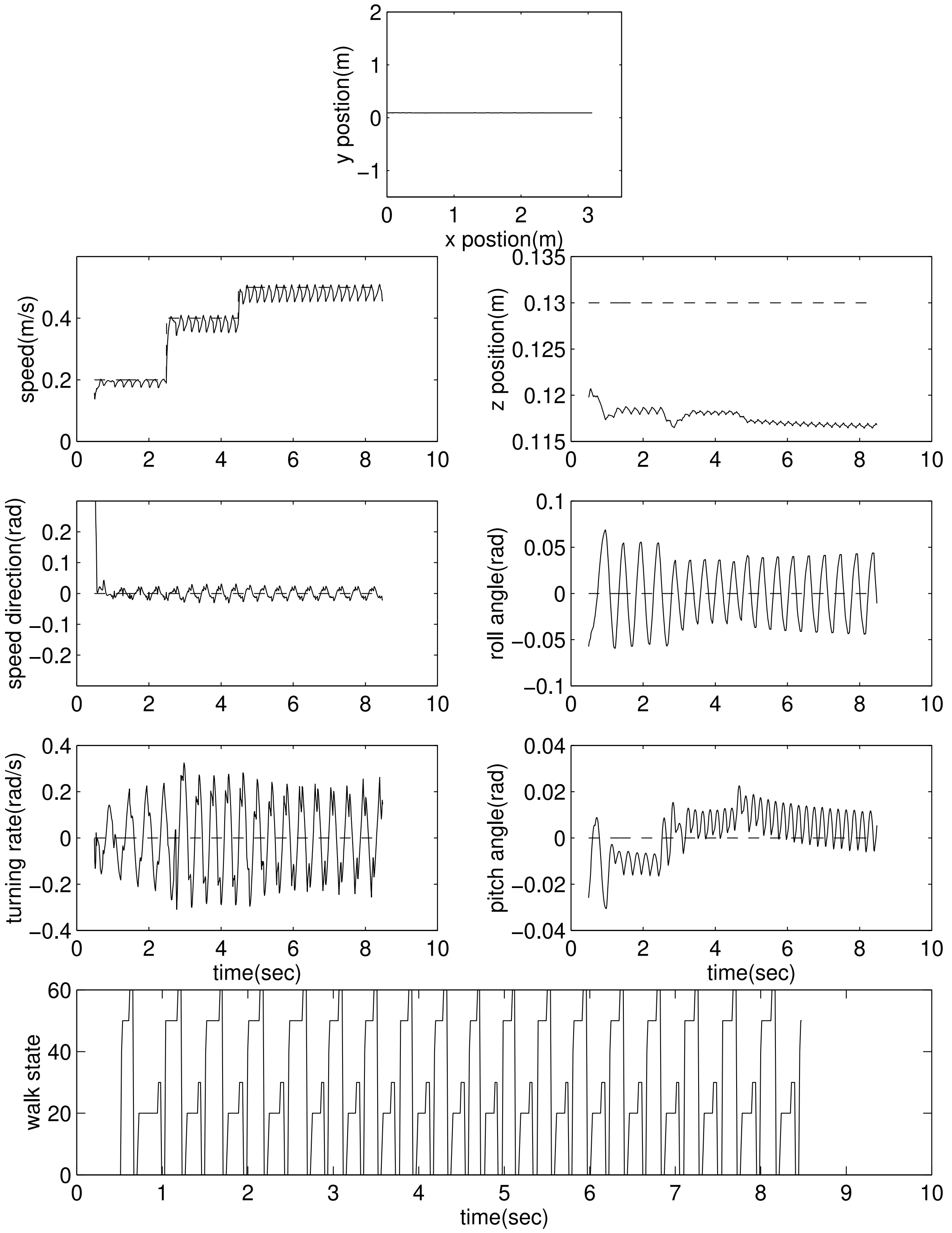

Figure 6-2: Graphs from the robot walking in straight line. The path the robot walks is illustrated in the first x-y plot. The three smaller graphs on the left show the desired user inputs (dashed lines) and the actual robot performance(solid lines). In this example, V_{mag} changes in steps from 0.2 to 0.4 to 0.5; $V_{ang} = 0.0$ and $\Omega = 0.0$. The desired and actual values of z, roll, and pitch are graphed on the right. The walk state graph on the bottom shows stable, periodic cycling through the walking algorithm.

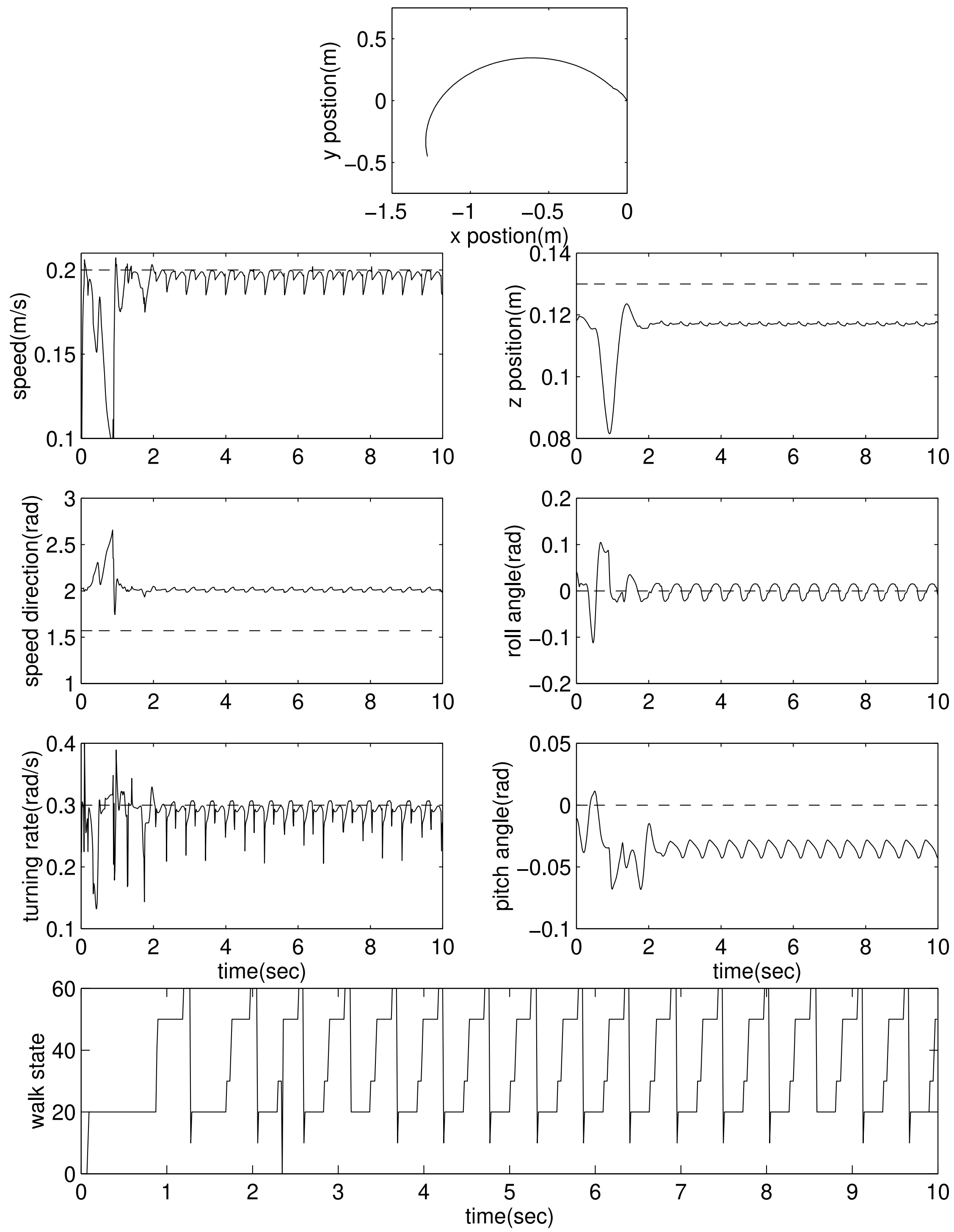

Figure 6-3: Graphs from the robot walking in a circle while always facing the center. This was accomplished by setting $V_{mag} = 0.2$, $V_{ang} = \pi/2$, and $\Omega = 0.3$. It is interesting to note that although the robot begins with a few stubbling steps, it is able to recover and exhibit stable walking.

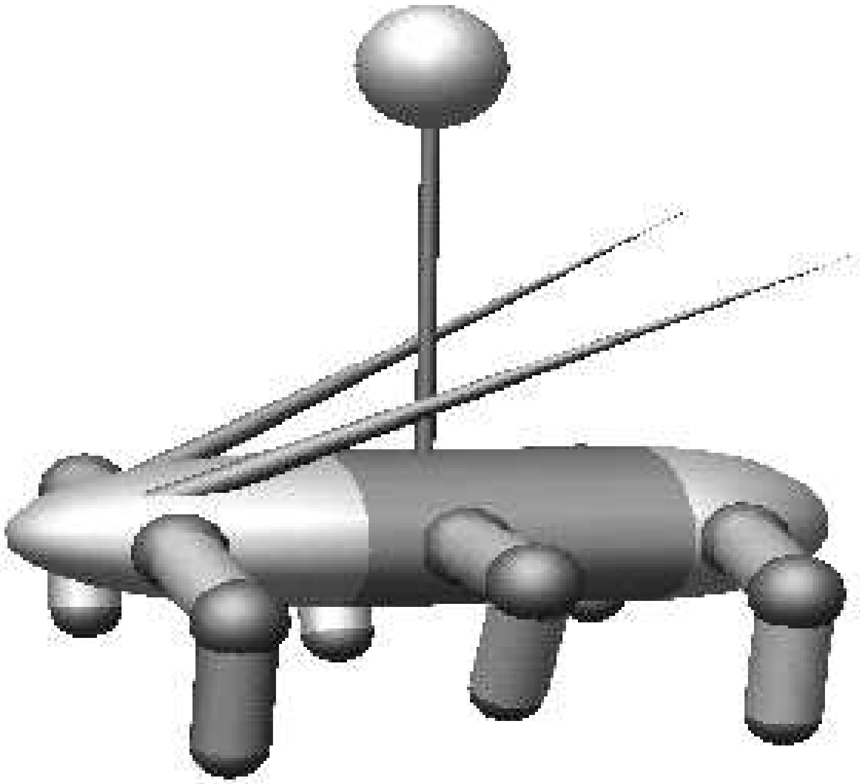

Figure 6-4: Picture of the hexapod robot with a pendulum on its back. The pendulum was connected to the hexapod in order to demonstrate the ease of controlling the robot's body using virtual model controllers. The robot was able to walk in a variety of different patterns while balancing the pendulum.

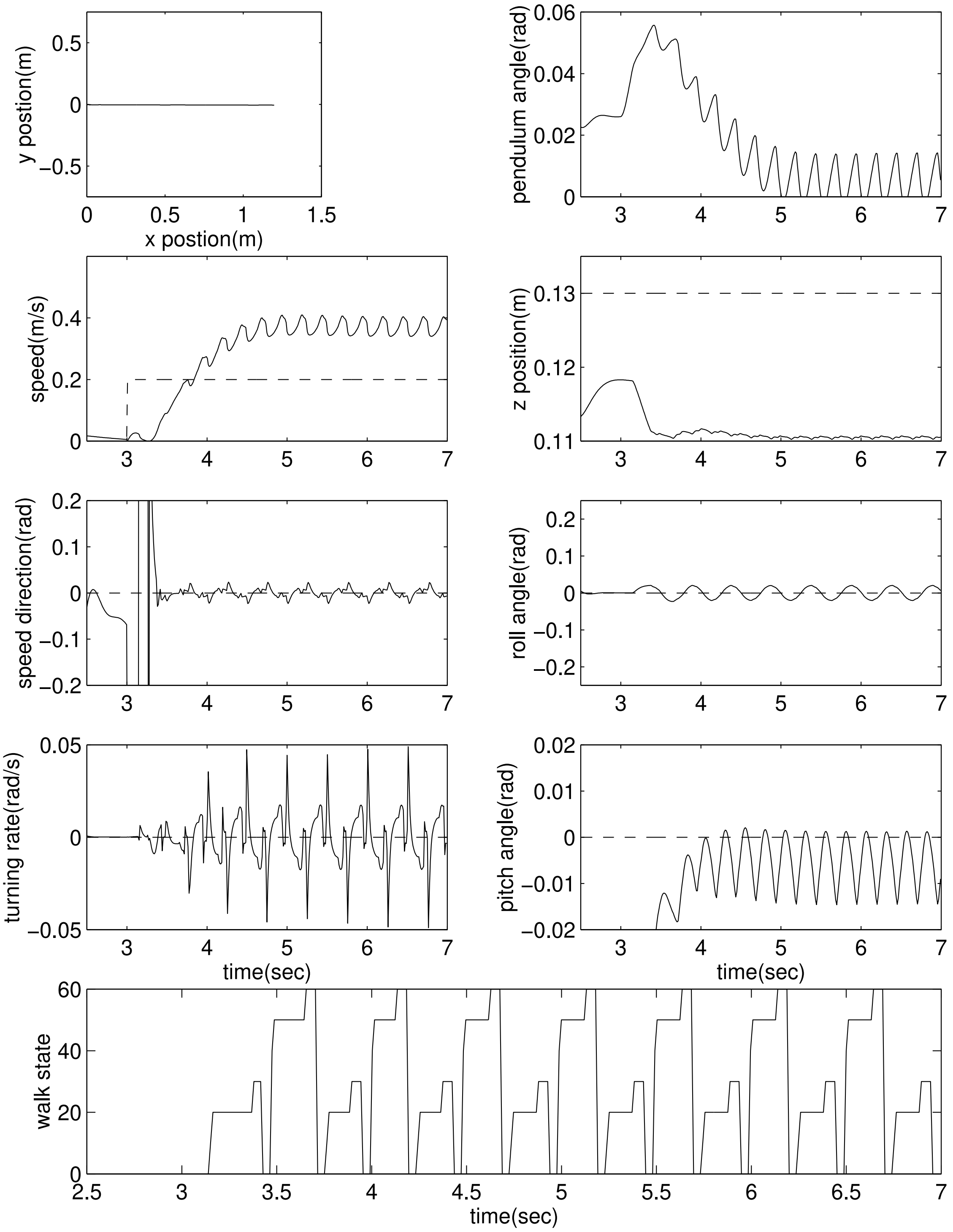

Figure 6-5: Graphs from the robot walking in a straight line with a pendulum on its back. The user inputs are $V_{mag} = 0.2$, $V_{ang} = 0.0$, and $\Omega = 0.0$. A graph is added in the upper right corner which exhibits the pendulum angle. The response exhibited by the robot and pendulum is typical of a non-minimum phase system.

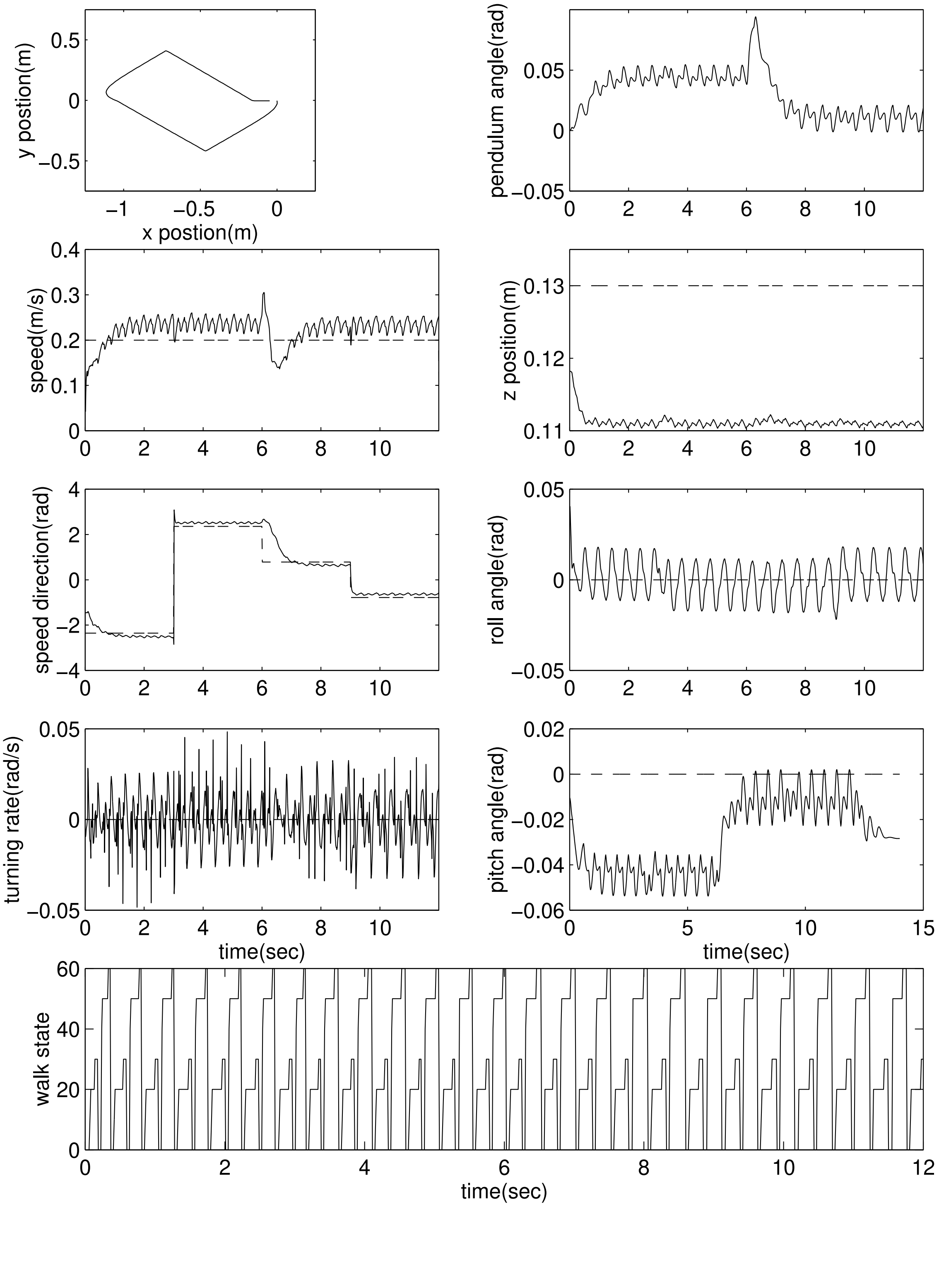

Figure 6-6: Graphs from the robot walking in a diamond pattern with a pendulum on its back. The hexapod traces out the diamond by varying its speed direction, V_{ang}. The turning rate remains constant at 0.0 and V_{mag} is also constant at approximately 0.25.

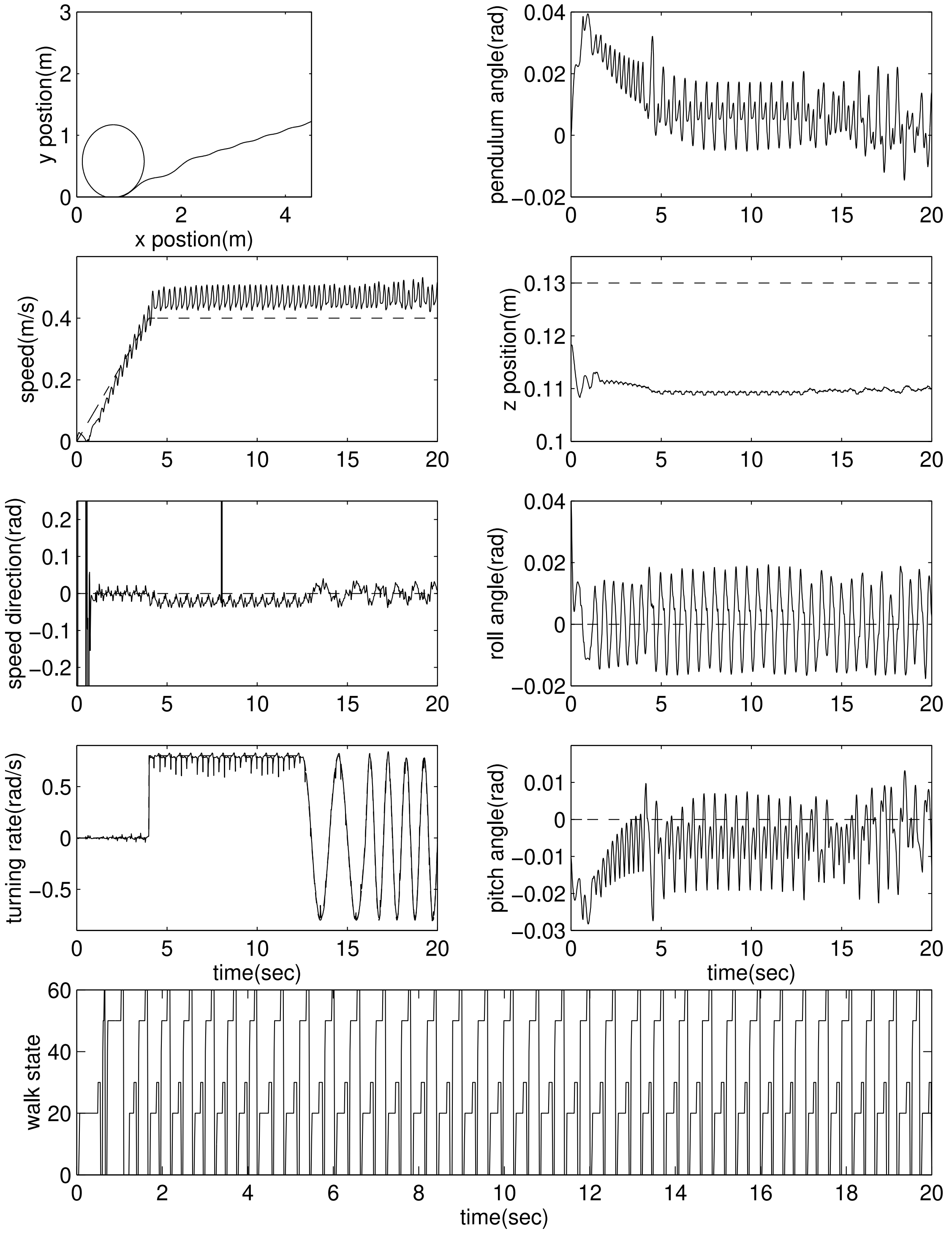

Figure 6-7: Graphs from the robot walking in a complex pattern with a pendulum on its back. In first four seconds, the speed of the robot ramps up to 0.4. After this the circular pattern traced out by the hexapod was dictated by the turning rate, Ω. From 5.0 to 14.0 seconds, Ω is constant at 0.7, and the robot traces a circle. After 14 seconds, the hexapod walks in a wave pattern by varying Ω in a sinusoidal fashion. At 16 second the frequency of the input is doubled. Note how well the actual turning rate mimics the desired rate.

Chapter 7

Conclusion

Through the implementation of virtual model controllers and a simple walking state machine, stable walking in a simulated hexapod was achieved. The simplicity of the control allowed the robot to maneuver in a variety of different patterns given only three user inputs. The robot was also able to successfully maneuver in these complex patterns while balancing a pendulum on its back.

These results suggest that Virtual Model Control is a powerful tool in robot design. In this case, Virtual Model Control was used to abstract the control of a three dimensional, multi-linked, parallel mechanism to three user inputs. Virtual Model Control not only produces successful results but the design and implementation of virtual model controller is relatively straightforward. Once the design of the robot is set, the virtual model math for each controller need only be derived once. Also because of the simplicity and modularity of virtual model controllers, higher level behaviors and control schemes can be implemented on top of them. Most importantly, the use of virtual components makes Virtual Model Control very intuitive. By using virtual components which are familiar to the designer, the behavior of the robot can be more easily directed.

The next step in this research will be to implement Virtual Model Control on an actual physical robot and have the robot conquer rough terrain. Although the success of the simulation can not guarantee that an actual robot will walk using Virtual Model Control, it suggests that Virtual Model Control might be suitable for use in an actual 3D walking hexapod robot.

Appendix A

Virtual Model Math Implementation

Most of this appendix is copied from a section in the Pratt, Torres paper [4].

The implementation of virtual models is straightforward. There are four major steps: definition of the virtual model reference frames, computation of the forward kinematics, calculation of a Jacobian matrix, and computation of the joint torques.

For parallel virtual models, one must divide the generalized force among the individual serial paths. This requires solving a system of equations (i.e. inverting a matrix). We describe an extension of Gardner's Partitioned Force Set Control Technique method which minimizes the necessary computational requirements. All equation derivation can be performed off-line.

A.1 Definition of the Virtual Model Frames

Each virtual model requires three coordinate frames. These are the action frame $\{B\}$, the reaction frame $\{A\}$, and the reference frame $\{O\}$ (Figure A-1). The action frame defines the virtual model connection upon which the generalized forces act. The reaction frame defines the second attachment point of the virtual model. The reference frame is the coordinate system in which all displacements, forces, etc are expressed.

In most treatments of similar control methods, the reaction frame is assumed to be fixed to ground and usually is not even mentioned. However, one must remember Newton's Third Law. One cannot simply describe a force *acting at a point.* One must describe forces acting *between two points.* Similarly, in the virtual model context, one cannot simply define a generalized force acting

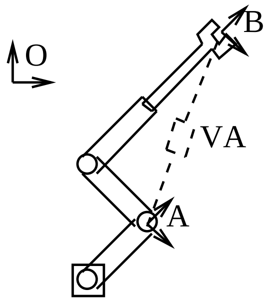

Figure A-1: Virtual Model Reference Frames. $\{B\}$ is the action frame. $\{A\}$ is the reaction frame. $\{O\}$ is the reference frame. VA represents the virtual model.

on a frame but must define a generalized force acting between two frames.

The action, reaction, and reference frames do not need to be inertial, nor cartesian, nor be directly attached to parts of the physical robot. All three frames simply need to be well defined. In most cases, however, all three frames will be cartesian; they will be connected to logical points on the robot such as joints, links, or end effectors; and the reference frame, {O}, will either be inertial or coincide with {A} or {B}.

In our use of the reference frame, {O}, we are not concerned about its location, and in fact it need not be specified. All that is needed is the relative rotation between the reference frame and the action and reaction frames.

Choice of the virtual model frames is a major part of defining a virtual model and must be done carefully if the desired results are to be obtained. For example, both the action and reaction frames might be defined so that there cannot be a relative rotation between them in any orientation of the robot. This is perfectly valid but it then makes it impossible to produce a relative torque between the two. The best tools for determining how to attach the virtual model's frames is physical intuition, insight, and experience.

A.2 Derivation of the Forward Kinematics

Computing the forward kinematic map, ${}^{A}_{B}\vec{X}$, is well documented [1]. For any serial manipulator with revolute and prismatic joints, ${}^{A}_{B}\vec{X}$ will be a closed form function of the joint angles and prismatic displacements, $\vec{\Theta}$, lying between frames {A} and {B}.

To express the kinematics between frames {A} and {B} with reference to frame {O}, we use the rotation matrix between {O} and {A}

$$ {}^{O}({}^{A}_{B}\vec{X}) = {}^{O}_{A}R\,{}^{A}_{B}\vec{X} \tag{A.1} $$

Note that we ignore the displacement between {O} and {A}. This is because we are concerned with the relative kinematics between the reaction and action frames and not the absolute location of any of the frames, with reference to {O} or World Coordinates.

All virtual models do not necessarily need to provide actuation. They can be used with the forward kinematics alone and act simply as virtual sensors.

A.3 Derivation of the Jacobian Matrix

Let the Jacobian Matrix for a virtual model be defined as:

$$ {}^{A}_{B}J = \frac{\partial}{\partial\Theta}\,{}^{A}_{B}\vec{X} \tag{A.2} $$

The Jacobian can be used for several mappings. It can be used to calculate generalized velocities by

$$ {}^{A}_{B}\dot{\vec{X}} = {}^{A}_{B}J\;\dot{\vec{\Theta}} \tag{A.3} $$

which holds by definition. To express velocities with respect to the reference frame, we again use the rotation matrix,

$$ {}^{O}({}^{A}_{B}\dot{\vec{X}}) = {}^{O}_{A}R\,{}^{A}_{B}\dot{\vec{X}} \tag{A.4} $$

The Jacobian is also used to transform generalized forces to joint torques as discussed next.

There are several techniques to compute the Jacobian for the 3D case. One method described in [1] is to recursively compute the joint to cartesian velocity relationship (Equation A.3) and extract the Jacobian Matrix.

A.4 Computation of the Joint Torques

To compute the joint torques which will successfully emulate the virtual model, we use the following equation

$$\vec{\tau} = {}^{A}_{B}J^{T} \, {}^{A}_{B}\vec{F} \tag{A.5}$$

where $\vec{\tau}$ is the vector of joint torques (or forces for prismatic joints) and ${}^{A}_{B}\vec{F}$ is the generalized force vector acting on action frame {B} from reaction frame {A} defined with respect to frame {A}. The generalized force vector will typically consist of a force and moment vector.

Equation A.5 can be derived starting from the energy balance $\vec{\tau}^{T}\delta\vec{\Theta} = \vec{F}^{T}\delta\vec{X}$ [1]. It requires that the generalized forces which act on action frame {B} be specified in terms of reaction frame {A}. If the forces are expressed in reference frame {O}, we must use the rotation matrix from {A} to {O} to express them in {A},

$${}^{A}_{B}\vec{F} = {}^{A}_{O}R \, {}^{O}({}^{A}_{B}\vec{F}) \tag{A.6}$$

We can combine Equations A.5 and A.6 to get

$$\vec{\tau} = {}^{A}_{B}J^{T} \, {}^{A}_{O}R \, {}^{O}({}^{A}_{B}\vec{F}) = {}^{O}({}^{A}_{B}J)^{T} \, {}^{O}({}^{A}_{B}\vec{F}) \tag{A.7}$$

where ${}^{O}({}^{A}_{B}J)^{T} = {}^{A}_{B}J^{T} \, {}^{A}_{O}R$.

One point of note is that "admissible" generalized forces lie in the row space of ${}^{O}({}^{A}_{B}J)^{T}$. By admissible, we mean forces which can be realized using the available joint forces. For example, if no relative motion can be realized along a certain direction, then no matter how large a force is produced along that direction by the virtual models, no effect will result on the robot (assuming rigid links). Similarly, any generalized forces which lie in the null space of ${}^{O}({}^{A}_{B}J)^{T}$ will produce no torque at the joints and hence have no effect on the robot. Whether or not such an inadmissible generalized force should be allowed probably depends on the implementation. In any case, an inadmissible force should be detectable. An easy test is to see if it lies in the row space of the Jacobian transpose. When implementing parallel virtual models, as described below, it is important that the inadmissible force constraints be known for each of the serial paths of the virtual model so that the desired generalized force can be accurately divided among the individual serial paths.

A.5 Parallel Mechanisms and Multi-Frame Virtual Models

With a serial link structure all the necessary equations are computationally inexpensive. With a parallel structure a matrix inversion is necessary in solving the force distribution problem.

We start by defining one action frame {B}, one reference frame {O} and multiple reaction frames $\{A_i\}$ as in Figure A-2. Frame $\{A^*\}$ is a construct used to represent all of the frames $\{A_i\}$ and can be viewed as the conglomerate reaction frame. The parallel virtual model can be considered as multiple serial sub-component acting on the same action frame. Each serial sub-component has a corresponding Jacobian which is calculated as in Section A.3. We combine these to get

$$\begin{bmatrix} \vec{\tau}_1 \\ \vec{\tau}_2 \\ \vdots \\ \vec{\tau}_p \end{bmatrix} = \begin{bmatrix} {}^{O}({}^{A_1}_{B}J)^{T} & 0 & \cdots & 0 \\ 0 & {}^{O}({}^{A_2}_{B}J)^{T} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & {}^{O}({}^{A_p}_{B}J)^{T} \end{bmatrix} \begin{bmatrix} {}^{O}({}^{A_1}_{B}\vec{F}) \\ {}^{O}({}^{A_2}_{B}\vec{F}) \\ \vdots \\ {}^{O}({}^{A_p}_{B}\vec{F}) \end{bmatrix} \tag{A.8}$$

We now have a mapping from sub-component generalized forces to joint torques. However, we wish to specify a single generalized force to act on the action frame. Since the action frame and reference frame are coincidental for all the sub-components, the vector sum of the individual generalized forces must equal the desired generalized force,

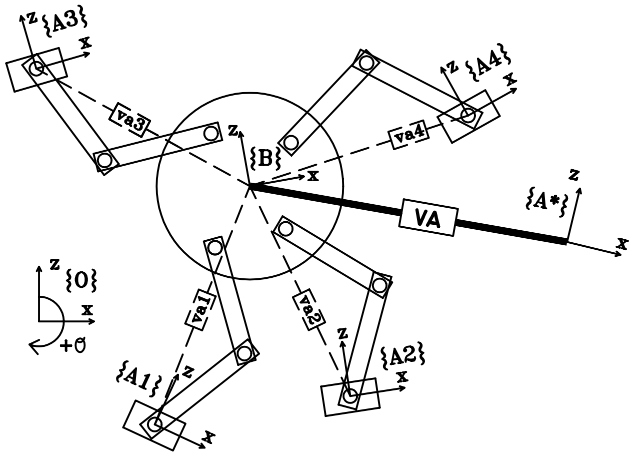

Figure A-2: Parallel Virtual Model Reference Frames. Frame {B} is the action frame. Frames $\{A_i\}$ are the reaction frames. Frame {O} is the reference frame. VA represents the conglomerate virtual model which is comprised of the individual virtual models VA_i. Frame $\{A^*\}$ is an imaginary construct which represents the reaction frame of the conglomerate virtual model.

$$\sum_{i=1}^{p} {}^{O}({}^{A_i}_{B}\vec{F}) = {}^{O}\,({}^{A^*}_{B}\vec{F}) \tag{A.9}$$

We need to solve for ${}^{O}({}^{A_i}_{B}\vec{F})$ in terms of ${}^{O}({}^{A^*}_{B}\vec{F})$. To do this we must add a number of constraints. Some of these constraints will arise due to the inadmissibility of certain individual force directions. These constraints can be determined by examining the row space of the individual ${}^{O}({}^{A_i}_{B}J)^T$. Others will arise from constraints on the robot such as unactuated joints. The rest of the constraints can be used as design degrees of freedom.

Once enough constraints have been determined, we will have a square invertible constraint matrix, K, so that the constraints can be written in the form

$$\begin{bmatrix} {}^{O}({}^{A^*}_{B}\vec{F}) \\ \vec{c} \end{bmatrix} = K \begin{bmatrix} {}^{O}({}^{\vec{A_1}}_{B}\vec{F}) \\ {}^{O}({}^{\vec{A_2}}_{B}\vec{F}) \\ \vdots \\ {}^{O}({}^{\vec{A_p}}_{B}\vec{F}) \end{bmatrix} \tag{A.10}$$

and the individual sub-force vectors solved for by inverting the constraint matrix, K.

The elements of $\vec{c}$ can be extra control variables, such as individual interaction forces, or 0 for constraints which are solely a function of the forces ${}^{O}({}^{A_i}_{B}\vec{F})$. Of course, we need not retain the columns of K^{-1} which are multiplied by 0. We can now substitute back into A.8 to get the final force to torque relationship.

A.5.1 Inverting the Constraint Matrix

Solving Equation A.10 requires inverting a potentially large sparse matrix. We show here a method for taking advantage of the structure of the constraint matrix, K, in order to reduce computational requirements. The method is an extension of Gardners's Partitioned Force Set Control Technique [3, 2].

Gardner partitions the forces into a Minimum Force Set (MFS) and a Redundant Force Set (RFS). We extend Gardner's method by adding the Constrained Force Set (CFS) for dealing with natural constraints, such as underactuated legs, point feet, and limp joints.

We assume that all the serial paths of the parallel virtual models are of the same structure, with the same number of constraints. We do this only to simplify the following explanation. The mathematics is easily extended to the general case.

We define the following constants,

n	dimension of the generalized force to be applied
p	number of serial paths of the parallel virtual model
l	number of constraints in each serial path
d = n-l	number of non-constrained degrees of freedom per serial path
r = pd - n	number of redundant serial path virtual force componentss

The number of force elements in the Minimum Force Set will be n; in the Constrained Force Set l per serial path (pl total); in the Redundant Force Set r. How the forces are partitioned into these sets depends on the design constraints one wishes to implement and the limitations placed on these constraints by the extended partitioned force set method.

As an example, suppose we have a 100 leg millipede with 2 joints per leg and point feet. We would have $n = 6$ for the 3 elements of the force vector and 3 elements of the moment vector which we wish to exert; $p = 100$ for the 100 legs; $l = 4$ for the 3 constraints of no torques at the feet and the 1 constraint provided by the underactuation of having only 2 joints per leg; $d = 2$ meaning each leg can provide a 2 dimensional force vector from the 6 dimensional space; and $r = 194$ meaning we have 194 redundancies and therefore can specify up to 194 design constraints. The Minimum Force Set would contain 6 elements; the Constrained Force Set 400; and the Redundant Force Set 194.

We first partition the virtual forces into the Constrained Force Set (CFS), f^{ib}, and those not in the CFS, f^{ia}, and rearrange Equation A.10 into the following form,

$$\begin{bmatrix} f_d^a \\ f_l^b \\ t_l^{1c} \\ t_l^{2c} \\ \vdots \\ t_l^{pc} \\ c_r \end{bmatrix} = \begin{bmatrix} I_{d\times d} & 0_{d\times l} & I_{d\times d} & 0_{d\times l} & \cdots & I_{d\times d} & 0_{d\times l} \\ 0_{l\times d} & I_{l\times l} & 0_{l\times d} & I_{l\times l} & \cdots & 0_{l\times d} & I_{l\times l} \\ J^{1a}_{l\times d} & J^{1b}_{l\times l} & 0_{l\times d} & 0_{l\times l} & \cdots & 0_{l\times d} & 0_{l\times l} \\ 0_{l\times d} & 0_{l\times l} & J^{2a}_{l\times d} & J^{2b}_{l\times l} & \cdots & 0_{l\times d} & 0_{l\times l} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0_{l\times d} & 0_{l\times l} & 0_{l\times d} & 0_{l\times l} & \cdots & J^{pa}_{l\times d} & J^{pb}_{l\times l} \\ D^{1a}_{r\times d} & D^{1b}_{r\times l} & D^{2a}_{r\times d} & D^{2b}_{r\times l} & \cdots & D^{pa}_{r\times d} & D^{pb}_{r\times l} \end{bmatrix} \begin{bmatrix} f_d^{1a} \\ f_l^{1b} \\ f_d^{2a} \\ f_l^{2b} \\ \vdots \\ f_d^{pa} \\ f_l^{pb} \end{bmatrix} \tag{A.11}$$

where I is the identity matrix, 0 is the zero matrix, J^{ia} , J^{ib} are natural constraint matrices, and D^{ia}, D^{ib} are design constraint matrices. The Constrained Force Set consists of the forces f^{ib} while the forces f^{ia} belong in the Minimum and Redundant Force Sets. The subscript on each matrix block show the size of the block.

The constrained torques, t^{ic} are those which can be measured or inferred but not controlled. For example, with an unactuated limp joint, $t^c = 0$ while for a joint containing a passive linear spring, $t^c = k\theta$.

We can reduce the size of the matrix in equation A.11 by taking advantage of the sparseness of the natural constraint blocks. Each natural constraint row can be written as

$$t^{ic} = J^{ia} f^{ia} + J^{ib} f^{ib}$$

Solving for f^{ib} we have,

$$f^{ib} = Q^i f^{ia} + (J^{ib})^{-1} t^{ic} \tag{A.12}$$

where,

$$Q^i = [-(J^{ib})^{-1}J^{ia}]_{l\times d} \tag{A.13}$$

We can substitute this back into Equation A.11 to get a reduced set of equations,

$$\begin{bmatrix} f_d^a \\ f_l^b - u_l \\ c_r - v_r \end{bmatrix} = \begin{bmatrix} I_{d\times d} & I_{d\times d} & \cdots & I_{d\times d} \\ Q^1 & Q^2 & \cdots & Q^p \\ S^1 & S^2 & \cdots & S^p \end{bmatrix} \begin{bmatrix} f_d^{1a} \\ f_d^{2a} \\ \vdots \\ f_d^{pa} \end{bmatrix} \tag{A.14}$$

where,

$$\begin{aligned} u_l &= \sum_i (J^{ib})^{-1} t^{ic} \\ v_r &= \sum_i D^{ib} (J^{ib})^{-1} t^{ic} \\ S^i &= [D^{ib}Q^i + D^{ia}]_{r\times d} \end{aligned} \tag{A.15}$$

We have now reduced the matrix size from $np \times np$ to $dp \times dp$ by eliminating the Constrained Force Set (CFS). Note that Equation A.14 is still in the general form, i.e. we have put no restrictions on the design constraint equations. We could stop here and invert the new $dp \times dp$ matrix, if it were computationally feasible. In order to further reduce the size of the matrix, we can eliminate the Redundant Force Set (RFS) by specifying our design constraints in a proper manner.

Similar to [3], we specify the Redundant Force Set, f^r, in terms of the Minimum Force Set, f^m.

$$f_r^r = c_r - B_{r\times n} f_n^m \tag{A.16}$$

where f^r is the Redundant Force Set, f^m is the Minimum Force Set, c is a vector of variables or constants, and B is the design constraint matrix.

Writing the design constraints in terms of Equation A.16 requires that

$$D^{ib} = 0 \; \forall i$$

and D^{ia} are restricted so that we can rearrange Equation A.14 into the following form,

$$\begin{bmatrix} f_n \\ c_r \end{bmatrix} = \begin{bmatrix} A^m_{n\times n} & A^r_{n\times r} \\ B_{r\times n} & I_{r\times r} \end{bmatrix} \begin{bmatrix} f_n^m \\ f_r^r \end{bmatrix} \tag{A.17}$$

We can now eliminate the RFS, f^r, from Equation A.17 to by substituting Equation A.16 into A.17 to get

$$[f - A^r c]_n = [A^m - A^r B]_{n\times n} [f_n^m] \tag{A.18}$$

To solve for the MFS, f^m, we must invert the $n \times n$ matrix in Equation A.18. The Redundant Force Set is then computed by plugging back into equation A.16. Finally, we can rearrange the Minimum and Redundant Force Sets to get f^{ia} and substitute back into Equation A.12 to solve for the Constrained Force Set.

In order to use the above method, one must write all the design constraints in the form of Equation A.16. In writing these constraints, one must specify the Redundant Force Set and the Minimum Force Set. The remaining forces are the Constrained Force Set. One must make sure that the choice of design constraints allows for a solution of Equation A.18 to exist.

Examining Equations A.14 to A.18 we see that we take p $[l \times l]$, and 1 $[n \times n]$ matrix inversions in solving for the virtual forces applied to each serial path. These inversions will take significantly less computational resources to perform than the original $np \times np$ inversion. Since the computational complexity of matrix inversion scales with the cube of the matrix size, the above method is $O(pl^3 + n^3)$

whereas inverting the original matrix is $O(p^3n^3)$.

A.5.2 Pseudo-Inversion of the Constraint Matrix

The above discussion assumed that we specified r design constraints. In some cases, we may not wish to specify as many design constraints, and we may not wish to be limited to specifying all the constraints in terms of Equation A.17. Suppose we wish to specify only s design constraints, where $s < r$. If we specify the design constraints in the form,

$$f_s^{r2} = c - B^1 f^m - B^2 f^{r1} \tag{A.19}$$

then instead of Equation A.17, we will have,

$$\begin{bmatrix} [f - A^{r2}c]_n \\ c_s \end{bmatrix} = \begin{bmatrix} A^m_{n\times n} & A^{r1}_{n\times(r-s)} & A^{r2}_{n\times s} \\ B^1_{s\times n} & B^2_{s\times(r-s)} & I_{s\times s} \end{bmatrix} \begin{bmatrix} f^m_n \\ f^{r1}_{r-s} \\ f^{r2}_s \end{bmatrix} \tag{A.20}$$

Utilizing the structure of this matrix, we get the following set of Equations,

$$\begin{bmatrix} f_n \end{bmatrix} = \begin{bmatrix} [A^m - A^{r2}B^1]_{n\times n} & [A^{r1} - A^{r2}B^2]_{n\times r-s} \end{bmatrix} \begin{bmatrix} f^m_n \\ f^{r1}_{r-s} \end{bmatrix} \tag{A.21}$$

The above matrix is non-square. To solve the above equation for f^m and f^{r1}, we can use the Moore-Penrose pseudo-inverse,

$$A^+ = A^T(AA^T)^{-1} \tag{A.22}$$

and then solve for f^{r2} by substituting back into Equation A.19.

The pseudo-inverse still requires inverting an $n \times n$ matrix so our computational requirements are about the same as the previous method.

Bibliography

[1] John J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, 1989.

[2] J. F. Gardner. Force distribution in walking machines over rough terrain. *J. of Dynamic Systems, Measurement and Control*, 113:754–758, 1991.

[3] J. F. Gardner, K. Srinivasan, and K. J. Waldron. A solution for the force distribution problem in redundantly actuated closed kinematic chains. *J. of Dynamic Systems, Measurement and Control*, 112:523–526, 1990.

[4] J. Pratt, A. Torres, P. Dilworth, and G. Pratt. Virtual actuator control. *IEEE International Conference on Intelligent Robots and Systems*, 1996. Submitted Feb. 1996.

[5] Jerry E. Pratt. Virtual model control of a biped walking robot. Master's thesis, Massachusetts Institute of Technology, June 1996.

[6] Marc. H. Raibert. *Legged Robots That Balance.* MIT Press, Cambridge, MA., 1986.

[7] Shin-Min Song and Kenneth J. Waldron. *Machines That Walk.* MIT Press, Cambridge, MA., 1989.

[8] Gilbert Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, Wellesley, MA., 1993.