Virtual Model Control of a Biped Walking Robot

by

Jerry E. Pratt

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1995

Author ....
Department of Electrical Engineering and Computer Science
August 25, 1995

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

Accepted by ....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Virtual Model Control of a Biped Walking Robot

by

Jerry E. Pratt

Submitted to the Department of Electrical Engineering and Computer Science
on August 25, 1995, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The transformation from high level task specification to low level motion control is a fundamental issue in sensorimotor control in animals and robots. This thesis develops a control scheme called virtual model control which addresses this issue.

Virtual model control is a motion control language which uses simulations of imagined mechanical components to create forces, which are applied through joint torques, thereby creating the illusion that the components are connected to the robot. Due to the intuitive nature of this technique, designing a virtual model controller requires the same skills as designing the mechanism itself. A high level control system can be cascaded with the low level virtual model controller to modulate the parameters of the virtual mechanisms. Discrete commands from the high level controller would then result in fluid motion.

An extension of Gardner's Partitioned Actuator Set Control method is developed. This method allows for the specification of constraints on the generalized forces which each serial path of a parallel mechanism can apply.

Virtual model control has been applied to a bipedal walking robot. A simple algorithm utilizing a simple set of virtual components has successfully compelled the robot to walk eight consecutive steps.

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

Acknowledgments

I thank my advisor, Gill Pratt, for letting me take his idea and run with it in whatever direction I thought appropriate.

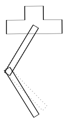
Thanks to all the members of the Leg Lab for the excellent support and advice throughout the year. Thanks especially to Karl Leeser for proofreading the document without remorse.

This thesis would not have been possible without the previous hard work and help from several people. Peter Dilworth designed and built Spring Turkey, our bipedal robot. Anne Wright developed the operating system software. Ann Torres was instrumental in helping develop the math for application to parallel mechanisms and the 3-D example. Ann also drew the figures of the 3D hexapod example. Abbe Cohen helped with the robotic foot example. Thanks to Robert Ringrose and whoever else helped develop the simulation software package which has become a foundation of the lab.

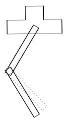
This thesis is dedicated to my best friend, Megan Benson.

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

Contents



List of Figures



Nomenclature

$\{A\}$, $\{A_i\}$	virtual model reaction frames
$\{B\}$	virtual model action frame
$\{O\}$	virtual model reference frame
$\{A^*\}$	conglomerate virtual model reaction frame
$\vec{\Theta}$	robot joint angles and prismatic displacements
${}^{A}_{B}\vec{X}$	forward kinematic map from frame {A} to {B} with reference to frame {A}
${}^{O}({}^{A}_{B}\vec{X})$	forward kinematic map from frame {A} to {B} with reference to frame {O}
${}^{O}_{A}R$	rotation matrix from frame {O} to frame {A}
${}^{A}_{O}R$	rotation matrix from frame {A} to frame {O}
${}^{A}_{B}J$	Jacobian matrix from frame {A} to frame {B} with reference to frame {A}
${}^{O}({}^{A}_{B}J)$	Jacobian matrix from frame {A} to frame {B} with reference to frame {O}
$\vec{\tau}$	vector of joint torques and forces
${}^{A}_{B}\vec{F}$	generalized forces acting on action frame {B} from reaction frame {A} with reference to frame {A}
${}^{O}({}^{A}_{B}\vec{F})$	generalized forces acting on action frame {B} from reaction frame {A} with reference to frame {O}
$\vec{v}$	user defined parameters and variables
$\vec{s}$	state vector of dynamic simulated entities
k	virtual spring function
b	virtual damper function
K	parallel virtual model constraint matrix
n	dimension of the generalized force to be applied
p	number of individual serial links comprising a parallel virtual component
l	number of constraints in each serial path of a parallel virtual component
d	number of non-constrained degrees of freedom per serial path of a parallel virtual component
r	number of redundant serial path virtual force components

G	torque controller transfer function
Z	torque controller impedance function
R	robot admittance function
VM	desired virtual model function
VM'	actual virtual model function
E	virtual model multiplicative error
T	virtual model and robot function in parallel with multiplicative error
CL	desired closed loop admittance function of robot and virtual model as seen by the environment
CL'	actual closed loop admittance function of robot and virtual model as seen by the environment
ENV	environment impedance function

Chapter 1

Introduction

Design and analysis of control systems for non-linear systems such as robots seems to be orders of magnitude more difficult than for linear systems. Despite these difficulties, researchers have developed several powerful techniques for controlling several classes of non-linear systems. For example, impedance control and adaptive controllers are two examples of powerful techniques for controlling robot arms. The former is well-suited for controlling contact impedances of the arm as seen by the environment. The latter is well suited for doing trajectory following while adapting to robot parameters.

A class of robots which suffers from lack of powerful techniques for its control is dynamic legged locomoting robots. These robots are extremely difficult to control since they

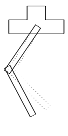
- Are nonlinear and operate throughout the range of their state space
- Act in a gravity field
- Interact with a semi-structured, complex environment
- Are nominally unstable
- Are Multi Input, Multi Output (MIMO)
- Exhibit time variant dynamics with zeroeth and first order discontinuities as support modes are transitioned (e.g. single support, double support, ballistic phase, etc.)
- Require both continuous control and discrete control (for step-to-step transitions)

In addition, the performance measures of such robots are much different than typical notions of performance such as command following and disturbance rejection. Performance for these robots is usually defined in some of the following terms:

- Biological similarity
- Efficiency, i.e. Distance traveled per unit energy input
- Locomotion smoothness
- Top speed
- Robustness to rough terrain

Because of these difficulties, the only acceptable tools for analyzing such systems is simulation or experimentation and the only good design tools are often physical intuition, parameter iteration, and "hand tweaking".

Despite these difficulties, robots have been developed at MIT's Leg Laboratory which can hop, run, and perform gymnastic maneuvers [25]. The control algorithms employed are very simple and



draw on physical intuition. Other factors contributing to the success of the robots include excellent design and construction, conservative over-actuation, and a bit of luck. However, it is difficult, if not impossible, to perform an acceptable mathematical analysis of why the control algorithms employed on these robots are successful in making them run.

If one wishes to expand the toolbox of analysis and design techniques for such a class of robots, he or she must either make a quantum leap in control system design and analysis mathematics, or else exploit physical intuition. In this thesis we present a control technique, dubbed virtual model control, which attempts the latter.

1.1 Virtual Model Controllers

Virtual model control is a language for describing interactive force behaviors. This control technique uses simulations of virtual mechanical components to generate actuator torques (or forces) thereby creating the illusion that the simulated components are connected to the real robot. Because any imaginable component can be used, virtual model control requires no loss of generality and traditional control techniques can be utilized in the same context. For example, a proportional-derivative (PD) servo at a joint can be implemented as a virtual torsional spring-damper with spring constant equal to servo position gain, damping constant equal to derivative gain, and set point equal to desired joint angle. Virtual components can even contain adaptive and learning elements [22].

One of the reasons virtual model control is being developed is because it is generally difficult to describe motion control. For example, how would one describe what he or she is doing when performing deep knee bends and how would one control a robot to perform this task? A quick and easy method would be to attach a virtual spring between the robot's feet and its waist and change the rest-length in a sinusoidal fashion.

Another complex task which is difficult to describe using traditional control methodologies is the throwing of a punch by a robotic boxer. A simple virtual model controller might consist of a virtual force source connected between the robot's chest (to throw a jab) and the end of its hand, producing a virtual force in the direction of the opponent. Virtual components could be connected between the feet and the body to help maintain balance (see Figure 1-1). If the robot wanted to throw a knock-out upper-cut, using everything it has, virtual actuators could be attached to its hand from its feet, hip, and chest. Whereas the jab controller would only cause chest and arm torques to be exerted, the upper-cut controller would use every available joint torque in the robot's body.

Some benefits of virtual model control are that it is compact, requires a relatively small amount of computation, and can be implemented in a distributed manner. These benefits are due to the fact that no matrix or function inverses need be computed for serial links; all equations can be precomputed in closed form and optimized; and multiple virtual components can be independently computed and superposed since their outputs, joint torques, are linearly additive.

Furthermore, a high level controller could be implemented as a state machine which simply changes virtual component connections or parameters at the state transitions. Even though a discrete high level controller would be used, the overall motion would be fluid as the virtual components would be continuous.

Like any new idea, virtual model control draws from several techniques that are currently used in practice. The following list contains six of these techniques and describes how virtual model control compares and contrasts.

- *Virtual Reality / Haptic Interfaces* [28]. Virtual Reality is a means of creating the illusion of virtual entities existing in a simulated environment. Haptic Interfaces are devices which provide tactile and force feedback for Virtual Reality applications. Virtual model control is similar to Virtual Reality and Haptic Interfaces in that it is used to emulate imaginary entities. It is different in that it emulates components which are connected to a robot in attempt to persuade the robot to *perform a certain task*, rather than to create an illusion to a human operator.

- *Hybrid Position/Force Control* [24]. Hybrid Position/Force Control is a method for com-

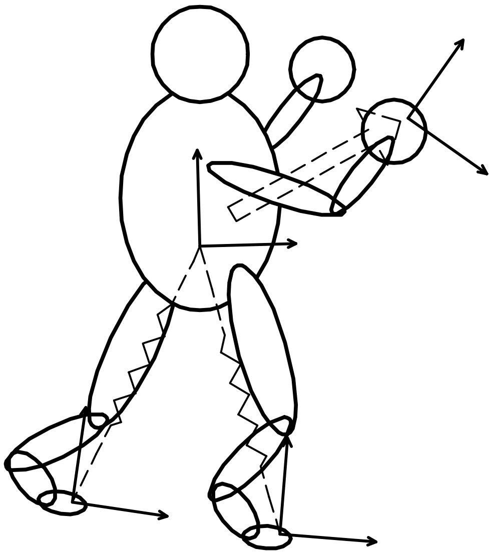


Figure 1-1: Robotic Boxer with virtual components attached from the feet to the body to maintain balance and from the body to the hand to throw a jab.

manding positions in an unconstrained cartesian subspace while commanding forces in the orthogonal subspace. Virtual model control is similar to hybrid position/force control in that both desired positions and forces can be controlled. However, with virtual model control the space in which positions are controlled need not be orthogonal to the space in which forces are controlled. In virtual model terms, a PD position controller is equivalent to a spring-damper mechanism. There is no reason why a real spring-damper could not be attached in parallel with a real force source, so there is no reason why both virtual components cannot be attached to a robot and act in non-orthogonal spaces.

- *Stiffness Control* [26]. Stiffness Control is a method for controlling a robot so that its endpoint cartesian stiffness matches a given stiffness matrix. Virtual model control can be used to implement Stiffness Control if linear virtual springs are used and connected to the robot in the necessary manner.

- *Impedance Control* [13]. Impedance control is a method for controlling a robot so that its impedance, as seen by the environment, matches a target impedance. In contrast, virtual model control is used to simulate components, which can be modelled as impedances, *between two points on the robot.* These two points can be attached anywhere on the robot, not just on the base and end effector.

- *Operational Space Formulation* [15]. The Operational Space Formulation is a framework for the analysis and control of manipulator systems with respect to the dynamic behavior of the end effector. Virtual model control is similar to Operational Space Formulation in that coordinate transformations are used in order to create a more intuitive space in which control is defined. However, the virtual model coordinate systems can be attached to any part of the robot, not just the end effector.

- *Coordinated Jacobian Transpose Control* [31], [2]. Coordinated Jacobian Transpose Control is an algorithm for coordinated position and force control for autonomous multi-limbed mobile robotic systems. Generalized control variables are defined with respect to a single inertial frame and controlled to a commanded position via a set of linear springs and dampers. Virtual model control allows for the use of multiple non-inertial frames for connecting components which aren't limited to springs and dampers. Also virtual model control allows for the specification of constraints (such as limp joints) and provides a framework for controlling parallel mechanisms.

Virtual model control is inspired by, and is a superset of the above control techniques. We stress the following major points,

- Virtual model control is an intuitive language for describing complex motion control tasks.
- Virtual model control allows for the use of generalized variables and generalized force functions.
- Virtual model control can be implemented on any serial link or set of serial links on the robot and not just between a base and end effector.
- Virtual model control can be implemented on serial or parallel, redundant or underactuated, fixed or free-flying robotic systems.
- Virtual model control allows one to specify mechanical constraints, such as unactuated joints, or design constraints, such as force equalization.
- Adaptive and learning techniques can be implemented with virtual models, thereby creating adaptive or learning virtual components.

Note that some of the above mentioned control techniques, along with other control schemes, use dynamical inversion in attempt to alter the behavior of the robot. We like to call this technique the "Virtual Robot" approach. Virtual model control can be used to simulate a virtual robot by adding virtual forces to counter gravity and virtual mass (negative or positive) at the center of mass of each link. However, this is not the intended application of virtual model control. We believe the following,

> The virtual robot approach (plant inversion) should only be used when high performance requirements or other extreme situations dictate. This is because,
>
> 1. Plant inversion adds computational complexity.
> 2. Fighting the natural dynamics of the robot can be inefficient.
> 3. Undesirable natural dynamics is an indication that the real robot was designed improperly.
> 4. Overcompensation can lead to instability.

Also note that with virtual model control, we usually talk in terms of spring set points, for example, and not *commanded* positions. Except for actuator non-idealities, we can perfectly implement virtual components whereas very few control algorithms can perfectly track a commanded trajectory. In this light we believe,

> Robots cannot be *commanded* to perform a task; they can only be given *hints* and *suggestions.*

1.2 Summary of Thesis Contents

This thesis is organized as follows:

Chapter 2 describes the notation and mathematics for implementing virtual models.

Chapter 3 explains several examples of virtual model formulation for various robot models.

Chapter 4 presents a simple example of when local control techniques, such as inverse kinematic control, fails but virtual model control succeeds.

Chapter 5 demonstrates, using Lyapunov analysis, that a passive robot with any number of passive virtual component connected to it will remain passive.

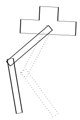
Chapter 6 describes how non-ideal actuators may affect virtual model control and proposes the relevant issues which must be addressed.

Chapter 7 defines the stability and performance requirement of virtual model controllers.

Chapter 8 analyzes a simple single input, single output (SISO), linear time-invariant (LTI) example of a virtual model controller being implemented with a non-ideal actuator.

Chapter 9 describes the application of virtual model controllers to bipedal dynamic walking.

Virtual model control is currently being used to control a dynamic walking bipedal robot, as described in Chapter 9. We have found that the robot can walk a number of steps using only a simple set of virtual components. Future work will focus on using virtual model control to produce more robust and efficient walking.



Chapter 2

Virtual Model Implementation

The implementation of virtual components is fairly simple. There are five major steps: definition of the virtual model reference frames; computation of the forward kinematics; calculation of a Jacobian matrix, computation of the joint torques; and definition of the virtual model force function.

Since forward kinematics of serial links produces closed form solutions for any robot with prismatic and revolute joints and since differentiation of closed form functions produces closed form functions, all the necessary equations will be relatively easy to derive and efficient to compute. For parallel virtual components, one must divide the generalized force amongst the individual serial paths. This requires solving a system of equations (i.e. inverting a matrix). Subsection 2.6.1 describes an extenstion of Gardner's Partitioned Actuator Set Control Technique method which minimizes the necessary computational requirements.

The derivation can all be done off-line and future work will focus on automating this process.

2.1 Definition of the Virtual Model Frames

Each virtual component requires three coordinate frames. These are denoted as

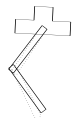
- Action Frame {B}
- Reaction Frame {A}
- Reference Frame {O}

where the frame symbols corresponds to those of Figure 2-1. The action frame defines the virtual component connection upon which the generalized forces act. The reaction frame defines the second attachment point of the virtual component. The reference frame is the coordinate system in which all displacements, forces, etc are expressed.

Choosing the right reaction frame is very important. 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 on a frame but must define a generalized force acting between two frames.

The action, reaction, and reference frames need not be inertial, nor cartesian, and none need to 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 reference frame {O} will either be inertial or coincide with {A} or {B}. In other words they will be connected where they make the most intuitive sense.

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.



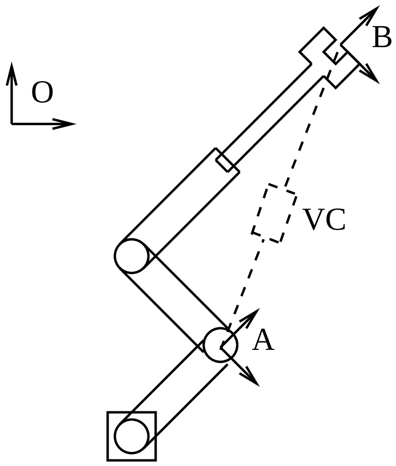


Figure 2-1: Virtual Component Reference Frames. {B} is the action frame. {A} is the reaction frame. {O} is the reference frame. VC represents the virtual component. The virtual component is drawn schematically to show it acts between frames {A} and {B}. The forces exerted by the virtual component can be in any admissible direction.

Choice of the virtual model frames is a major part of defining a virtual component and must be done carefully if the desired results are to be obtained. For example, both the action and reaction frames may be defined so that there is no 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 component's frames is physical intuition, insight, and experience, i.e. the same tools required to design the robot mechanism itself.

2.2 Derivation of the Forward Kinematics

Computing the forward kinematic map, ${}^A_B\vec{X}$, is relatively easy and well documented [7]. 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}. The robot link length parameters will enter this function as constant coefficients if frames {A} and {B} are rigidly attached to the links of the robot. This is not true in general, however.

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

$$ {}^O({}^A_B\vec{X}) = {}^O_A R\, {}^A_B\vec{X} \tag{2.1} $$

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

2.3 Derivation of the Jacobian Matrix

Let the Jacobian Matrix for a virtual component be defined in the following manner

$$ {}^A_B J = \frac{\partial}{\partial \Theta} {}^A_B\vec{X} \tag{2.2} $$

The Jacobian will be in closed form if ${}^A_B\vec{X}$ is in closed form. For the 2D case, the Jacobian can easily be computed by partial differentiation of the forward kinematic map. If the link parameters enter the forward kinematics as constant coefficients, then they'll also be constant coefficients in the Jacobian.

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

$$ {}^A_B\dot{\vec{X}} = {}^A_B J \, \dot{\vec{\Theta}} \tag{2.3} $$

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

$$ {}^O({}^A_B\dot{\vec{X}}) = {}^O_A R \, {}^A_B\dot{\vec{X}} \tag{2.4} $$

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 [19, 7]. One method described in [7] is to recursively compute the joint to cartesian velocity relationship (Equation 2.3) and extract the Jacobian Matrix.

2.4 Computation of the Joint Torques

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

$$ \vec{\tau} = {}^A_B J^T \, {}^A_B\vec{F} \tag{2.5} $$

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 2.5 can be easily derived starting from the energy balance $\vec{\tau}^T \delta\vec{\Theta} = \vec{F}^T \delta\vec{X}$ [7]. 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},

$$ {}^A_B\vec{F} = {}^A_O R \, {}^O({}^A_B\vec{F}) \tag{2.6} $$

We can combine Equations 2.5 and 2.6 to get

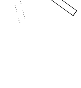
$$ \vec{\tau} = {}^A_B J^T \, {}^A_O R \, {}^O({}^A_B\vec{F}) = {}^O({}^A_B J)^T \, {}^O({}^A_B\vec{F}) \tag{2.7} $$

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 actuators. 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 in real life, 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 components, as in Section 2.6, it is important that the inadmissible force constraints be known for each of the serial paths of the virtual component so that the desired generalized force can be accurately divided amongst the individual serial paths.

2.5 Definition of the Generalized Forces

The final part of defining a virtual component is to find a relation between the relative displacement and rotations of the action and reaction frames and the generalized forces to be applied to the action



frame. In other words, we need to choose a generalized force function

$$^{O}({}^{A}_{B}\vec{F}) = {}^{O}({}^{A}_{B}\vec{F}({}^{O}({}^{A}_{B}\vec{X}), \vec{v}, \vec{s})) \tag{2.8}$$

where ${}^{A}_{B}\vec{F}$ is the generalized forces acting on action frame {B} from reaction frame {A}, ${}^{A}_{B}\vec{X}$ is the generalized displacement of frame {B} from frame {A} computed in section 2.2, $\vec{v}$ are other parameters and variables, and $\vec{s}$ is the state of dynamic simulated virtual entities such as masses and inertias. The O superscripts denote that all measurements are defined with respect to reference frame {O}. If virtual components which have state are used, then one must implement a real-time dynamic simulation to update the component's state.

As an example, to implement a generalized spring-damper mechanism, one could use

$$^{O}({}^{A}_{B}\vec{F}) = k({}^{O}({}^{A}_{B}\tilde{\vec{X}})) - b({}^{O}({}^{A}_{B}\dot{\vec{X}})) \tag{2.9}$$

where ${}^{O}({}^{A}_{B}\tilde{\vec{X}})$ is the error between the spring set-point and the actual position and k and b are the spring and damper functions.

Of course, the virtual component need not be just a spring or damper; any component which can be expressed in terms of Equation 2.8 is valid. The simulated mechanism need not be linear, monotonic, nor even physically realizable. In fact, one could simply treat the virtual component as a virtual actuator and produce the virtual forces by any desired control method.

2.6 Parallel Mechanisms and Multi-Frame Virtual Components

Up to this point we have only dealt with serial link manipulators. With a serial link structure all the necessary equations are relatively simple to derive. With a parallel structure a matrix inversion is necessary.

We start by defining one action frame {B}, one reference frame {O} and multiple reaction frames $\{A_i\}$ as in Figure 2-2. 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 component structure can be considered as multiple serial structures acting on the same action frame. Each serial sub-component has a corresponding Jacobian which is calculated as in Section 2.3. We combine these to get

$$\begin{bmatrix} \vec{\tau}_1 \\ \vec{\tau}_2 \\ \vdots \\ \vec{\tau}_p \end{bmatrix} = \begin{bmatrix} {}^{O}({}^{A_1}_{B}J)^T & 0 & \cdots & 0 \\ 0 & {}^{O}({}^{A_2}_{B}J)^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & {}^{O}({}^{A_p}_{B}J)^T \end{bmatrix} \begin{bmatrix} {}^{O}({}^{A_1}_{B}\vec{F}) \\ {}^{O}({}^{A_2}_{B}\vec{F}) \\ \vdots \\ {}^{O}({}^{A_p}_{B}\vec{F}) \end{bmatrix} \tag{2.10}$$

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,

$$\sum_{i=1}^{p} {}^{O}({}^{A_i}_{B}\vec{F}) = {}^{O}\ ({}^{A^*}_{B}\vec{F}) \tag{2.11}$$

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,

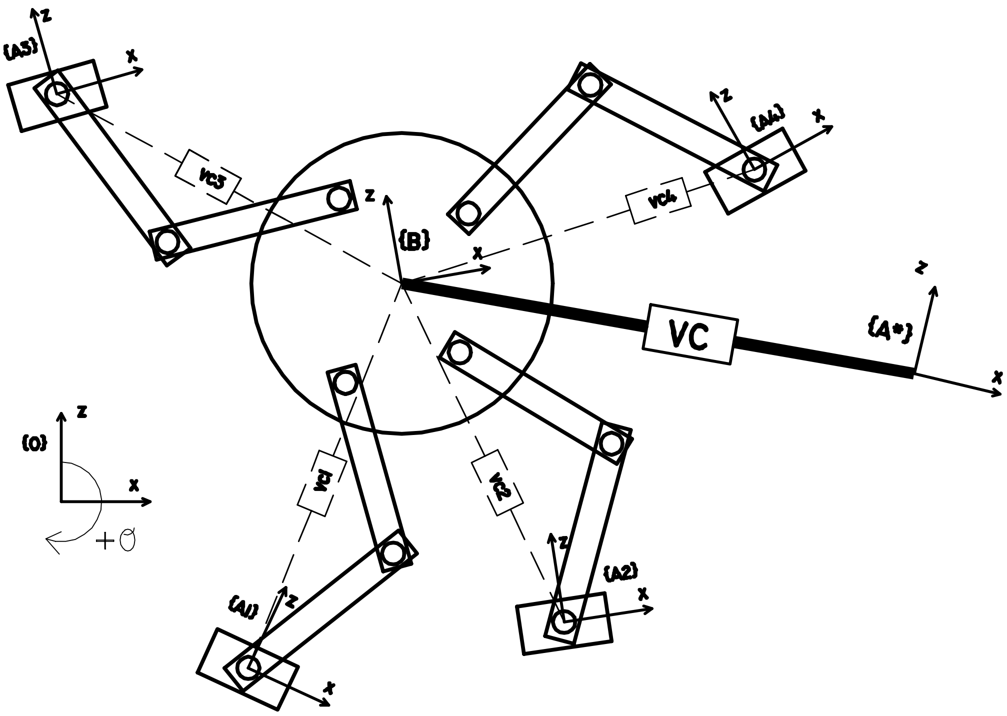


Figure 2-2: Parallel Virtual Component Reference Frames. Frame {B} is the action frame. Frames $\{A_i\}$ are the reaction frames. Frame {O} is the reference frame. VC represents the conglomerate virtual component which is comprised of the individual virtual components VC_i. Frame $\{A^*\}$ is an imaginary construct which represents the reaction frame of the conglomerate virtual component.

K, so that the constraints can be written in the form

$$\begin{bmatrix} {}^{O}({}^{A^*}_{B}\vec{F}) \\ \vec{c} \end{bmatrix} = K \begin{bmatrix} {}^{O}({}^{A_1}_{B}\vec{F}) \\ {}^{O}({}^{A_2}_{B}\vec{F}) \\ \vdots \\ {}^{O}({}^{A_p}_{B}\vec{F}) \end{bmatrix} \tag{2.12}$$

and the individual sub-force vectors resolved as,

$$\begin{bmatrix} {}^{O}({}^{A_1}_{B}\vec{F}) \\ {}^{O}({}^{A_2}_{B}\vec{F}) \\ \vdots \\ {}^{O}({}^{A_p}_{B}\vec{F}) \end{bmatrix} = K^{-1} \begin{bmatrix} {}^{O}({}^{A^*}_{B}\vec{F}) \\ \vec{c} \end{bmatrix} \tag{2.13}$$

where 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 Equation 2.13 into 2.10 to get the final force to torque relationship.

2.6.1 Inverting the Constraint Matrix

Equation 2.13 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 Actuator Set Control Technique [11, 10].

Gardner partitions the actuators into a Minimum Actuator Set (MAS) and a Redundant Actuator Set (RAS). We stress here that we are not dealing with actuators at this level but rather virtual force distribution. Therefore we specify the Minimum Force Set (MFS) and the 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 components 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 component
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 components

The number of force elements in the Minimum Force Set will be n; in the Constrained Actuator 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. These limitations are explained below.

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 2.12 into the following form,

$$\begin{bmatrix} f^a_d \\ f^b_l \\ O_l \\ O_l \\ \vdots \\ O_l \\ c_r \end{bmatrix} = \begin{bmatrix} I_{d\times d} & 0_{d\times l} & I_{d\times d} & 0_{d\times l} & \cdots & I_{d\times d} & 0_{d\times l} \\ 0_{l\times d} & I_{l\times l} & 0_{l\times d} & I_{l\times l} & \cdots & 0_{l\times d} & I_{l\times l} \\ J^{1a}_{l\times d} & J^{1b}_{l\times l} & 0_{l\times d} & 0_{l\times l} & \cdots & 0_{l\times d} & 0_{l\times l} \\ 0_{l\times d} & 0_{l\times l} & J^{2a}_{l\times d} & J^{2b}_{l\times l} & \cdots & 0_{l\times d} & 0_{l\times l} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0_{l\times d} & 0_{l\times l} & 0_{l\times d} & 0_{l\times l} & \cdots & J^{pa}_{l\times d} & J^{pb}_{l\times l} \\ D^{1a}_{r\times d} & D^{1b}_{r\times l} & D^{2a}_{r\times d} & D^{2b}_{r\times l} & \cdots & D^{pa}_{r\times d} & D^{pb}_{r\times l} \end{bmatrix} \begin{bmatrix} f^{1a}_d \\ f^{1b}_l \\ f^{2a}_d \\ f^{2b}_l \\ \vdots \\ f^{pa}_d \\ f^{pb}_l \end{bmatrix} \tag{2.14}$$

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. Equation 2.12 can always be written in the form of Equation 2.14 since the natural constraints for a given serial path will only be a function of the virtual forces on that path.

We can reduce the size of the matrix in equation 2.14 by taking advantage of the sparseness of the natural constraint blocks. Each natural constraint row can be written as

$$0 = J^{ia} f^{ia} + J^{ib} f^{ib}$$

Solving for f^{ib} we have,

$$f^{ib} = -(J^{ib})^{-1} J^{ia} f^{ia} \tag{2.15}$$

We can substitute this back into Equation 2.14 to get a reduced set of equations,

$$\begin{bmatrix} f^a_d \\ f^b_l \\ c_r \end{bmatrix} = \begin{bmatrix} I_{d\times d} & I_{d\times d} & \cdots & I_{d\times d} \\ [-(J^{1b})^{-1} J^{1a}]_{l\times d} & [-(J^{2b})^{-1} J^{2a}]_{l\times d} & \cdots & [-(J^{pb})^{-1} J^{pa}]_{l\times d} \\ [-D^{1b}(J^{1b})^{-1} J^{1a} + D^{1a}]_{r\times d} & [-D^{2b}(J^{2b})^{-1} J^{2a} + D^{2a}]_{r\times d} & \cdots & [-D^{pb}(J^{pb})^{-1} J^{pa} + D^{pa}]_{r\times d} \end{bmatrix} \begin{bmatrix} f^{1a}_d \\ f^{2a}_d \\ \vdots \\ f^{pa}_d \end{bmatrix} \tag{2.16}$$

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 2.16 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.

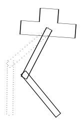
Similar to [11], we specify the Redundant Force Set, f^r, in terms of the Minimum Force Set, f^m.

$$f^r_r = c_r - B_{r\times n} f^m_n \tag{2.17}$$

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 constrain matrix.

Writing the design constraints in terms of Equation 2.17 requires that

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



and D^{ia} are restricted so that we can rearrange Equation 2.16 into the following form,

$$\begin{bmatrix} f_n \\ c_r \end{bmatrix} = \begin{bmatrix} A^m_{n\times n} & A^r_{n\times r} \\ B_{r\times n} & I_{r\times r} \end{bmatrix} \begin{bmatrix} f^m_n \\ f^r_r \end{bmatrix} \tag{2.18}$$

We can now eliminate the RFS, f^r, from Equation 2.18 to by substituting Equation 2.17 into 2.18 to get

$$[f - A^r c]_n = [A^m - A^r B]_{n\times n}[f^m_n] \tag{2.19}$$

To solve for the MFS, f^m, we must invert the $n \times n$ matrix in Equation 2.19. The Redundant Force Set is then computed by plugging back into equation 2.17. Finally, we can rearrange the Minimum and Redundant Force Sets to get f^{ia} and substitute back into Equation 2.15 to solve for the Constrained Force Set.

We now describe how one partitions the virtual forces into the three sets. In order to use the above method, one must write all the design constraints in the form of Equation 2.17. 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 2.19 to exist.

Examining Equations 2.16 to 2.19 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 $O(p^3n^3)$.

2.6.2 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 2.18. Suppose we wish to specify only s design constraints, where $s < r$. If we specify the design constraints in the form,

$$f^{r2}_s = c - B^1 f^m - B^2 f^{r1} \tag{2.20}$$

then instead of Equation 2.18, we will have,

$$\begin{bmatrix} [f - A^{r2}c]_n \\ c_s \end{bmatrix} = \begin{bmatrix} A^m_{n\times n} & A^{r1}_{n\times(r-s)} & A^{r2}_{n\times s} \\ B^1_{s\times n} & B^2_{s\times(r-s)} & I_{s\times s} \end{bmatrix} \begin{bmatrix} f^m_n \\ f^{r1}_{r-s} \\ f^{r2}_s \end{bmatrix} \tag{2.21}$$

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

$$\begin{bmatrix} f_n \end{bmatrix} = \begin{bmatrix} [A^m - A^{r2}B^1]_{n\times n} & [A^{r1} - A^{r2}B^2]_{n\times(r-s)} \end{bmatrix} \begin{bmatrix} f^m_n \\ f^{r1}_{r-s} \end{bmatrix} \tag{2.22}$$

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,

$$A^+ = A^T(AA^T)^{-1} \tag{2.23}$$

and then solve for f^{r2} by substituting back into Equation 2.20.

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

2.7 Summary

The implementation of a virtual component is relatively straightforward. The math may become cumbersome but equation derivation need be done only once for a given mechanism. For a serial mechanism, computational requirements are low as no functional or matrix inversions are required. For a parallel mechanism, a matrix inversion is necessary. However, by taking advantage of the form of the constraint matrix and by specifying the design constraints in the correct manner, the size of the matrices which need to be inverted are reduced.

The most important part of creating a virtual component is that the virtual components be defined properly via the three frames and the generalized force function. This can be accomplished by using the same physical intuition and mechanism design approach used in constructing the robot. Therefore, virtual model control is as suited for someone who is comfortable with robot design as someone who is comfortable with control systems theory.

Chapter 3

Examples

In this Chapter we present several robot examples in which virtual models are used. We do not discuss the virtual components which are used but rather derive and discuss the relevant mathematics for each of the examples. These examples will be used in later chapters at which point actual virtual components will be used to do something interesting. For instance the foot example mathematics is derived in Section 3.3 and used in Chapter 4 as a simple example in which virtual model control works but blind application of inverse kinematics control fails.

3.1 Single Leg Example

Figure 3-1 shows a simple 2-D, four link, three joint, serial robot model which we use to represent a single leg of our walking robot (See Chapter 9). We wish to connect a virtual component between frame {A}, which is attached to the foot, and frame {B}, which is attached to the body. The angles θ_a, θ_k, and θ_h are those of the ankle, knee, and hip. The lower link (tibia) is of length L_1, while the upper link (femur) is of length L_2. In this example we assume that the foot is flat on the ground, so that ${}^{O}_{A}R = I$.

The forward kinematic map from frame {A} to frame {B} of this example is as follows,

$$ {}^{A}_{B}\vec{X} = \begin{bmatrix} x \\ z \\ \theta \end{bmatrix} = \begin{bmatrix} -L_1 \sin(\theta_a) - L_2 \sin(\theta_a + \theta_k) \\ L_1 \cos(\theta_a) + L_2 \cos(\theta_a + \theta_k) \\ -\theta_h - \theta_k - \theta_a \end{bmatrix} \tag{3.1} $$

Partial differentiation produces the Jacobian,

$$ {}^{A}_{B}J = \begin{bmatrix} -L_1 \cos(\theta_a) - L_2 \cos(\theta_a + \theta_k) & -L_2 \cos(\theta_a + \theta_k) & 0 \\ -L_1 \sin(\theta_a) - L_2 \sin(\theta_a + \theta_k) & -L_2 \sin(\theta_a + \theta_k) & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{3.2} $$

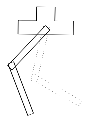
The Jacobian relates the virtual velocity between frames A and B,

$$ {}^{A}_{B}\dot{\vec{X}} = {}^{A}_{B}J\,\dot{\vec{\Theta}} \tag{3.3} $$

and the virtual force to joint torque,

$$ \vec{\tau} = ({}^{A}_{B}J)^T\ ({}^{A}_{B}\vec{F}) \tag{3.4} $$

The Jacobian is of full rank, indicating that all virtual force directions are admissible. This is the typical setup for using the Jacobian to transfer from endpoint forces to joint torques. To keep things interesting, we add the constraint that $\tau_a = 0$. An unactuated ankle will constrain the direction in which virtual forces can be applied. This constraint is equivalent to having a point foot since this



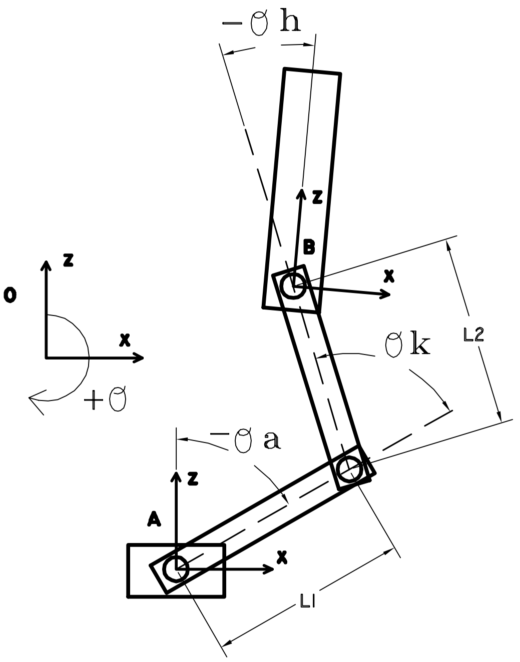


Figure 3-1: Single Leg example. Reaction frame {A} is assumed to be in the same orientation as reference frame {O} so that ${}^{O}_{A}R = I$.

example is 2-D. With a limp ankle, Equation 3.4 is constrained,

$$\begin{bmatrix} 0 \\ \tau_k \\ \tau_h \end{bmatrix} = \begin{bmatrix} -L_1\cos(\theta_a) - L_2\cos(\theta_a+\theta_k) & -L_1\sin(\theta_a) - L_2\sin(\theta_a+\theta_k) & -1 \\ -L_2\cos(\theta_a+\theta_k) & -L_2\sin(\theta_a+\theta_k) & -1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_x \\ f_z \\ f_\theta \end{bmatrix} \tag{3.5}$$

For our walking robot we are more concerned about applying forces in the vertical direction and torques about the body then we are concerned about applying horizontal forces. Therefore, we specify f_z and f_θ and solve for f_x

$$f_x = \frac{-1}{L_1\cos(\theta_a) + L_2\cos(\theta_a+\theta_k)} \begin{bmatrix} L_1\sin(\theta_a) + L_2\sin(\theta_a+\theta_k) & 1 \end{bmatrix} \begin{bmatrix} f_z \\ f_\theta \end{bmatrix} \tag{3.6}$$

Plugging this back into 3.5, we get

$$\begin{bmatrix} \tau_k \\ \tau_h \end{bmatrix} = \begin{bmatrix} \frac{-L_1L_2\sin(\theta_k)}{L_1\cos(\theta_a)+L_2\cos(\theta_a+\theta_k)} & \frac{-L_1\cos(\theta_a)}{L_1\cos(\theta_a)+L_2\cos(\theta_a+\theta_k)} \\ 0 & -1 \end{bmatrix} \begin{bmatrix} f_z \\ f_\theta \end{bmatrix} \tag{3.7}$$

We now have a simple set of equations for determining joint torques given virtual forces. Note that the matrix in Equation 3.7 is of full rank for all values of $\vec{\Theta}$ except for $\theta_k = 0$. This corresponds to a fully extended knee, during which no virtual forces can be applied in the z direction. If the knee is not fully extended, all virtual forces are admissible. Also note that the denominator in Equation 3.7, $L_1\cos(\theta_a) + L_2\cos(\theta_a+\theta_k)$, is ${}^{A}_{B}z$ and hence is already computed in the forward kinematics. Throughout this example we have assumed that the feet are flat on the ground and that we can measure all angles. In actuality, we use point feet and measure the body angle via a boom or gyroscope, rather than the ankle. Therefore, we must make the substitution

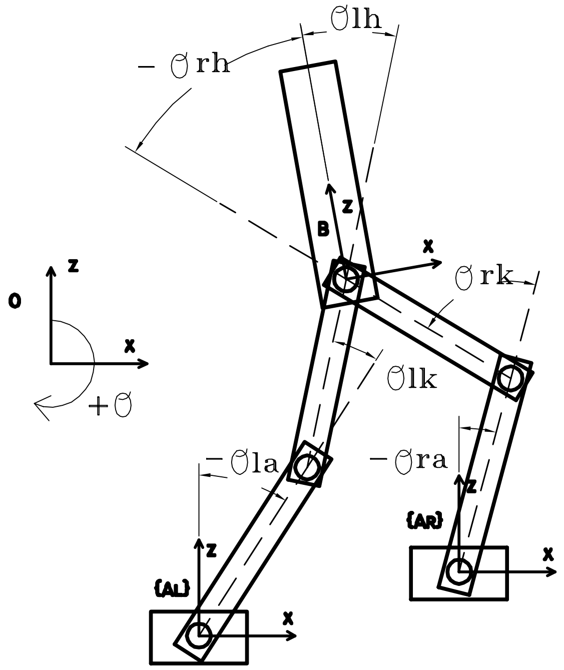


Figure 3-2: Dual Leg example. Reaction frames {A} and {C} are assumed to be in the same orientation as reference frame {O} so that ${}^{O}_{A_l}R = {}^{O}_{A_r}R = I$.

$$\theta_a = -\theta - \theta_h - \theta_k$$

These equations will be used in Chapter 9 in the control of a bipedal walking robot during single support phase.

3.2 Dual Leg Example

The previous example discussed a serial chain manipulator. Here we examine a parallel mechanism representing a simple, 2-D, bipedal robot (See Figure 3-2). As discussed in Chapter 2, we can attach single-frame virtual components across any serial link or we can create a multi-frame virtual component with a common action frame. We describe the latter here.

Our model consists of the previous single leg example plus another leg. We wish to connect a multi-frame virtual component between the reaction frames $\{A_l\}$ and $\{A_r\}$ which are connected to the feet, and the action frame {B} which is connected to the body. The individual leg parameters and joint angles are identical to those of the single leg example with the l subscript denoting the left leg and the r subscript denoting the right leg. Again, we assume that the feet are flat on the ground so that ${}^{O}_{A_l}R = {}^{O}_{A_r}R = I$.

We already have the kinematics for each leg from the previous example. To calculate the body kinematics, we choose to use the average of these,

$$\vec{X} = \frac{{}^{A_l}_{B}\vec{X} + {}^{A_r}_{B}\vec{X}}{2}$$

$$\dot{\vec{X}} = \frac{{}^{A_l}_{B}\dot{\vec{X}} + {}^{A_r}_{B}\dot{\vec{X}}}{2}$$

One could also choose the minimum, maximum, etc. or simply keep them separate.

We have computed the Jacobian for each serial link of this parallel mechanism in the previous example. We now combine them in the following manner,

$$\begin{bmatrix} \vec{\tau_l} \\ \vec{\tau_r} \end{bmatrix} = \begin{bmatrix} {}^{A_l}_{B}J^T & 0 \\ 0 & {}^{A_r}_{B}J^T \end{bmatrix} \begin{bmatrix} \vec{F_l} \\ \vec{F_r} \end{bmatrix} \tag{3.8}$$

This expands to

$$\begin{bmatrix} \tau_{la} \\ \tau_{lk} \\ \tau_{lh} \\ \tau_{ra} \\ \tau_{rk} \\ \tau_{rh} \end{bmatrix} = \begin{bmatrix} A & B & -1 & 0 & 0 & 0 \\ Q & R & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & C & D & -1 \\ 0 & 0 & 0 & S & T & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_{xl} \\ f_{zl} \\ f_{\theta l} \\ f_{xr} \\ f_{zr} \\ f_{\theta r} \end{bmatrix} \tag{3.9}$$

where

$$A = -L_1 \cos(\theta_{la}) - L_2 \cos(\theta_{la} + \theta_l k)$$

$$B = -L_1 \sin(\theta_{la}) - L_2 \sin(\theta_{la} + \theta_l k)$$

$$C = -L_1 \cos(\theta_{ra}) - L_2 \cos(\theta_{ra} + \theta_{rk})$$

$$D = -L_1 \sin(\theta_{ra}) - L_2 \sin(\theta_{ra} + \theta_{rk})$$

$$Q = -L_2 \cos(\theta_{la} + \theta_{lk})$$

$$R = -L_2 \sin(\theta_{la} + \theta_{lk})$$

$$S = -L_2 \cos(\theta_{ra} + \theta_{rk})$$

$$T = -L_2 \sin(\theta_{ra} + \theta_{rk})$$

Equation 3.9 maps the virtual forces from each leg to the required joint torques, whereas we wish to specify a single virtual force. We therefore need to solve the individual leg forces in terms of the combined virtual force subject to several constraints.

Since the action frame {B} is coincidental, we have the compatibility relation that the force vector must equal the vector sum of the forces produced by each serial chain,

$$\begin{bmatrix} f_x \\ f_z \\ f_\theta \end{bmatrix} = \begin{bmatrix} f_{xl} \\ f_{zl} \\ f_{\theta l} \end{bmatrix} + \begin{bmatrix} f_{xr} \\ f_{zr} \\ f_{\theta r} \end{bmatrix} \tag{3.10}$$

Since we have six joints and wish to control three force directions, we require three constraints. Unactuated ankles provide two constraints,

$$\tau_{la} = 0 \tag{3.11}$$

$$\tau_{ra} = 0 \tag{3.12}$$

The third constraint provides us with a design degree of freedom. We could choose it to maximize some performance criterion, etc. Here we simply choose to match the hip torques,

$$\tau_{lh} = \tau_{rh} \Longrightarrow f_{\theta l} = f_{\theta r} \tag{3.13}$$

Putting the above constraints in vector form we have,

$$\begin{bmatrix} f_x \\ f_z \\ f_\theta \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ A & B & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & C & D & -1 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_{xl} \\ f_{zl} \\ f_{\theta l} \\ f_{xr} \\ f_{zr} \\ f_{\theta r} \end{bmatrix} \tag{3.14}$$

We must now perform a 6 by 6 matrix inversion to solve for the individual leg forces. We drop the terms which are multiplied by zero. The result is a 6 by 3 matrix relating the single vector of virtual forces to the individual leg virtual forces,

$$\begin{bmatrix} f_{xl} \\ f_{zl} \\ f_{\theta l} \\ f_{xr} \\ f_{zr} \\ f_{\theta r} \end{bmatrix} = \begin{bmatrix} \frac{CB}{E} & \frac{DB}{E} & -\frac{B+D}{2E} \\ -\frac{CA}{E} & -\frac{AD}{E} & \frac{C+A}{2E} \\ 0 & 0 & 1/2 \\ -\frac{AD}{E} & -\frac{DB}{E} & \frac{B+D}{2E} \\ \frac{CA}{E} & \frac{CB}{E} & -\frac{C+A}{2E} \\ 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} f_x \\ f_z \\ f_\theta \end{bmatrix} \tag{3.15}$$

where

$$E = CB - AD \tag{3.16}$$

Plugging Equation 3.15 into Equation 3.9 and simplifying, we get the virtual force to joint torque relation

$$\begin{bmatrix} \tau_{lk} \\ \tau_{lh} \\ \tau_{rk} \\ \tau_{rh} \end{bmatrix} = \begin{bmatrix} \frac{CV}{E} & \frac{DV}{E} & \frac{-V-QD+RC}{2E} - \frac{1}{2} \\ 0 & 0 & -1/2 \\ \frac{-AW}{E} & \frac{-BW}{E} & \frac{W+SB-TA}{2E} - \frac{1}{2} \\ 0 & 0 & -1/2 \end{bmatrix} \begin{bmatrix} f_x \\ f_z \\ f_\theta \end{bmatrix} \tag{3.17}$$

where

$$V = QB - RA = -L_1 L_2 \sin(\theta_{lk})$$

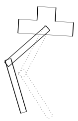
$$W = SD - TC = -L_1 L_2 \sin(\theta_{rk})$$

Once again, we have a simple set of equations for relating virtual forces to joint torques. Intuitively, the matrix in Equation 3.17 should be of full rank for all Θ except when a knee is fully extended or the two feet and hip are colinear. In all other configurations, all virtual forces are admissible. Again, we will use point feet and measure the body angle via a boom or gyroscope, rather than the ankle angles. Therefore, we must make the substitutions

$$\theta_{la} = -\theta - \theta_{lh} - \theta_{lk}$$

$$\theta_{ra} = -\theta - \theta_{rh} - \theta_{rk}$$

These equations will be used in Chapter 9 in the control of a bipedal walking robot during double support phase.



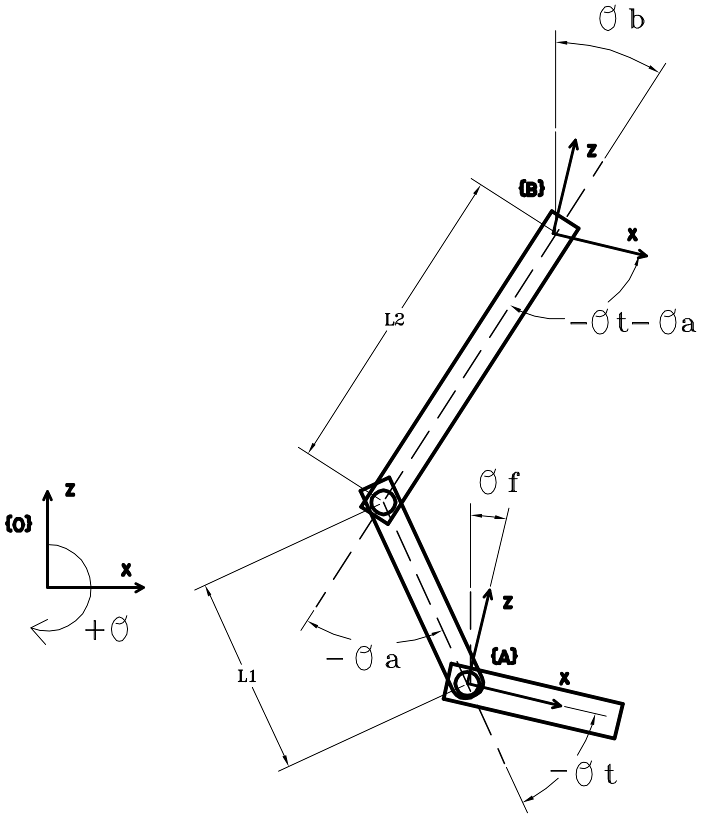


Figure 3-3: Foot example. θ_t is the toe angle, θ_a is the ankle angle, θ_f is the angle of the foot with respect to frame {O}, and θ_b is the angle of the body with respect to frame {O}.

3.3 Foot Example

Figure 3-3 shows a simple 2-D, three link, two joint, serial robot model which represents a foot. Unlike the previous examples, we do not assume that the last link is flat on the ground. We wish to reference all vectors to inertial frame {O}.

The kinematics between frames {A} and {B} are easily calculated

$$\begin{bmatrix} {}^A_B x \\ {}^A_B z \end{bmatrix} = \begin{bmatrix} -L_2 \cos(\theta_t + \theta_a) - L_1 \cos(\theta_t) \\ -L_2 \sin(\theta_t + \theta_a) - L_1 \sin(\theta_t) \end{bmatrix} \tag{3.18}$$

Partial differentiation produces the Jacobian,

$${}^A_B J = \begin{bmatrix} L_2 \sin(\theta_t + \theta_a) + L_1 \sin(\theta_t) & L_2 \sin(\theta_t + \theta_a) \\ -L_2 \cos(\theta_t + \theta_a) - L_1 \cos(\theta_t) & -L_2 \cos(\theta_t + \theta_a) \end{bmatrix} \tag{3.19}$$

We use the rotation matrix between frames {O} and {A} to transform the kinematics,

$${}^O_A R = \begin{bmatrix} \cos(\theta_f) & \sin(\theta_f) \\ -\sin(\theta_f) & \cos(\theta_f) \end{bmatrix} \tag{3.20}$$

$${}^O({}^A_B \vec{X}) = {}^O_A R \, {}^A_B \vec{X} = \begin{bmatrix} -L_2 \cos(\theta_f - \theta_a - \theta_t) - L_1 \cos(\theta_f - \theta_t) \\ L_2 \sin(\theta_f - \theta_a - \theta_t) + L_1 \sin(\theta_f - \theta_t) \end{bmatrix} \tag{3.21}$$

$$
\begin{aligned}
{}^{O}({}^{A}_{B}\vec{\dot{X}}) &= {}^{O}_{A}R \; {}^{A}_{B}J \; \vec{\dot{\Theta}} \\
&= \begin{bmatrix} -L_2 \sin(\theta_f - \theta_a - \theta_t) - L_1 \sin(\theta_f - \theta_t) & -L_2 \sin(\theta_f - \theta_a - \theta_t) \\ -L_2 \cos(\theta_f - \theta_a - \theta_t) - L_1 \cos(\theta_f - \theta_t) & -L_2 \cos(\theta_f - \theta_a - \theta_t) \end{bmatrix} \vec{\dot{\Theta}} \quad (3.22)
\end{aligned}
$$

We use the rotation matrix between frames {A} and {O} to transform the virtual forces,

$$
{}^{A}_{O}R = \begin{bmatrix} \cos(\theta_f) & -\sin(\theta_f) \\ \sin(\theta_f) & \cos(\theta_f) \end{bmatrix} \quad (3.23)
$$

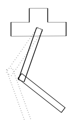
$$
\begin{aligned}
\vec{\tau} &= {}^{A}_{B}J^{T} \; {}^{A}_{O}R \; {}^{O}({}^{A}_{B}\vec{F}) \\
&= \begin{bmatrix} -L_2 \sin(\theta_f - \theta_a - \theta_t) - L_1 \sin(\theta_f - \theta_t) & -L_2 \cos(\theta_f - \theta_a - \theta_t) - L_1 \cos(\theta_f - \theta_t) \\ -L_2 \sin(\theta_f - \theta_a - \theta_t) & -L_2 \cos(\theta_f - \theta_a - \theta_t) \end{bmatrix} {}^{O}({}^{A}_{B}\vec{F})
\end{aligned}
$$

(3.24)

The determinant of the matrix in Equation 3.24 is $L_1 L_2 \sin(\theta_a)$. Therefore, all forces will be admissible until the ankle joint is fully extended, which makes intuitive sense. Since the body angle, rather than the foot angle, will be measured via a boom or a gyroscope we need to make the substitution

$$
\theta_f = \theta_b + \theta_t + \theta_a
$$

This simple example will be used in Chapter 4 as a case when inverse kinematics fails while virtual models succeed.



3.4 3D Hexapod Example

The previous examples have all been two dimensional. This section presents a three dimensional example of a hexapod robot alternately being supported by a tripod composed of three of its six legs. Using three legs for support allows the hexapod to be statically stable and is a typical snapshot of the tripod walking gait.

In the two dimensional examples, the Jacobians were computed by differentiation of the kinematics. In contrast, the Jacobian for this three dimensional example is computed using a technique found in [7]. In this technique, the mapping between joint velocity and virtual model frame velocity is directly computed. The Jacobian is then extracted from this mapping.

Since the multiple legs operate as a parallel mechanism, techniques described in Section 2.6 applicable and thus used to determine the force distribution among the three supporting legs.

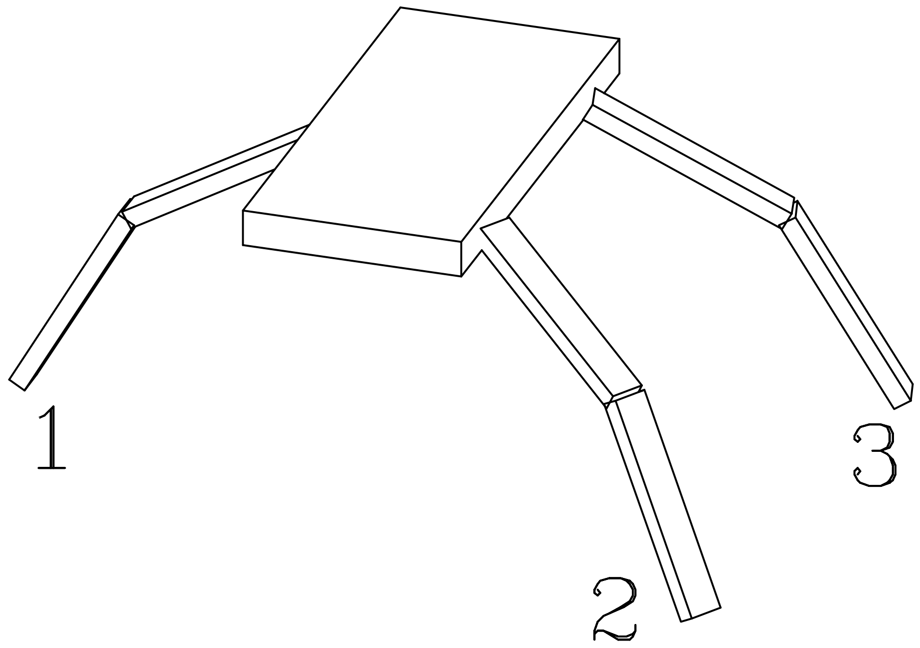


Figure 3-4: Drawing of hexapod, used in the 3D example, standing on three of its six legs. The other three legs are not shown for clarity.

Figure 3-4 shows the hexapod model standing on three legs. Leg 1 is in the middle of one side of the body while legs 2 and 3 are in the front and back of the opposite side of the body. The other three legs of the hexapod are not shown. Figure 3-5 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_i} and the hip angles are θ_{h1_i} and θ_{h2_i}. If the joint angles are all zero, the leg will point straight down from the body. Each leg is comprised of an upper and lower link, both of length L. The location of the leg with respect to the action frame {B} is $(P_x, P_y, 0)$. We assume that we can measure only the angles of the three joints of each of the three legs.

We wish to specify a generalized force, ${}^{B}({}^{A^*}_{B}F)$ which acts on the body from the three legs. As described in Chapter 2, we must determine the mapping from virtual forces to joint torques and the mapping from a generalized force to the individual virtual forces applied by each leg.

The Jacobian from the reaction frame to the action frame for a given leg, ${}^{B}({}^{A_i}_{B}J)$ is determined via an iterative computation of the joint velocity to cartesian velocity mapping, as described in [7]. This Jacobian maps the virtual model generalized forces to joint torques for the ith leg,

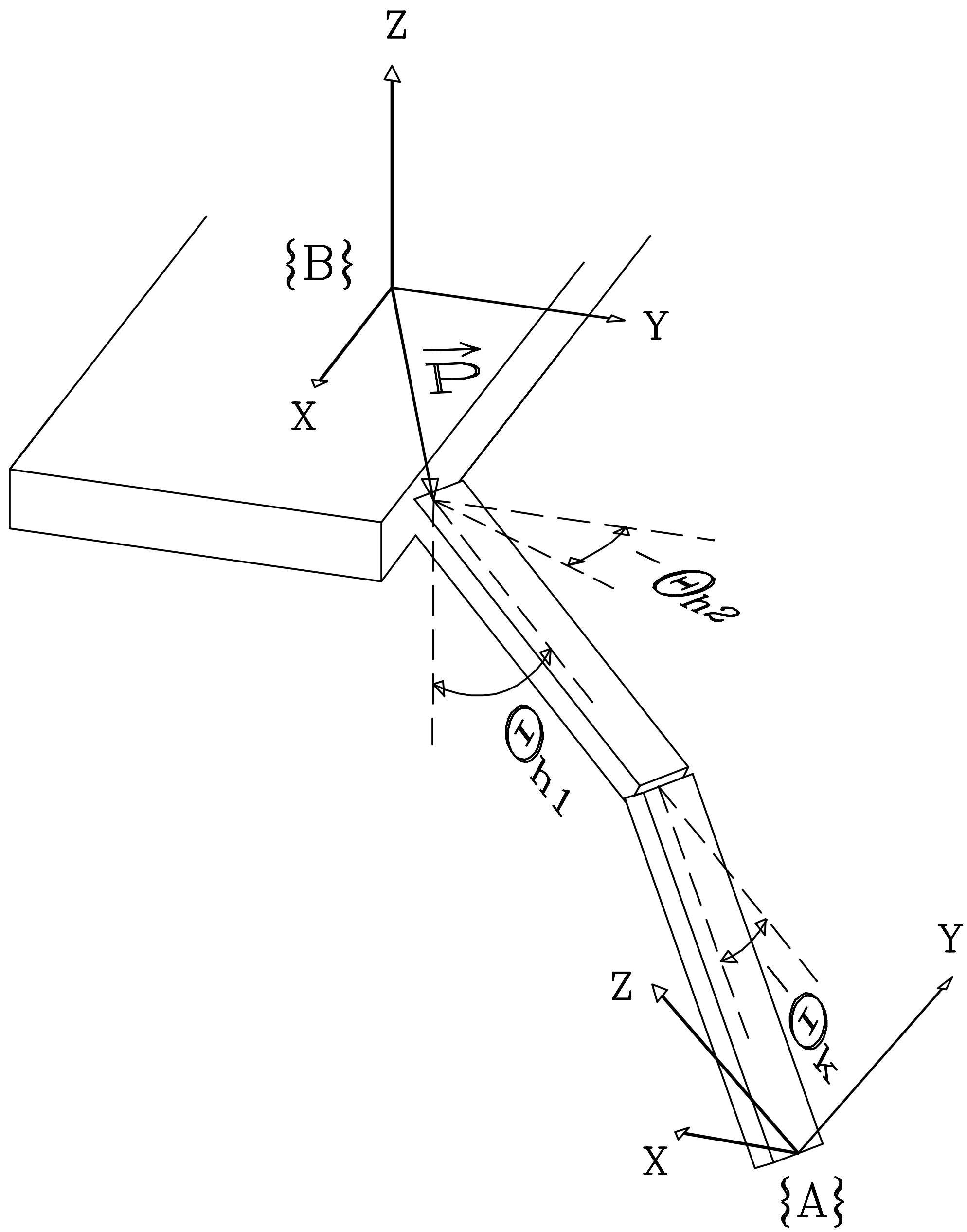
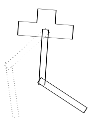


Figure 3-5: Diagram of a single leg of the hexapod example. There is one actuated degree of freedom at the knee and two at the hip. The knee angle is θ_k and the hip angles are θ_{h1} and θ_{h2}. The leg links are both of length L. The location of the leg with respect to frame {B} is $(P_x,\, P_y,\, 0)$.



$$
\begin{bmatrix} \tau_{k_i} \\ \tau_{h1_i} \\ \tau_{h2_i} \end{bmatrix} = \begin{bmatrix} -L\, s_{h2_i} c_{h1_i} & L\, c_{h2_i} c_{h1_i} & L\, s_{h1_i} - P_{x_i}\, s_{h2_i} + P_{y_i}\, c_{h2_i} & -c_{h2_i} & -s_{h2_i} & 0 \\ 0 & 0 & -P_{x_i}\, s_{h2_i} + P_{y_i}\, c_{h2_i} & -c_{h2_i} & -s_{h2_i} & 0 \\ -P_{y_i} & +P_{x_i} & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \\ n_{x_i} \\ n_{y_i} \\ n_{z_i} \end{bmatrix} \tag{3.25}
$$

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

Since no moments can be applied at the point feet each leg has three natural constraints. Following the method in Section 2.6.1 we have

$$n = 6 \tag{3.26}$$
$$p = 3 \tag{3.27}$$
$$l = 3 \tag{3.28}$$
$$d = 3 \tag{3.29}$$
$$r = 3 \tag{3.30}$$

To reduce the size of the matrix to be inverted, we must partition the individual components of the virtual forces into the Minimum Force Set (6 elements), Redundant Force Set (3 elements), and Constrained Force Set (9 elements). For our 3 design constraints, we decide to match the horizontal forces of legs 2 and 3 and match the lateral force of leg 1 with the sum of the vertical forces of legs 2 and 3,

$$f_{x3} = f_{x2} \tag{3.31}$$
$$f_{y3} = f_{y2} \tag{3.32}$$
$$f_{y1} = 2 f_{y2} \tag{3.33}$$

These design constraints are written in the terms of Equation 2.17 if we partition the virtual forces 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 constraint equation (2.12) can now be written in expanded form,

$$
\begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
=
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
J^{1a}_{1,1} & J^{1a}_{1,2} & J^{1a}_{1,3} & J^{1b}_{1,1} & J^{1b}_{1,2} & J^{1b}_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
J^{1a}_{2,1} & J^{1a}_{2,2} & J^{1a}_{2,3} & J^{1b}_{2,1} & J^{1b}_{2,2} & J^{1b}_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
J^{1a}_{3,1} & J^{1a}_{3,2} & J^{1a}_{3,3} & J^{1b}_{3,1} & J^{1b}_{3,2} & J^{1b}_{3,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & J^{2a}_{1,1} & J^{2a}_{1,2} & J^{2a}_{1,3} & J^{2b}_{1,1} & J^{2b}_{1,2} & J^{2b}_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & J^{2a}_{2,1} & J^{2a}_{2,2} & J^{2a}_{2,3} & J^{2b}_{2,1} & J^{2b}_{2,2} & J^{2b}_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & J^{2a}_{3,1} & J^{2a}_{3,2} & J^{2a}_{3,3} & J^{2b}_{3,1} & J^{2b}_{3,2} & J^{2b}_{3,3} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J^{3a}_{1,1} & J^{3a}_{1,2} & J^{3a}_{1,3} & J^{3b}_{1,1} & J^{3b}_{1,2} & J^{3b}_{1,3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J^{3a}_{2,1} & J^{3a}_{2,2} & J^{3a}_{2,3} & J^{3b}_{2,1} & J^{3b}_{2,2} & J^{3b}_{2,3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J^{3a}_{3,1} & J^{3a}_{3,2} & J^{3a}_{3,3} & J^{3b}_{3,1} & J^{3b}_{3,2} & J^{3b}_{3,3} \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} f_{x1} \\ f_{y1} \\ f_{z1} \\ n_{x1} \\ n_{y1} \\ n_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ n_{x2} \\ n_{y2} \\ n_{z2} \\ f_{x3} \\ f_{y3} \\ f_{z3} \\ n_{x3} \\ n_{y3} \\ n_{z3} \end{bmatrix}
\tag{3.34}
$$

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 3.33.

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

$$
(J^{ia}) = \begin{bmatrix}
-\mathrm{L}\mathrm{s}_{\mathrm{h}2_\mathrm{i}}(\mathrm{c}_{\mathrm{k_i+h1_i}} + \mathrm{c}_{\mathrm{h}1_\mathrm{i}}) & \mathrm{L}\mathrm{c}_{\mathrm{h}2_\mathrm{i}}(\mathrm{c}_{\mathrm{k_i+h1_i}} + \mathrm{c}_{\mathrm{h}1_\mathrm{i}}) & \mathrm{L}\mathrm{s}_{\mathrm{k_i+h1_i}} + \mathrm{L}\mathrm{s}_{\mathrm{h}1_\mathrm{i}} + \mathrm{P}_{\mathrm{y_i}}\,\mathrm{c}_{\mathrm{h}2_\mathrm{i}} - \mathrm{P}_{\mathrm{x_i}}\,\mathrm{s}_{\mathrm{h}2_\mathrm{i}} \\
-\mathrm{L}\mathrm{c}_{\mathrm{h}2_\mathrm{i}}(1 + \mathrm{c}_{\mathrm{k_i}}) - \mathrm{P}_{\mathrm{y_i}}\,\mathrm{s}_{\mathrm{k_i+h1_i}} & -\mathrm{L}\mathrm{s}_{\mathrm{h}2_\mathrm{i}}(1 + \mathrm{c}_{\mathrm{k_i}}) + \mathrm{P}_{\mathrm{x_i}}\,\mathrm{s}_{\mathrm{k_i+h1_i}} & -\mathrm{P}_{\mathrm{y_i}}\,\mathrm{s}_{\mathrm{h}2_\mathrm{i}}\mathrm{c}_{\mathrm{k_i+h1_i}} - \mathrm{P}_{\mathrm{x_i}}\,\mathrm{c}_{\mathrm{h}2_\mathrm{i}}\mathrm{c}_{\mathrm{k_i+h1_i}} \\
\mathrm{L}\mathrm{c}_{\mathrm{h}2_\mathrm{i}}\mathrm{s}_{\mathrm{k_i}} - \mathrm{P}_{\mathrm{y_i}}\,\mathrm{c}_{\mathrm{k_i+h1_i}} & \mathrm{L}\mathrm{s}_{\mathrm{h}2_\mathrm{i}}\mathrm{s}_{\mathrm{k_i}} + \mathrm{P}_{\mathrm{x_i}}\,\mathrm{c}_{\mathrm{k_i+h1_i}} & \mathrm{P}_{\mathrm{y_i}}\,\mathrm{s}_{\mathrm{h}2_\mathrm{i}}\mathrm{s}_{\mathrm{k_i+h1_i}} + \mathrm{P}_{\mathrm{x_i}}\,\mathrm{c}_{\mathrm{h}2_\mathrm{i}}\mathrm{s}_{\mathrm{k_i+h1_i}}
\end{bmatrix}
\tag{3.35}
$$

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

$$
(J^{ib}) = \begin{bmatrix}
-c_{h2_i} & -s_{h2_i} & 0 \\
s_{h2_i} c_{k_i+h1_i} & -c_{h2_i} c_{k_i+h1_i} & -s_{k_i+h1_i} \\
-s_{h2_i} s_{k_i+h1_i} & c_{h2_i} s_{k_i+h1_i} & -c_{k_i+h1_i}
\end{bmatrix}
\tag{3.36}
$$

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 3.25. 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. We verified that this subspace is the same for any foot angles. Therefore we were able to arbitrarily set the foot angles 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,

$$(J^{ib})^{-1} = \begin{bmatrix} -c_{h2_i} & s_{h2_i}c_{k_i+h1_i} & -s_{h2_i}s_{k_i+h1_i} \\ -s_{h2_i} & -c_{h2_i}c_{k_i+h1_i} & c_{h2_i}s_{k_i+h1_i} \\ 0 & -s_{k_i+h1_i} & -c_{k_i+h1_i} \end{bmatrix} \tag{3.37}$$

We eliminate the Constrained Force Set from Equation 3.34 using Equations 2.15 and 2.16,

$$\begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ -J^{-1b}J^{1a}_{1,1} & -J^{-1b}J^{1a}_{1,2} & -J^{-1b}J^{1a}_{1,3} & -J^{-2b}J^{2a}_{1,1} & -J^{-2b}J^{2a}_{1,2} & -J^{-2b}J^{2a}_{1,3} & -J^{-3b}J^{3a}_{1,1} & -J^{-3b}J^{3a}_{1,2} & -J^{-3b}J^{3a}_{1,3} \\ -J^{-1b}J^{1a}_{2,1} & -J^{-1b}J^{1a}_{2,2} & -J^{-1b}J^{1a}_{2,3} & -J^{-2b}J^{2a}_{2,1} & -J^{-2b}J^{2a}_{2,2} & -J^{-2b}J^{2a}_{2,3} & -J^{-3b}J^{3a}_{2,1} & -J^{-3b}J^{3a}_{2,2} & -J^{-3b}J^{3a}_{2,3} \\ -J^{-1b}J^{1a}_{3,1} & -J^{-1b}J^{1a}_{3,2} & -J^{-1b}J^{1a}_{3,3} & -J^{-2b}J^{2a}_{3,1} & -J^{-2b}J^{2a}_{3,2} & -J^{-2b}J^{2a}_{3,3} & -J^{-3b}J^{3a}_{3,1} & -J^{-3b}J^{3a}_{3,2} & -J^{-3b}J^{3a}_{3,3} \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{x1} \\ f_{y1} \\ f_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ f_{x3} \\ f_{y3} \\ f_{z3} \end{bmatrix} \tag{3.38}$$

where,

$$-(J^{ib})^{-1}J^{ia} = \begin{bmatrix} 0 & \mathrm{Lc_{k_i+h1_i} + Lc_{h1_i}} & \mathrm{P_{y_i} + Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i})} \\ \mathrm{-L(c_{k_i+h1_i} + c_{h1_i})} & 0 & \mathrm{Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{x_i}} \\ \mathrm{-Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{y_i}} & \mathrm{-Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) + P_{x_i}} & 0 \end{bmatrix} \tag{3.39}$$

Equation 3.38 is now rearranged so that it is in the form of Equation 2.18.

$$
\begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \\ 0 \\ 0 \\ 0 \end{bmatrix}
=
\begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
-J^{-1b}J^{1a}_{1,1} & -J^{-1b}J^{1a}_{1,3} & -J^{-2b}J^{2a}_{1,1} & -J^{-2b}J^{2a}_{1,2} & -J^{-2b}J^{2a}_{1,3} & -J^{-3b}J^{3a}_{1,3} & -J^{-3b}J^{3a}_{1,1} & -J^{-3b}J^{3a}_{1,2} & -J^{-1b}J^{1a}_{1,2} \\
-J^{-1b}J^{1a}_{2,1} & -J^{-1b}J^{1a}_{2,3} & -J^{-2b}J^{2a}_{2,1} & -J^{-2b}J^{2a}_{2,2} & -J^{-2b}J^{2a}_{2,3} & -J^{-3b}J^{3a}_{2,3} & -J^{-3b}J^{3a}_{2,1} & -J^{-3b}J^{3a}_{2,2} & -J^{-1b}J^{1a}_{2,2} \\
-J^{-1b}J^{1a}_{3,1} & -J^{-1b}J^{1a}_{3,3} & -J^{-2b}J^{2a}_{3,1} & -J^{-2b}J^{2a}_{3,2} & -J^{-2b}J^{2a}_{3,3} & -J^{-3b}J^{3a}_{3,3} & -J^{-3b}J^{3a}_{3,1} & -J^{-3b}J^{3a}_{3,2} & -J^{-1b}J^{1a}_{3,2} \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix} f_{x1} \\ f_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ f_{z3} \\ f_{x3} \\ f_{y3} \\ f_{y1} \end{bmatrix}
\tag{3.40}
$$

Now A^m is the upper left 6×6 submatrix, A^r is the upper right 6×3 submatrix, B is the lower left 3×6 submatrix, and the 3×3 identity matrix is in the lower right corner.

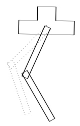
We can now eliminate the Redundant Force Set as in Equations 2.18 and 2.19, to get

$$
\begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix}
=
\begin{bmatrix}
1 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & s_{4,2} & 0 & s_{4,4} & s_{4,5} & s_{4,6} \\
s_{5,1} & s_{5,2} & s_{5,3} & 0 & s_{5,5} & s_{5,6} \\
s_{6,1} & 0 & s_{6,3} & s_{6,4} & 0 & 0
\end{bmatrix}
\begin{bmatrix} f_{x1} \\ f_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ f_{z3} \end{bmatrix}
\tag{3.41}
$$

where,

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

To solve for the Minimum Force Set, we need to invert the 6×6 matrix in Equation 3.41. This inversion is shown in Appendix A. We next use Equation 2.17 to solve for the Redundant Force Set in terms of the Minimum Force Set,



$$
\begin{bmatrix} f_{x3} \\ f_{y3} \\ f_{y1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{x1} \\ f_{z1} \\ f_{x2} \\ f_{y2} \\ f_{z2} \\ f_{z3} \end{bmatrix} \tag{3.42}
$$

Finally Equation 2.15 is used to solve for the Constrained Force Set in terms of the Minimum and Redundant Force Sets,

$$
\begin{bmatrix} n_{x_i} \\ n_{y_i} \\ n_{z_i} \end{bmatrix} = \begin{bmatrix} 0 & Lc_{k_i+h1_i} + Lc_{h1_i} & P_{y_i} + Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) \\ -L(c_{k_i+h1_i} + c_{h1_i}) & 0 & Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{x_i} \\ -Lc_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) - P_{y_i} & -Ls_{h2_i}(s_{k_i+h1_i} + s_{h1_i}) + P_{x_i} & 0 \end{bmatrix} \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \end{bmatrix} \tag{3.43}
$$

We now have a relatively small set of equations for coordinating three legs of a hexapod robot. To implement virtual model control, one needs only to define a generalized force function as described in Section 2.5. The above equations can then be used to compute the required joint torques for the three legs.

Chapter 4

Virtual Models vs. Local Control Techniques

In this Chapter, we describe a simple example in which virtual model control works but blind application of local control techniques, such as inverse kinematics, fail. The example is that of a simple 2-D foot which is modelled by three links and two joints. We wish to make the foot perform several maneuvers. These include moving to a desired configuration and transitioning from being supported by its heel to being only on its toes.

The mathematics required for implementing virtual components on this robot was described in Section 3.3. We attempt to provide an intuitive description of why virtual model control works with this example and inverse kinematics fails. Simulations were performed which verified the technique.

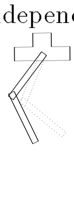
4.1 Intuitive Explanation

Figure 4-1 shows the foot model in a hypothetical position. The solid figure shows the current position of the foot. We wish to control the foot so that it moves towards the desired position, indicated by the dashed figure. To do this we can connect a virtual spring between frames {A} and {B} which will push the foot toward the desired position. Another option is to use inverse kinematics control to servo the individual joints to the corresponding desired angles.

A virtual spring will produce the force F_{vm} as in the figure. Transformation to joint torques will produce a toe torque T_t which would seem to cause the toe angle to increase. This is opposite to the direction which we desire it to travel! However, the joint will indeed travel against the applied torque (decrease in angle), thereby producing negative work. This has been demonstrated via simulation which we discuss below. In contrast, inverse kinematic control will produce a toe torque which opposes the joint angle error and thus will be opposite in sign of the torque generated via virtual model control. This torque will indeed cause the angle error to decrease but instead of the foot pushing on the ground it will simply lift its toes. Therefore, the blind application of inverse kinematics control will produce undesirable results with this example.

One can demonstrate the above result by standing on his or her toes and pushing up and down while keeping one's knees locked. To go up, one must push harder on his or her toes which results in the toe joint rotating against the applied torque. At first it may take some effort to determine exactly what is happening. Although the virtual force (push down) and cartesian movement (move up) is easily understandable, the actual joint torques and joint motion takes some effort to properly observe. This suggests that the joint angles and torques in this task don't even enter ones conscious. If so, then coordinate transformations similar to those used in virtual models must occur outside ones conscious in order to perform this task.

One fundamental difference between the two methods is that virtual model control utilizes information from all the joints and is thus a global method. In contrast, inverse kinematics is generally a local method. Once the desired joint angles are determined, each joint independently attempts to



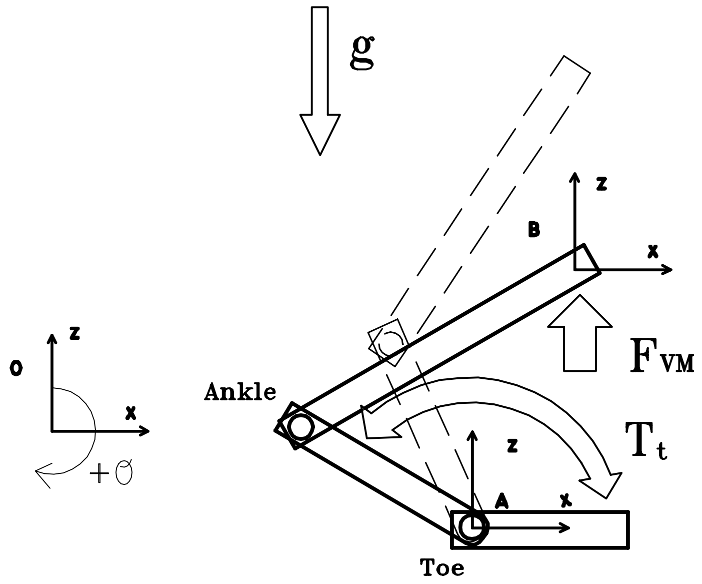


Figure 4-1: Simple foot example showing virtual model forces, F_{VM} and required toe torque, T_t.

servo to that position.

It is possible that a *non-local* variant of inverse kinematics could work with this example. For instance, if we examine the error of the virtual angle between the toes and the body and produce a toe torque which opposes this error, then the torque will at least have the desired sign. However, this would be a global technique, since it would require knowledge of both joint angles and would require the use of a virtual sensor (measurement of the angle between the toe joint and the body).

Another difference between the two methods is that intuition can be easily applied to virtual model control while it is not clear how to apply it to local techniques. With our foot example, intuition helps us decide where to place the virtual model frames and which components to use. Blindly applying a technique such as inverse kinematics doesn't even allow one to exploit the fact that the foot is not rigidly attached to the ground. If the foot were attached to the ground, inverse kinematics would work. However, it is not clear how to exploit this knowledge to alter the inverse kinematic technique for application to our foot example.

4.2 Foot Simulation

Simulations were performed to demonstrate that virtual model control can be used to successfully control the robot foot example. The following components were used: a virtual spring-damper acting in the horizontal (x) direction; a separate virtual spring-damper acting in the vertical (z) direction; and a virtual vertical force to help counter gravity. We stress that gravity cancellation was probably not necessary but was used so that the virtual vertical spring could have a lower spring constant and hence be a "less demanding" virtual component in terms of bandwidth requirements (see Chapter 6).

Figure 4-2 shows the results of a particular simulation in which the foot, initially in a "back on its heels" position, moved to a desired "up on its toes" position. The actual x and z values (solid) are plotted along with the virtual spring set-points (dotted) in the first two graphs. The resultant virtual horizontal and vertical forces are shown in the next two graphs. The applied joint torques are plotted in the bottom two graphs. On the right is a graphical representation of the foot going "up on its toes."

When the foot is "back on its heel", we place the virtual model reaction frame at the ankle

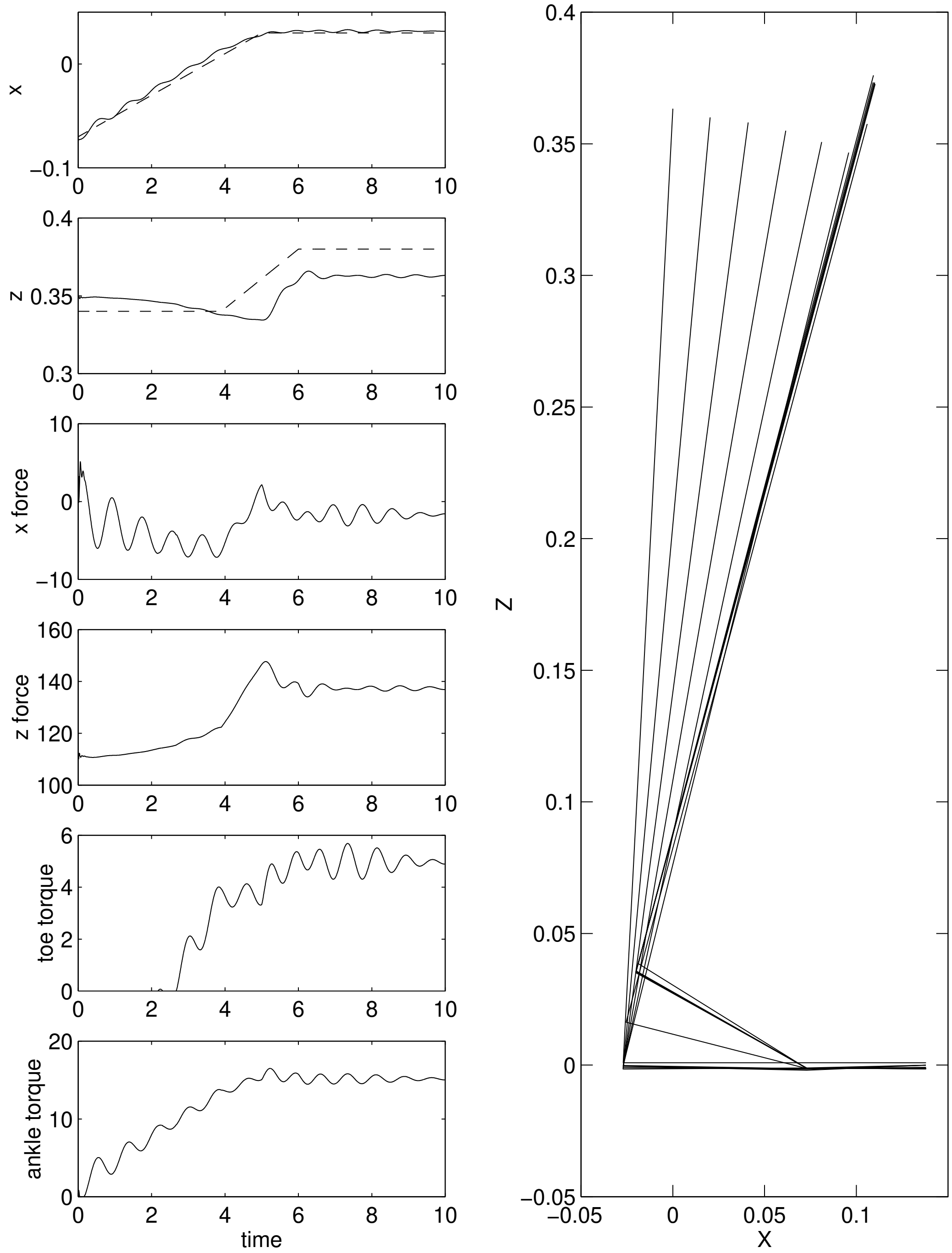


Figure 4-2: Simple foot example simulation. The top two graphs show the x and z positions of the top of the upper link with respect to the toe joint. The middle two graphs show the forces applied in the x and z directions due to the virtual components. The bottom two graphs show the resultant toe and ankle torques. On the right is a graphical representation of the foot simulation.

joint rather than at the toe joint. This is accomplished simply by applying no toe torque, which effectively projects the virtual forces unto the line perpendicular to the upper leg link. Once the center of pressure passes the toe joint, which is sensed via ground contact interaction, then toe torque is applied. We do this because toes are not needed to support the foot if the heel is doing so and can therefore remain unactuated.

This simulation exhibits rather oscillatory behavior. This is because we chose a low virtual damper constant. Although the tracking error is relatively low for the x value, we are not interested in tracking error here. We are more interested that the simulation works and that it behaves as though the virtual components were actually connected to the foot. This simulation demonstrates that a few simple virtual components can be used to perform a somewhat complex task that a non-global controller such as blind application of inverse kinematics can not perform.

Chapter 5

Developing Lyapunov Functions for the Stability Analysis of Virtual Model Controllers

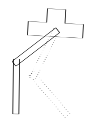
There are two powerful methods for deriving Lyapunov functions for a given virtual component. The first method draws from physical intuition and results in energy-based Lyapunov functions. The second method involves designing the component so that it forces a certain Lyapunov function to decrease. Both methods have their own niche and neither allows for the analysis of an arbitrary virtual component. Therefore, it is best to keep Lyapunov stability analysis in mind while designing virtual components rather than apply it after the components are designed. We briefly describe both methods below and in section 5.3 we use energy-based Lyapunov Functions to demonstrate that a passive robot controlled with any number of passive virtual spring-damper mechanisms remains passive.

5.1 Energy Based Lyapunov Functions

If physically realizable virtual components are implemented, a good candidate for a Lyapunov function is the virtual energy stored in the components. For example, if virtual mass-spring-damper mechanisms are used, the Lyapunov function can contain the kinetic energy of the mass plus the potential energy of the spring. If several such virtual components are used, one can simply sum these virtual energies with the kinetic energy of the robot itself ($\frac{1}{2}\dot{\Theta}^T H \dot{\Theta}$ in the case of a serial robot). The derivative of the Lyapunov function will then be the power dissipation of the virtual dampers. This is demonstrated in 5.3.

5.2 Design Based Lyapunov Functions

A second method for deriving Lyapunov functions is to develop a control law such that a given Lyapunov function decreases. Such an approach is used in sliding controllers and adaptive controllers. It is probably possible to use such techniques to develop "virtual adaptive controllers". This is not explored in this thesis however. Learning virtual components have been implemented previously using CMAC neural networks [22] but no Lyapunov analysis was performed.



5.3 Demonstration of Energy Based Lyapunov Functions for Virtual Spring-Damper Mechanisms

Energy methods can be used to develop Lyapunov function candidates for virtual mechanisms applied to a robot. Below we demonstrate, using Lyapunov analysis, that a passive robot, with passive spring-damper mechanisms connected to it, remains passive. In the analysis, we assume that the virtual components are perfectly implemented. Intuitively, if we had a passive robot with real versions of the passive components attached to it, then the system would remain passive.

The dynamic equation of a robot arm can be written

$$H(\Theta)\ddot{\Theta} + C(\Theta, \dot{\Theta})\dot{\Theta} + g(\Theta) = \tau \tag{5.1}$$

where H is the inertia matrix, C contains gyroscopic and coriolis terms, g contains gravity terms, and τ is the joint torques. Assume that the arm is operating in the horizontal plane so that $g(\Theta) = 0$.

Suppose we have a virtual component with action frame {B}, reaction frame {A}, and reference frame {O} and the following force function

$${}^{O}({}^{A}_{B}F) = -k_1({}^{O}({}^{A}_{B}\tilde{X})) - b_1({}^{O}({}^{A}_{B}\dot{X})) \tag{5.2}$$

were ${}^{O}({}^{A}_{B}\tilde{X})$ is the stretch of the virtual spring, ${}^{O}({}^{A}_{B}\dot{X})$ is the velocity of the virtual damper, k_1 is the virtual spring function, and b_1 is the virtual damper function. Neither the spring, nor the damper is assumed linear. We use ${}^{O}({}^{A}_{B}\dot{X})$ for the argument to the virtual damper equation in this case because the damper is connected between frames {A} and {B} and not connected to the spring set point. If it were, we would use ${}^{O}({}^{A}_{B}\dot{\tilde{X}})$. However, in this analysis, the spring set point isn't changing and hence ${}^{O}({}^{A}_{B}\dot{X}) ={}^{O} ({}^{A}_{B}\dot{\tilde{X}})$. Assume that the spring is passive, which implies that

$$\tilde{X}^T\ k_1(\tilde{X}) > 0 \quad \forall \tilde{X} \tag{5.3}$$

and that the damper is dissipative, which implies that

$$\dot{X}^T\ b_1(\dot{X}) > 0 \quad \forall \dot{X} \tag{5.4}$$

A good choice of Lyapunov function is the energy in the virtual spring

$$V = \int_0^{{}^{O}({}^{A}_{B}\tilde{X})} k_1(\tilde{X})\delta\tilde{X} \tag{5.5}$$

We now add this to the robot energy to get

$$V = \frac{1}{2}\dot{\Theta}^T H\dot{\Theta} + \int_0^{{}^{O}({}^{A}_{B}\tilde{X})} k_1(\tilde{X})\delta\tilde{X} \tag{5.6}$$

V will be positive definite and radially unbounded if the passive spring condition (Equation 5.3) holds. Taking the derivative we get,

$$\dot{V} = \dot{\Theta}^T H\ddot{\Theta} + \frac{1}{2}\dot{\Theta}^T \dot{H}\dot{\Theta} + k_1({}^{O}({}^{A}_{B}\tilde{X}))^{O}({}^{A}_{B}\dot{\tilde{X}})$$

Expanding $H\ddot{\Theta}$ using equation 5.1, using the fact that $\dot{H} - 2C$ is skew symmetric, and $\dot{\tilde{X}} = \dot{X}$, this becomes

$$\dot{V} = \dot{\Theta}^T({}^{O}({}^{A}_{B}J)^T\ {}^{O}({}^{A}_{B}F)) + k_1({}^{O}({}^{A}_{B}\tilde{X}))^{O}({}^{A}_{B}\dot{X})$$

Taking the transpose of Equation 2.3 we get

$$\dot{\Theta}^T\ {}^{O}({}^{A}_{B}J)^T ={}^{O} ({}^{A}_{B}\dot{X})^T$$

Using this equation, and substituting in the virtual component force function, we get

$$\dot{V} = {}^O({}^A_B\dot{X})^T(-k_1({}^O({}^A_B\tilde{X})) - b_1({}^O({}^A_B\dot{\tilde{X}}))) + k_1({}^O({}^A_B\tilde{X}))^O({}^A_B\dot{X})$$

which becomes

$$\dot{V} = -{}^O({}^A_B\dot{X})^T b_1({}^O({}^A_B\dot{X})) \tag{5.7}$$

$\dot{V}$ will be negative definite as long as the assumption of equation 5.4 holds.

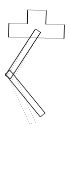
Notice that this demonstration says nothing about where the virtual components are attached nor makes any assumptions about linearity. Thus this analysis holds for all passive virtual spring-damper mechanisms connected to a robot arm with no gravity present. If gravity is present, one can add virtual vertical forces at the center of mass of each link to compensate for it. This is equivalent to adding $g(\Theta)$ to the applied torque vector. Our analysis then reduces to the present one and all results hold.

Because energy is a scalar, and hence adds, if we attach a number of virtual spring-dampers to the robot with action, reaction, and reference frames $\{B_i\}$, $\{A_i\}$ and $\{O_i\}$ and virtual spring and damper functions k_i and b_i, we can use the Lyapunov function and derivative

$$V = \frac{1}{2}\dot{\Theta}^T H \dot{\Theta} + \sum_i \int_0^{{}^{O_i}({}^{A_i}_{B_i}\tilde{X})} k_i(\tilde{X})\delta\tilde{X} \tag{5.8}$$

$$\dot{V} = -\sum_i {}^{O_i}({}^{A_i}_{B_i}\dot{X})^T b_i({}^{O_i}({}^{A_i}_{B_i}\dot{X})) \tag{5.9}$$

Thus we have shown that a passive robot arm remains passive when any number of passive virtual spring-damper mechanisms are attached to it.



Chapter 6

Virtual Model Limitations Due to Non-Ideal Actuators

The implementation and analysis of virtual components to this point has assumed that the desired joint torques required to implement the virtual component could be directly applied, i.e. that perfect torque sources existed at the joints. Actuator limitations can severely affect the stability and performance of a virtual model controller. In this chapter, we attempt to quantify the actuator limitations and determine where they come into play. I use series elastic actuator techniques [34] in this analysis since it is the actuator method used on our bipedal walking robot. The concepts presented below hold for any actuation technique however.

6.1 Virtual Actuator Saturation

If we ignore dynamic forces, we can solve for F_{max} in the following static equation to compute the steady-state virtual actuator saturation limits,

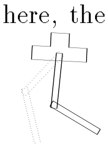
$$\tau_{max} = {}^{O}({}^{A}_{B}J)^{T}\,{}^{O}({}^{A}_{B}F_{max}) \tag{6.1}$$

To solve Equation 6.1, one must take the inverse of the Jacobian transpose. Near singularities large virtual forces can be exerted with small motions. This takes place, for example, with an almost straight knee. Away from singularities (large knee bend, for example), the maximum allowed virtual forces may be much smaller. Equation 6.1 gives us the static limitations of the virtual components. The next section discusses dynamic limitations.

6.2 Series Elastic Actuators

Robot force control has proven to be a difficult endeavor due mainly to actuator limitations. Since most actuator technologies deliver high speed and low torque, speed reduction has to take place in order to generate high torques at low speeds which is desirable in most robotic applications. Various methods of speed reduction exist, but the most common method involves gear trains. Geared transmissions suffer from friction, backlash, and non-collocation, all which are undesirable effects. Therefore, the most successful force controlled robots have used large pulleys for reduction and cable transmissions, thereby eliminating gears [27]. The drawbacks of this technique are the space requirements of the large pulleys and the need for low friction motors.

Series Elastic Actuation is a technique which allows one to perform force control with non-ideal actuators and transmissions. An elastic element (represented by k_s in Figure 6-1) is intentionally placed between the motor mass and the load. Since the force on the load is a function of the spring stretch, the force control problem effectively reduces to position control of the motor end of the spring [34, 21]. Although we examine Series Elastic Actuation limitations here, the following



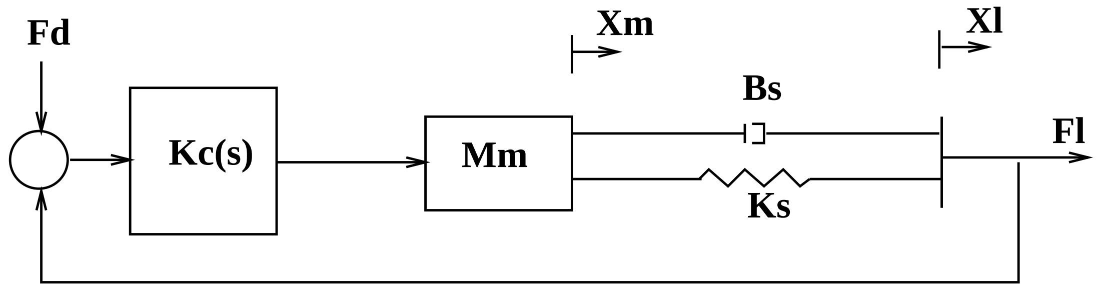


Figure 6-1: Series elastic actuator model. Mm is the motor mass. Ks and Bs are the spring constant and damping constant. Kc(s) is the compensator. Fd is the desired force while Fl is the actual force applied to the load. Xm and Xl are the motor and load displacements.

analysis will hold for any non-ideal torque control scheme. We use Series Elastic Actuation in the control of our bipedal robot (see Chapter 9).

6.3 System Block Diagram

Figure 6-2 shows two system block diagrams. Figure 6-2 a) is the desired system, achievable if we had perfect actuators while Figure 6-2 b) is the entire system, including non-ideal actuators. VM represents the virtual model, an impedance which takes as input the generalized positions of the robot, Θ_R and produces joint torques τ_d. R is the robot itself, an admittance which takes in joint torques, τ_R and outputs it's joint positions, Θ_R. G represents the actuator transfer function matrix from desired torque to actual torque and Z represents the actuator impedance matrix.

Using series elastic actuation with linear elements and implementing a PD compensator, $K_c(s)$, we get the following actuator transfer function matrices

$$G(s) = \frac{2\zeta_n\omega_n s + \omega_n^2}{s^2 + 2\zeta_n\omega_n s + \omega_n^2} I \tag{6.2}$$

$$Z(s) = \frac{-k_s s^2}{s^2 + 2\zeta_n\omega_n s + \omega_n^2} I \tag{6.3}$$

Bode diagrams of these transfer functions for the SISO case are shown in Figure 6-3 for $k_s = 10.0$, $\zeta_n = 0.7071$, $\omega_n = 30.0$. Ideally we want $G(s) = 1$ and $Z(s) = 0$. From the bode diagrams, we see that this is a good approximation at low frequencies, but above the actuator bandwidth frequency, $G(s) \rightarrow 0$ at -20 dB/dec and $Z(s)$ approaches the impedance of the spring, $-k_s$. This is because at high frequencies, the motor mass cannot be accelerated quickly enough and to the load, the system appears as the spring connected to ground.

6.4 Discussion

The bandwidth limitations of $G(s)$ are typical of any force control actuation scheme. The impedance transfer function $Z(s)$ is also typical except that with Series Elastic Actuation, the impedance at high frequencies is relatively low due to the elastic element. For this reason, Series Elastic Actuation successfully handles shock loads.

Due to the bandwidth limitations of the actuators, the implementable virtual components are limited. Intuitively, we can only use "low frequency" virtual components. In order to quantify this better, we must develop a method to analyze the effects of the non-ideal actuators. Several issues to consider are

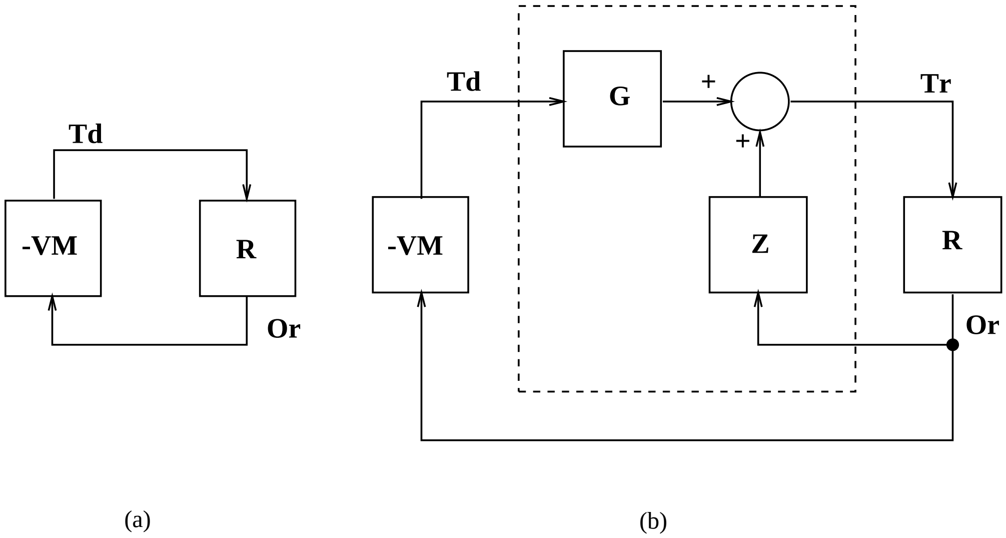


Figure 6-2: Block diagrams of a) Virtual Model (VM) and Robot (R) with perfect torque controllers and b) with non-ideal controllers. G is the torque controller transfer function matrix and Z is the torque controller impedance matrix.

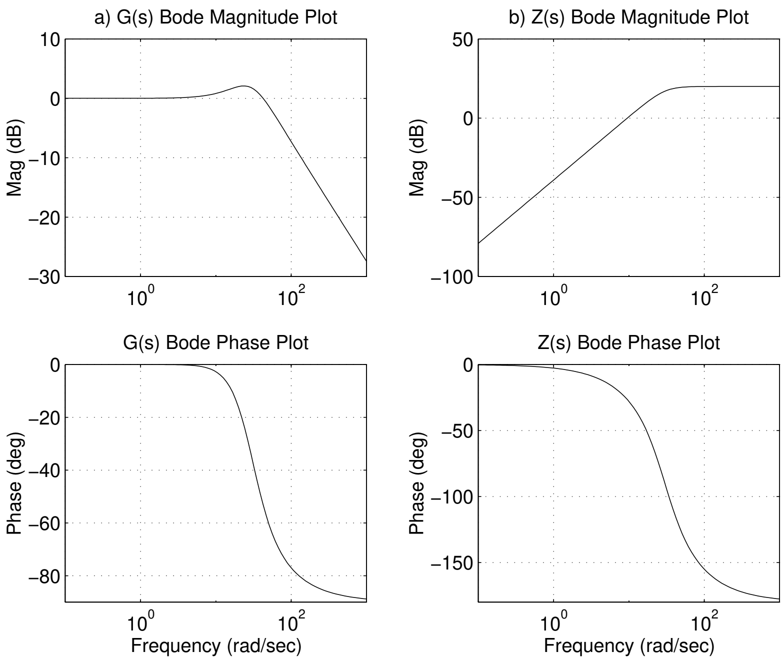


Figure 6-3: Typical bode diagrams of $G(s)$ and $Z(s)$ for Series Elastic Actuation. Ideally $G(s) = 1$ (0 db Gain, 0 deg Phase) and $Z(s) = 0$ (- ∞ Gain, 0 deg Phase). This is a good approximation at low frequencies but poor at high frequencies. At high frequencies $G(s) \approx 0$ and $Z(s) \approx K_s$.

- Stability of the virtual model and robot with ideal actuators (Figure 6-2, (a))
- Stability of the complete system including the non-ideal actuators (Figure 6-2, (b))
- Deviation of the complete system from the desired system
- Design of the virtual model controller such that the result of it and the non-ideal actuators better represents the desired virtual model.

Chapter 5 discussed Lyapunov methods to address the first issue. Tools to explore the second and third issues are discussed in Chapter 7.

We conjecture that the fourth issue is moot. We believe that it is not beneficial to attempt to modify the virtual model implementation, in light of the actuator limitations, so that the resultant effect is closer to that desired. At low frequencies, the actuators exhibit desirable behavior and hence there is no reason to attempt compensation. At high frequencies, the actuator performance is poor. However, the reasons for which the performance is poor are the same reasons which prevent successful compensation. These reasons include sampling rate, unmodelled dynamics, actuator saturation, etc. Therefore, given an actuator plant and force controller which are already tuned (G and Z of Figure 6-2 are fixed), we conjecture that no performance increases can be achieved by modifying the virtual model implementation. Therefore, one should take into account actuator and force controller limitations when choosing virtual components but assume perfect actuators when implementing them.

Chapter 7

Stability and Performance Analysis

In this chapter we examine stability and performance issues in implementing virtual model controllers. In doing so, we first define what we mean by these terms and then examine the available tools for analyzing stability and performance properties.

7.1 Stability

In Chapter 5 we discussed methods for constructing Lyapunov functions to show that a robot controlled with a particular virtual model controller is stable. This analysis, however, assumed perfect actuators. We require a method which will determine stability when non-ideal actuators are used assuming the system is stable when perfect actuators are used. Unfortunately, it is not clear how to construct Lyapunov functions for the case with non-ideal actuators. Hence, the non-linear case is not considered here. We hope that by considering the linear case, we can gain insight into the issues.

Figure 7-1 shows that the error due to non-ideal actuators can be expressed as a multiplicative error, E. If the actuator diagram of Figure 6-2 is used, we get

$$E(s) = [-G(s) + I] + Z(s)VM(s)^{-1} \tag{7.1}$$

The virtual model and robot can then be lumped into $T(s)$,

$$T(s) = -VM(s)R(s)[I + VM(s)R(s)]^{-1} \tag{7.2}$$

We can now invoke the Small Gain Theorem which states that the system in Figure 7-1 b) is stable if $T(s)$ is stable and

$$\sigma_{max}[T(jw)] < \frac{1}{\sigma_{max}[E(jw)]} \quad \forall\, \omega \tag{7.3}$$

This is a sufficient, but not necessary condition and hence is conservative. It is very useful, however, since one only needs an upper bound on $\sigma_{max}[E(jw)]$ and doesn't need to know $E(s)$ exactly.

7.2 Performance

As stated in Chapter 1, it is often very difficult to express a performance requirement for an interactive robot. In this light, we instead define the performance of the virtual model controller as how well it implements the desired virtual model while using non-ideal actuators. This is similar to the analysis in [6]. In other words, we wish to quantify how much VM' in Figure 7-2 will differ

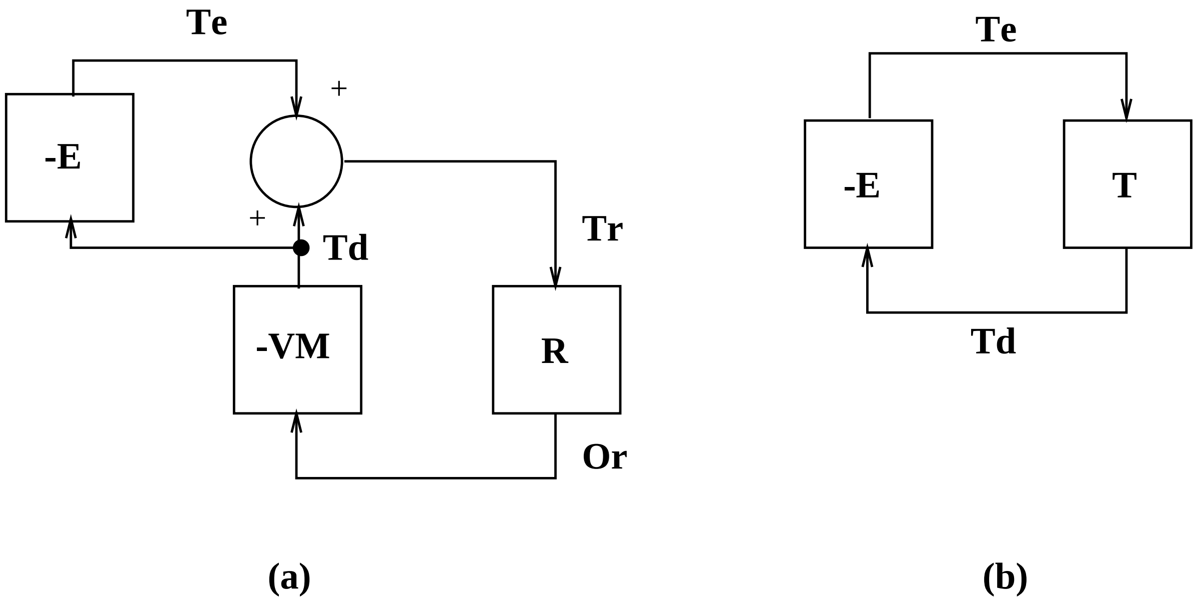


Figure 7-1: a) Virtual model error expressed as a multiplicative error (E). b) VM and R are combined into T for the linear case so that the Small Gain Theorem can be employed.

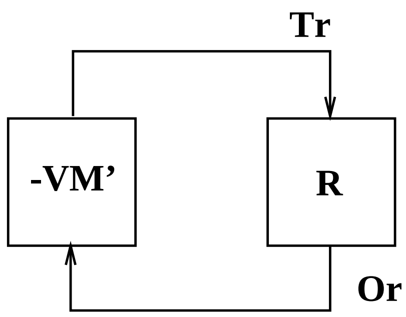


Figure 7-2: Actual virtual model implemented (VM') will differ from desired virtual model (VM).

from VM in Figure 6-2 a) or how much CL' in Figure 7-3 will differ from the desired closed loop response.

We can calculate the actual virtual model, VM', the desired closed loop response, CL, and the actual closed loop response as

$$VM'(s) = G(s)VM(s) - Z(s) \tag{7.4}$$

$$CL(s) = R(s)[I + R(s)VM(s)]^{-1} \tag{7.5}$$

$$CL'(s) = R(s)[I + R(s)VM'(s)]^{-1} \tag{7.6}$$

We can now define two performance specifications, namely that the maximum and minimum singular values of the actual virtual model are within the spread, δ_{VM}, of those of the desired virtual model,

$$Abs[\sigma_{max}[VM(jw)] - \sigma_{max}[VM'(jw)]] \leq \delta_{VM} \quad \forall\, \omega \leq \omega_b$$

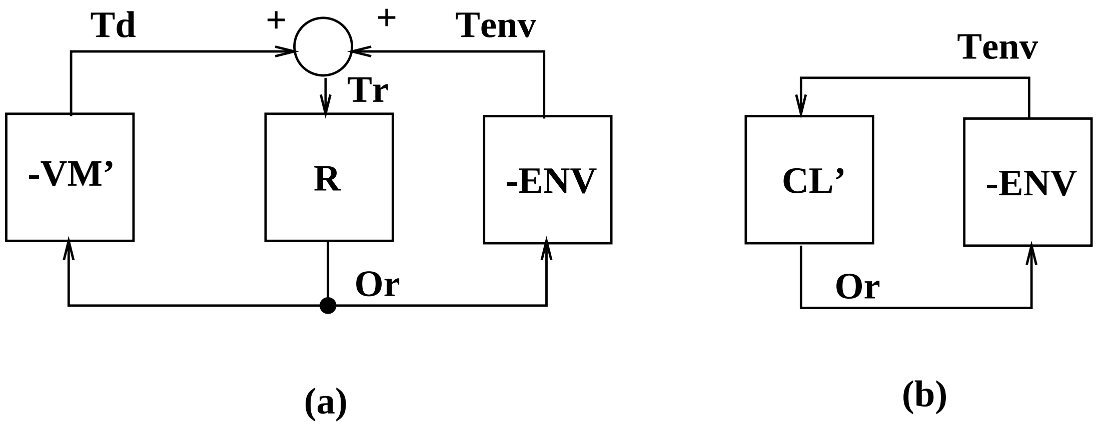


Figure 7-3: Robot (R) and virtual model controller (VM') as seen by the environment (ENV). In b) CL' is the closed loop admittance of the system as seen by the environment.

$$Abs[\sigma_{min}[VM(jw)] - \sigma_{min}[VM'(jw)]] \leq \delta_{VM} \quad \forall\, \omega \leq \omega_b$$

and that the maximum and minimum singular values of the actual closed loop system are within δ_{CL} of those of the desired closed loop system,

$$Abs[\sigma_{max}[CL(jw)] - \sigma_{max}[CL'(jw)]] \leq \delta_{CL} \quad \forall\, \omega \leq \omega_b$$

$$Abs[\sigma_{min}[CL(jw)] - \sigma_{min}[CL'(jw)]] \leq \delta_{CL} \quad \forall\, \omega \leq \omega_b$$

where ω_b is the break frequency below which we desire good performance.

7.3 Robustness

Finally, we may wish to analyze how the robot with virtual model controller will behave in the presence of environmental disturbances as depicted in figure 7-3. To this end, we can utilize passivity theory which states that a passive robot will be stable when in interaction with any passive environment [6]. The closed loop system will be passive if

$$jwCL'(jw) + jwCL'^T(-jw) > 0 \quad \forall\, \omega \geq 0 \tag{7.7}$$

We add the jw terms since CL' is defined as $\frac{\Theta_r}{\tau_{env}}$ while passivity requires it to be in the form $\frac{\dot{\Theta}_r}{\tau_{env}}$. For a single input - single output (SISO) system, equation 7.7 reduces to

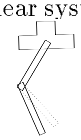
$$0^o < phase[CL'(jw)] < 180^o \quad \forall\, \omega \geq 0 \tag{7.8}$$

Note that if Equation 7.7 holds, the closed loop system will be stable when in contact with *any* passive environment. If we know that the environment will be compliant and damped to a minimum extent, then we can relax this requirement to CL' being passive below the "roll-off frequency", w_{env}, of the environment [4], i.e.

$$jwCL'(jw) + jwCL'^T(-jw) > 0 \quad \forall\, \omega \leq w_{env} \tag{7.9}$$

7.4 Discussion

This section has presented stability and performance requirements for linear robots controlled with linear virtual model controllers. It is not clear how to perform such analysis with non-linear systems.



We hope that by examining a few linear systems we may gain insight into the relevant issues. In Chapter 8 we present a simple SISO LTI example.

Chapter 8

Example of a Simple SISO, LTI Plant and Virtual Model Controller

In this chapter we examine a simple SISO LTI plant with a linear spring-damper virtual model controller. Although this is far from the non-linear MIMO case which we wish to analyze, it should still give insight into the relative control issues and may possibly provide some design guidelines.

In this example, the plant is simply a point mass robot with mass of M_r and the virtual model is a spring-damper mechanism with spring constant k_v and damping constant b_v. The desired virtual model, VM, and the robot, R, of Figure 6-2 are then,

$$VM(s) = b_v s + k_v \tag{8.1}$$

$$R(s) = \frac{1}{M_r s^2} \tag{8.2}$$

The desired closed loop system is thus

$$CL(s) = \frac{R(s)}{1 + R(s)VM(s)} = \frac{1}{M_r s^2 + b_v s + k_v} = \frac{1}{M_r(s^2 + 2\zeta_v w_v s + w_v^2)} \tag{8.3}$$

where ζ_v and w_v are used to represent the desired damping ratio and natural frequency.

Using Series Elastic Actuators, as in Chapter 6, we have the following force Transfer Function and impedance Transfer Function

$$G(s) = \frac{2\zeta_n \omega_n s + \omega_n^2}{s^2 + 2\zeta_n \omega_n s + \omega_n^2} \tag{8.4}$$

$$Z(s) = \frac{-k_s s^2}{s^2 + 2\zeta_n \omega_n s + \omega_n^2} \tag{8.5}$$

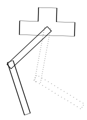
The apparent impedance to the robot is

$$VM'(s) = -Z(s) + G(s)VM(s) \tag{8.6}$$

$$= \frac{(-k_s + M_r(2\zeta_n w_n)(2\zeta_v w_v))s^2 + M_r(2\zeta_v w_v w_n^2 + 2\zeta_n w_n w_v^2)s + M_r w_v^2 w_n^2}{s^2 + 2\zeta_n w_n s + w_n^2}$$

resulting in a multiplicative error

$$E(s) = \frac{s^2(M_r(2\zeta_v w_v + w_v^2) - k_s)}{(s^2 + 2\zeta_n w_n s + w_n^2)M_r(2\zeta_v w_v + w_v^2)} \tag{8.7}$$



The transfer function used in invoking the small gain theorem, $T(s)$ in Figure 7-1, will then be

$$T(s) = \frac{2\zeta_v w_v + w_v^2}{M_r(s^2 + 2\zeta_v w_v s + w_v^2)} \tag{8.8}$$

and the actual closed loop system,

$$\mathrm{CL}'(s) = \frac{s^2 + 2\zeta_n w_n s + w_n^2}{M_r s^4 + 2\zeta_n w_n M_r s^3 + (-k_s + M_r((2\zeta_n w_n)(2\zeta_v w_v) + w_n^2))s^2 + M_r(2\zeta_v w_v w_n^2 + 2\zeta_n w_n w_v^2)s + M_r w_v^2 w_n^2} \tag{8.9}$$

The denominator of $CL'(s)$ is the characteristic polynomial of the entire system and its roots are the closed loop poles. In Figures 8-1 to 8-3 we show bode diagrams for these transfer functions, closed loop impulse response, and pole-zero diagrams of the desired and actual closed loop system for three different values of w_v. In Figure 8-4 we show the locus of closed loop poles of the system as we vary w_v from 1.0 to 60.0, with

$$k_s = 10.0$$

$$w_n = 30.0$$

$$\zeta_n = 0.7071$$

$$M_r = 10.0$$

$$\zeta_v = 0.7071$$

8.1 Results

In Figure 8-1, we use a value of w_v which is $\frac{1}{6}$ that of w_n. In other words, the bandwidth of the force controller is much higher than the bandwidth of the robot with virtual controller. Because of this, $\sigma_{max}[T(s)]$ is well below $\frac{1}{\sigma_{max}[E(s)]}$ thereby guaranteeing stability by the small gain theorem (Equation 7.3). We see that the bode magnitude plot of $VM'(s)$ resembles that of $VM(s)$ up to about 30.0 rad/sec and $CL'(s)$ and $CL(s)$ are similar over all frequencies. The actual and desired impulse response are nearly identical and the pole-zero diagram shows that the desired poles are placed well, while the extra poles due to the actuator dynamics are faster and nearly cancelled by near-by zeros. We see that phase$[CL'(jw)]$ stays between 0^o and 180^o, thereby guaranteeing stability when in contact with any passive environment by Equation 7.8.

In Figure 8-2, we use a value of w_v which is $\frac{1}{3}$ that of w_n. Stability margins begin to decrease and performance begins to degrade but may still be acceptable. Phase$[CL'(jw)]$ still stays below 180^o.

In Figure 8-3, we use a value of w_v which is $\frac{5}{6}$ that of w_n. The system is still stable but performance is unacceptable. The actuator dynamics are more pronounced than the virtual model dynamics. Also, Phase$[CL'(jw)]$ begins to go beyond 180^o. Therefore the system will not be stable for contact with all passive environments.

From Figure 8-4 we see that the actual closed loop poles match the desired ones for low w_v but start to diverge at higher w_v. The poles due to the actuator dynamics are fast and less significant for low w_v but move toward the jw axis as w_v is increased, dominating the system dynamics for $w_v > 25\frac{rad}{sec}$, and finally cross into the right hand plane, making the system unstable.

8.2 Discussion

From this simple example we see that even though the nominal system may be stable, as soon as actuator dynamics are considered, stability and performance may be compromised if the virtual model and robot have components at high frequencies compared to the actuator bandwidth. This

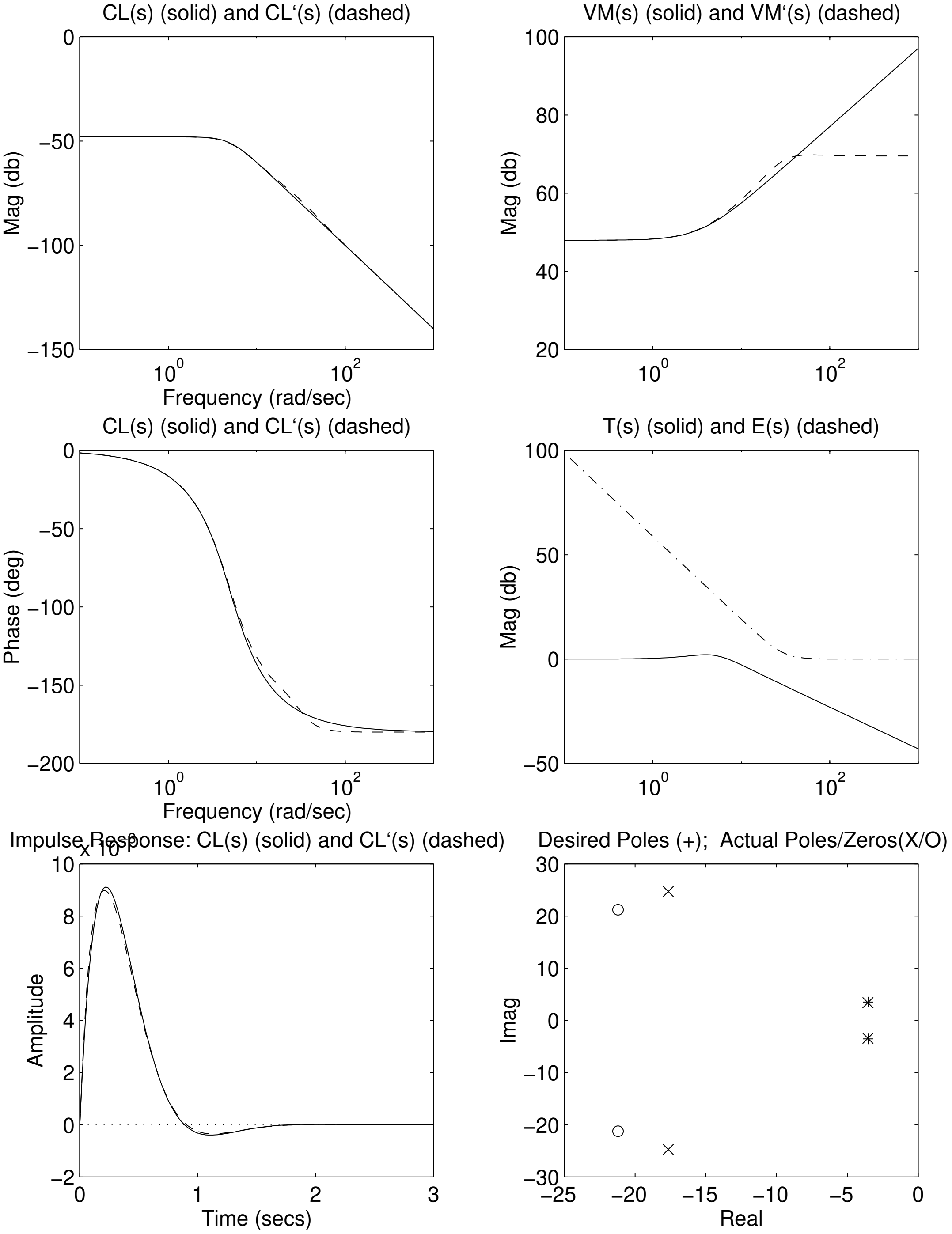
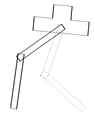


Figure 8-1: Simple SISO example with $\omega_v = 5.0$



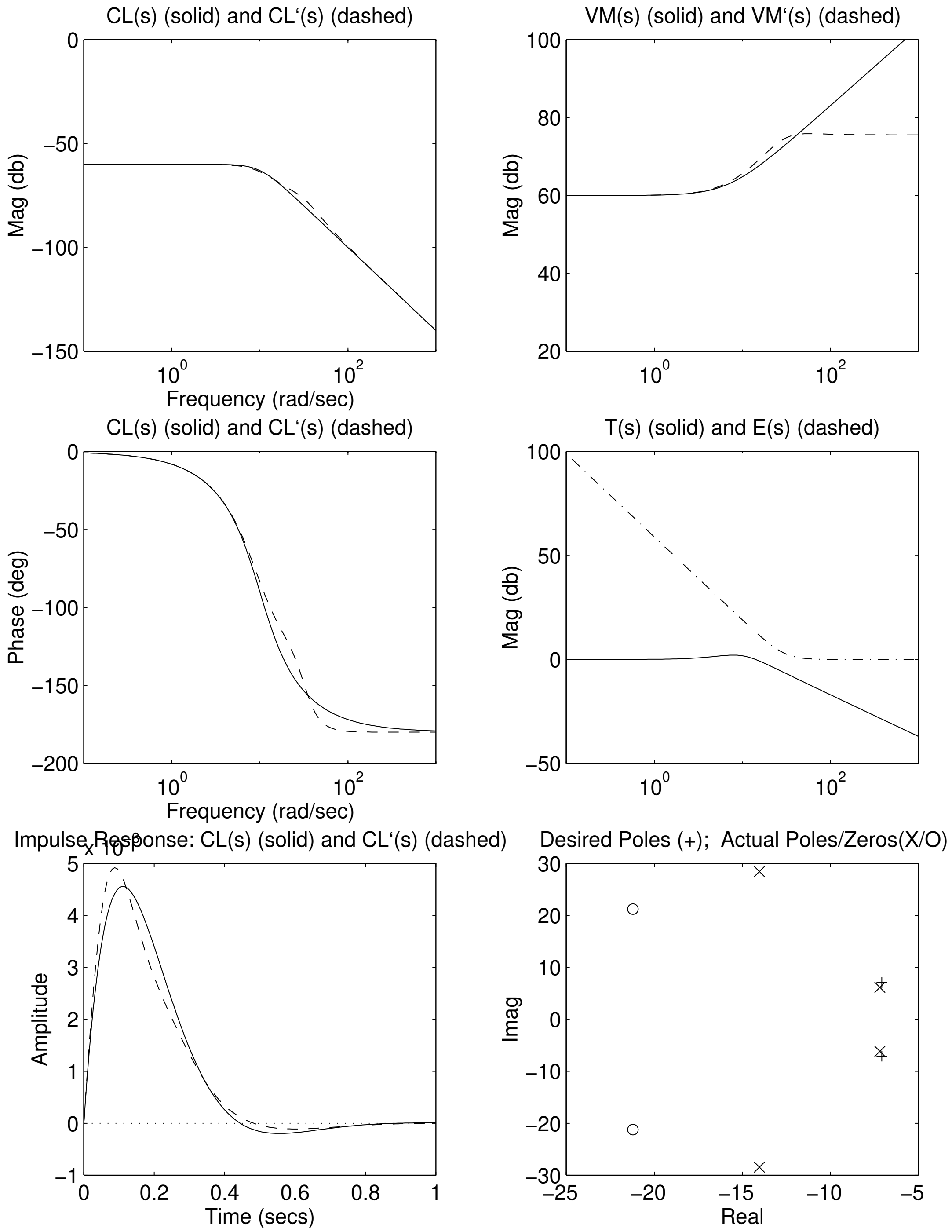


Figure 8-2: Simple SISO example with $\omega_v = 10.0$

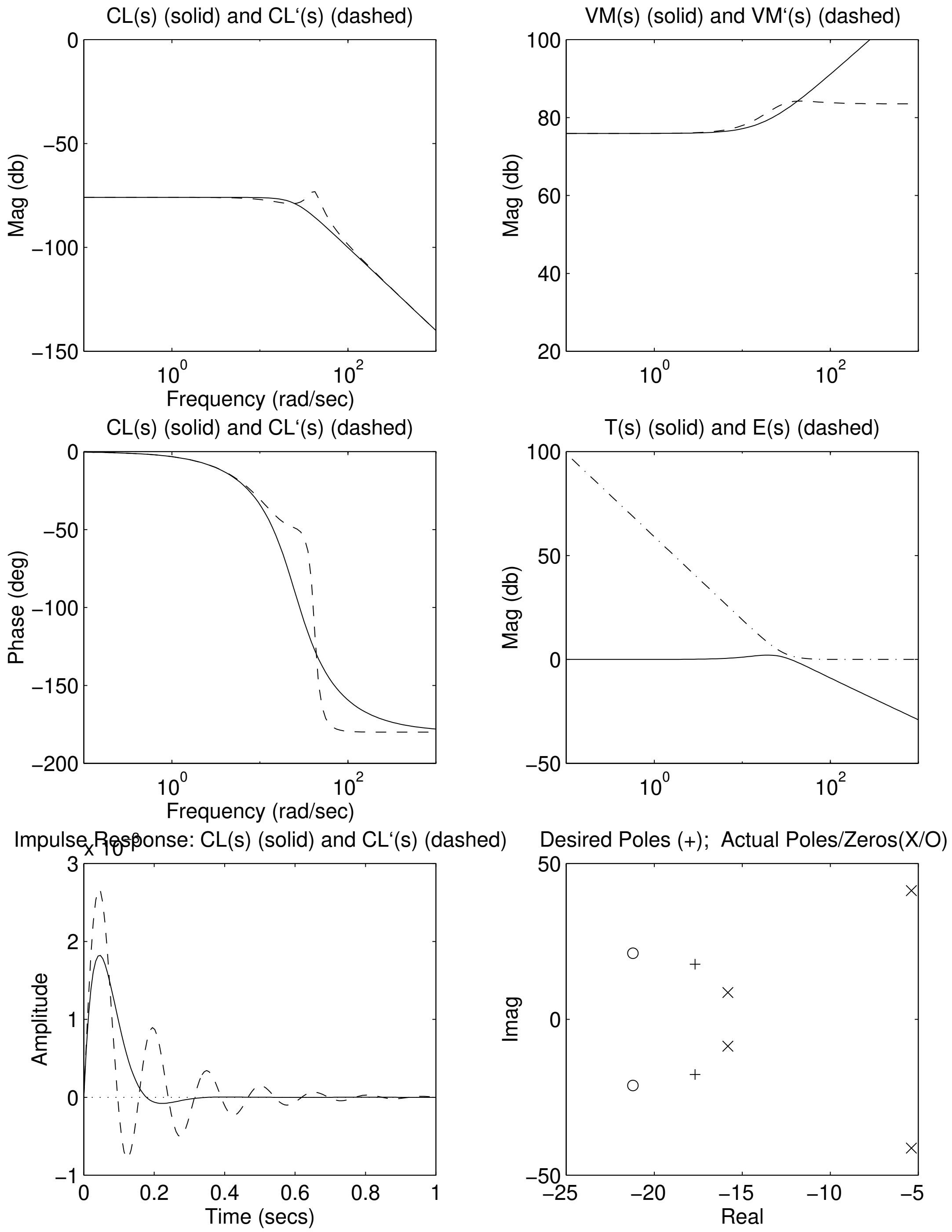


Figure 8-3: Simple SISO example with $\omega_v = 25.0$

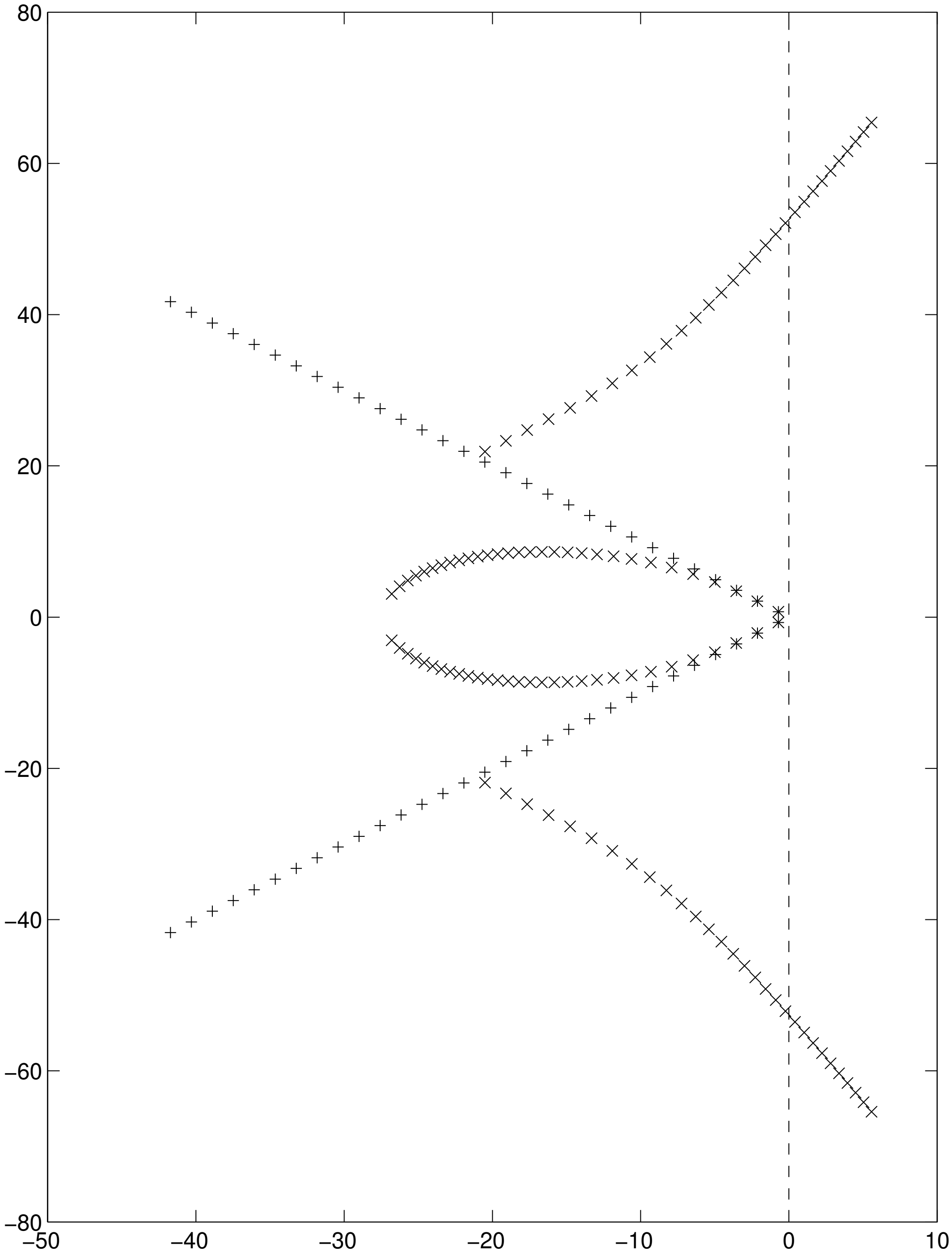


Figure 8-4: Locus of closed loop poles as ω_v is increased from 1 to 60. At low ω_v, the poles due to desired dynamics dominate. At high ω_v, the poles due to the actuator dynamics dominate and eventually enter the right half plane.

analysis holds for any non-ideal actuator scheme. We used Series Elastic Actuators here because it is the method we use for our bipedal walking robot.

For more complex systems, it may be difficult to perform such an analysis. However, we can conclude from this example that a higher bandwidth force controller will allow more accurate emulation of virtual components.

Chapter 9

Bipedal Walking Robot

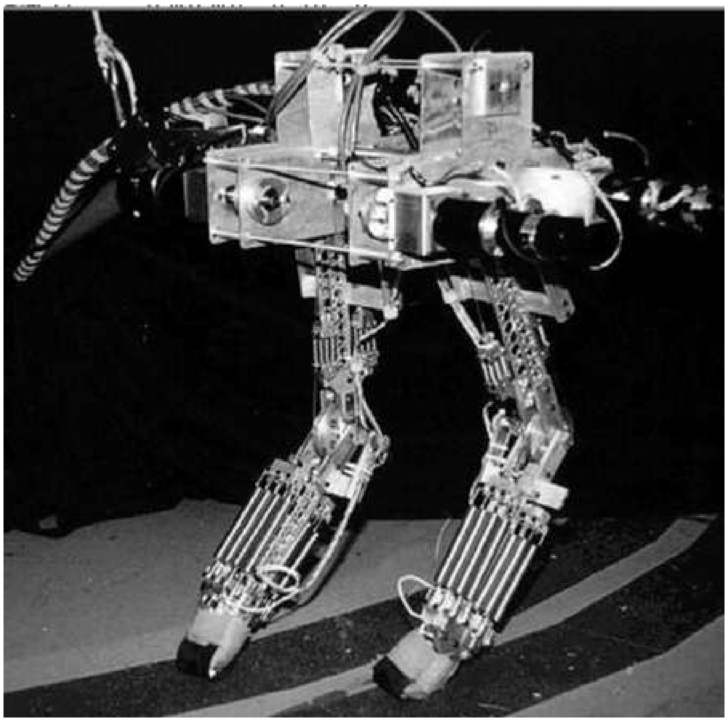


Figure 9-1: Photograph of Spring Turkey, our bipedal walking robot. There are four actuators attached to the body. Power is transmitted to the hips and knees via cables. The unactuated feet consist of a strip of rubber. A boom is used to prevent motion in the lateral, roll, and yaw directions. Note the spring packs which are used to implement series elastic actuation.

Two applications of virtual model control have been applied to Spring Turkey, our bipedal walking robot. The first application is deep knee bends which verifies the use of multi-frame virtual components for parallel mechanisms. The second application is simple walking. In both applications only a simple set of virtual components is used.

9.1 Spring Turkey

Figure 9-1 is a photograph of Spring Turkey, our bipedal walking robot. Spring Turkey was designed and built by Peter Dilworth in 1994. It has an actuated hip and knee on each leg. An unactuated boom constrains Spring Turkey's roll, yaw, and lateral motion thereby reducing it to a two dimensional robot. All of Spring Turkey's motors are located in its upper body, with power being transmitted to the joints via cable drives. Series Elastic Actuation [21] is employed at each degree of freedom, allowing for accurate application of torques and a high degree of shock tolerance. The maximum torque that can be applied to the hips is approximately 12 Nm while approximately 18 Nm can be applied to the knees. The force control bandwidth we achieve is approximately 20 Hz. Spring Turkey weighs in at approximately 22 lbs (10 kg) and stands 2 ft (60 cm) tall.

Rotary potentiometers at the hips, knees, and boom measure joint angles and body pitch. Linear springs are used at the hips and knees to implement Series Elastic Actuation. Rotary potentiometers measure hip spring stretch while linear potentiometers measure knee spring stretch.

9.2 Deep Knee Bends

Virtual model control was applied to Spring Turkey to make it perform deep knee bends using a very simple set of virtual components. A virtual spring-damper mechanism was simulated in the vertical direction. The rest position of the spring was modulated in a sinusoidal fashion to cause the body to move up and down. A virtual force source was applied in parallel with the vertical spring to help counter gravity. A separate virtual spring-damper mechanism was applied in the horizontal direction to regulate the body position to be midway between the two feet. A torsional virtual spring-damper mechanism was applied to regulate the body pitch to zero.

Figure 9-2 shows experimental data from Spring Turkey while performing deep knee bends. The top three graphs show the body's horizontal position (x), vertical position (z), and pitch (theta), and the corresponding spring set points (dotted). The next three graphs show the virtual forces applied to the body due to the virtual components. The resultant torques which are applied to each of the joints are plotted in the bottom graphs along with the position of the joints.

We have demonstrated that deep knee bends can be performed using only a simple set of virtual components. Although this might not be the "best" algorithm for implementing knee bends, it is easy to implement and hence a good start. The main issue is that the robot behaves as if the virtual components are really connected to the robot. We are less interested in "performance" in the traditional sense.

9.3 Stupid Walking

Virtual model control was applied to Spring Turkey to make it perform simple walking. We attempted to develop the simplest possible algorithm which would successfully allow Spring Turkey to take several steps before stopping or falling down. Simply stated the algorithm is as follows:

- Maintain a constant height and pitch.
- Transition from double support to single support if the body's x position becomes close to that of a foot.
- Transition from single support to double support if the body's x position becomes far away from the support foot.
- Servo the swing leg so that the foot is placed the nominal stride length away from the support foot when transitioning to double support.
- During double support, push in the direction of desired travel with a constant virtual force.

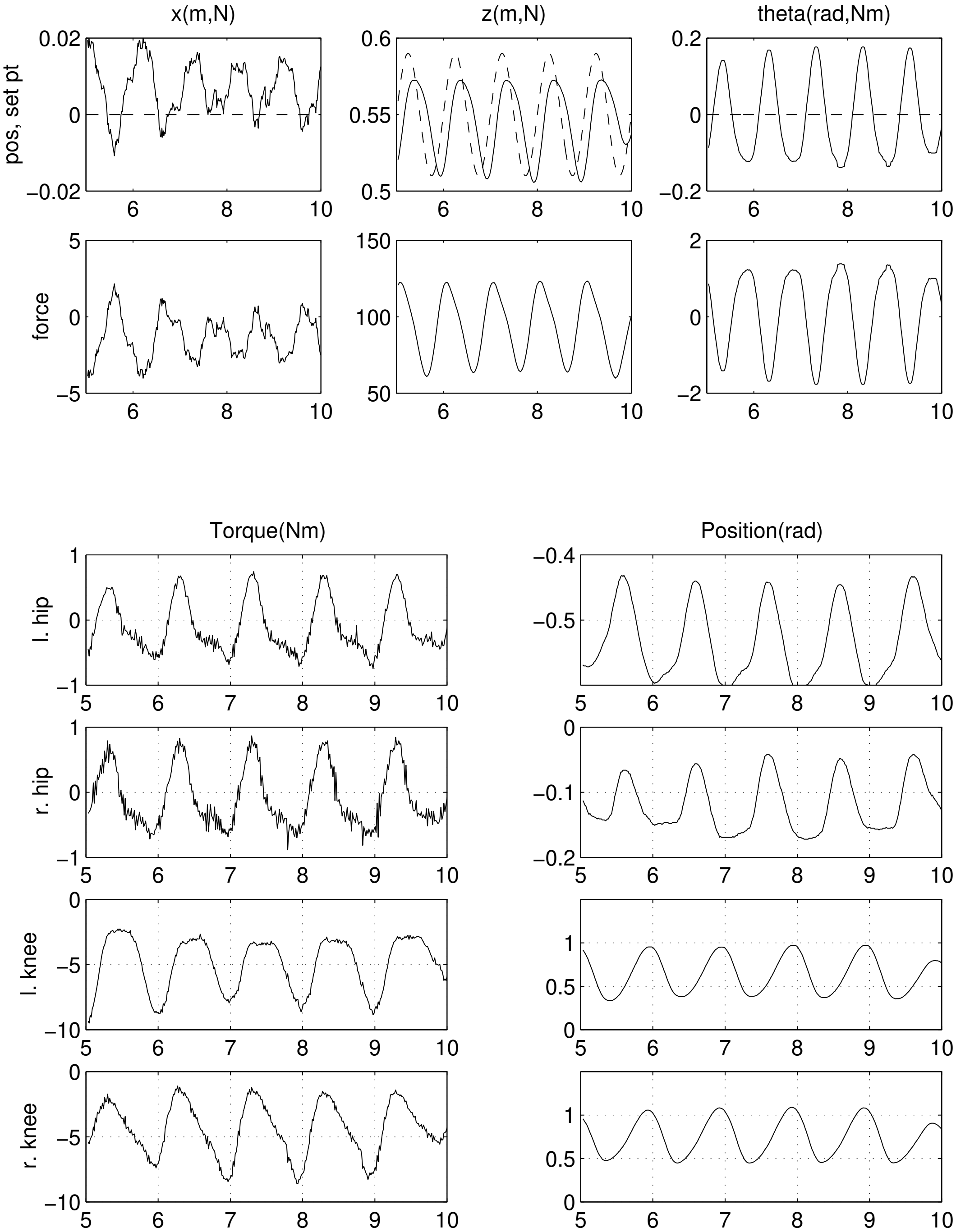


Figure 9-2: Knee bends experimental data. The top row of graphs show Spring Turkey's position and the virtual spring set points. The second row shows the virtual forces applied due to the virtual components. The bottom left column of graphs show the resultant torques applied to the joints while the bottom right column shows the position of the joints.

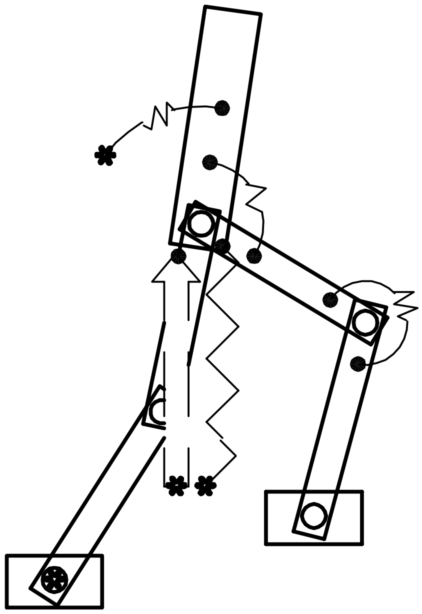


Figure 9-3: Spring Turkey with virtual components during single support phase. We use a virtual vertical spring-damper, a virtual vertical force source, and a torsional spring-damper connected between the body and support foot to maintain balance and height. The * indicates that these components are connected to the support foot. We use virtual spring-damper mechanisms on the swing leg for foot placement.

Because of the simplicity of this algorithm and the lack of any speed control or robustness mechanism, we dub it "Stupid Walking". We are not interested in the specific aspects of the algorithm here. Rather, we are interested in examining the effort required to implement it using virtual components and a simple state machine. We only hoped to achieve a few consecutive steps with this algorithm. Future work will concentrate on developing more robust algorithms.

To implement Stupid Walking, we use a simple set of virtual components. Figure 9-4 shows the state machine used for the Stupid Walking algorithm. Table 9.1 lists the trigger and branch events and the virtual components which are utilized in each state. During both double support and single support, a virtual spring-damper mechanism in the vertical (z) direction with a constant set point maintains a constant height. A virtual vertical force is applied to help counter gravity. Also, during both double and single support a virtual torsional spring-damper mechanism regulates the pitch angle to zero. During double support, we apply a constant virtual force in the forward horizontal (x) direction to help push Spring Turkey in the desired direction of travel. The swing leg is controlled via a local virtual spring-damper mechanism at each individual joint, thereby implementing an inverse kinematic algorithm. The set point of these springs are modulated so that the foot of the swing leg stays a given height off the ground during the swing phase and sets down at the nominal stride length distance when transitioning to double support. States LEFT_SUPPORT2 and RIGHT_SUPPORT2 are used as buffer states between single and double support. Because Spring Turkey has no foot switches, in these states the swing leg is simply made limp (zero torque applied to the joints) for a set delay time, allowing for the swing leg to fall to the ground before the large forces which double support require are applied.

The various virtual spring, damper, and force variables and walking parameters were chosen using physical insight and a manual search. The virtual vertical force matched the weight of the robot; the various virtual spring-damper constants were experimentally varied while physically examining their effects (resistance to being pushed on, decay rate, etc.) until the desired effects were achieved; the walking parameters and virtual horizontal force were changed through trial and error until

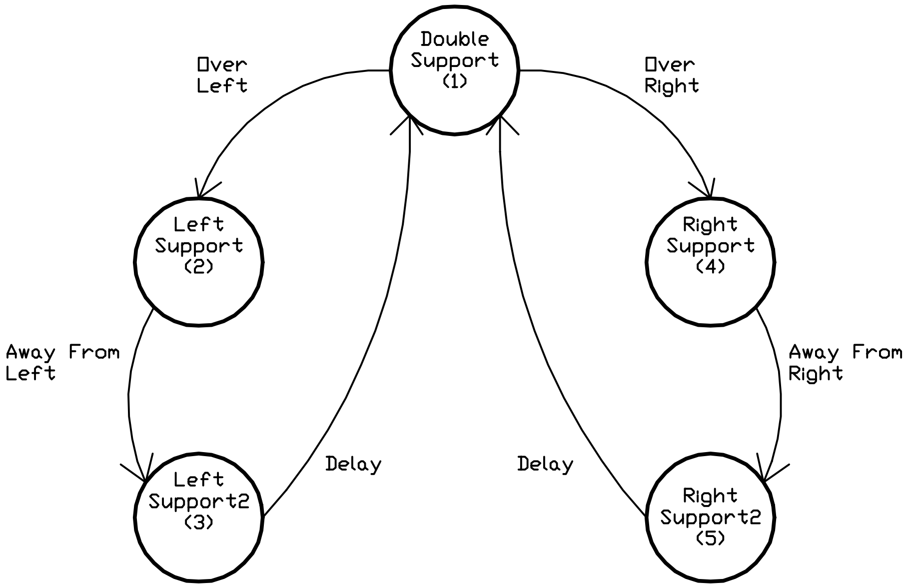


Figure 9-4: State machine used in the Stupid Walking algorithm.

Table 9.1: Details of Stupid Walking state machine.

State	Trigger Event	Virtual Components
1 Double Support	Delay after left or right support2	Vertical Spring-Damper Vertical Force Torsional Spring-Damper Horizontal Force
2 Left Support	Body nearly over left foot.	Vertical Spring-Damper Vertical Force Torsional Spring-Damper Local Spring-Dampers on Right Leg
3 Left Support2	Body away from left foot.	Vertical Spring-Damper Vertical Force Torsional Spring-Damper
4 Right Support	Body nearly over right foot.	Vertical Spring-Damper Vertical Force Torsional Spring-Damper Local Spring-Dampers on Left Leg
5 Right Support2	Body away from right foot.	Vertical Spring-Damper Vertical Force Torsional Spring-Damper

the robot successfully walked. These walking parameters consisted of nominal stride length and percent of stride length spent in single support. Because of the lack of speed control or robustness mechanisms in the Stupid Walking algorithm, an experimental search through the space of walking parameters had to be performed whenever the slightest mechanical changes were made to the robot or environment.

Walking was initiated in the single support phase. A slight push was applied to the robot to propel it forward. After the push, no external intervention occured.

Figure 9-6 shows experimental data from Spring Turkey while performing Stupid Walking. The top three graphs show the body's horizontal position (x), vertical position (z), and pitch (theta), and the corresponding spring set points (dotted). The next three graphs show the virtual forces applied to the body due to the virtual components. The resultant torques which are applied to each of the joints are plotted in the bottom graphs along with the position of the joints. The set points (dotted) of the springs used for the swing leg are plotted with the joint positions (solid). The state of the state machine is plotted in the middle graph.

The data in Figure 9-6 is plotted in graphical form in Figure 9-5 and also on the bottom of this document. The snapshots in Figure 9-5 are approximately 0.5 seconds apart. Lines are drawn to show the path of the tips of the feet and the center of the body. The graphics on the bottom of this document are spaced approximately 0.18 seconds apart. By flipping through the document, one can see an animation of the walking data of Figure 9-6.

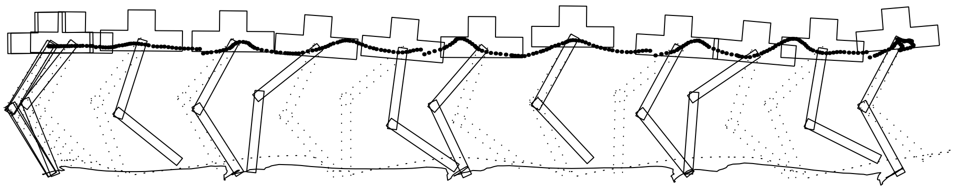


Figure 9-5: Elapsed time snapshot of the bipedal walking data in Figure 9-6. The drawings of the robot are spaced approximately 0.5 seconds apart. The left leg is dotted while the right leg is solid. Lines show the path of the tips of the foot and the center of the body.

Spring Turkey walked approximately 3 meters in 5 seconds for an average speed of approximately 0.6 m/s (1.35 mph). It took 8 steps (left to right or right to left support transitions), giving a step time of 0.6 seconds. It deviated a maximum of 4 cm from the nominal height of 52 cm and pitch was confined to ± 0.15 radians (± 8.6 deg). The hip actuators and torque controllers performed well while the knee actuators occasionally became saturated. Swing leg tracking was rather poor since the virtual spring constants were kept low.

9.4 Discussion

Spring Turkey performed deep knee bends and walked eight steps using a simple set of virtual components. We stress here that we *augmented* the natural dynamics of the robot with passive virtual components, rather than attempted to *cancel* the natural dynamics. In no case did we assume linear dynamics.

In both knee bends and walking, the state trajectory converges to a stable limit cycle for an appropriate choice of initial conditions. However, the knee bends experiment was more robust in response to variations in initial conditions and disturbances than walking. The stupid walking

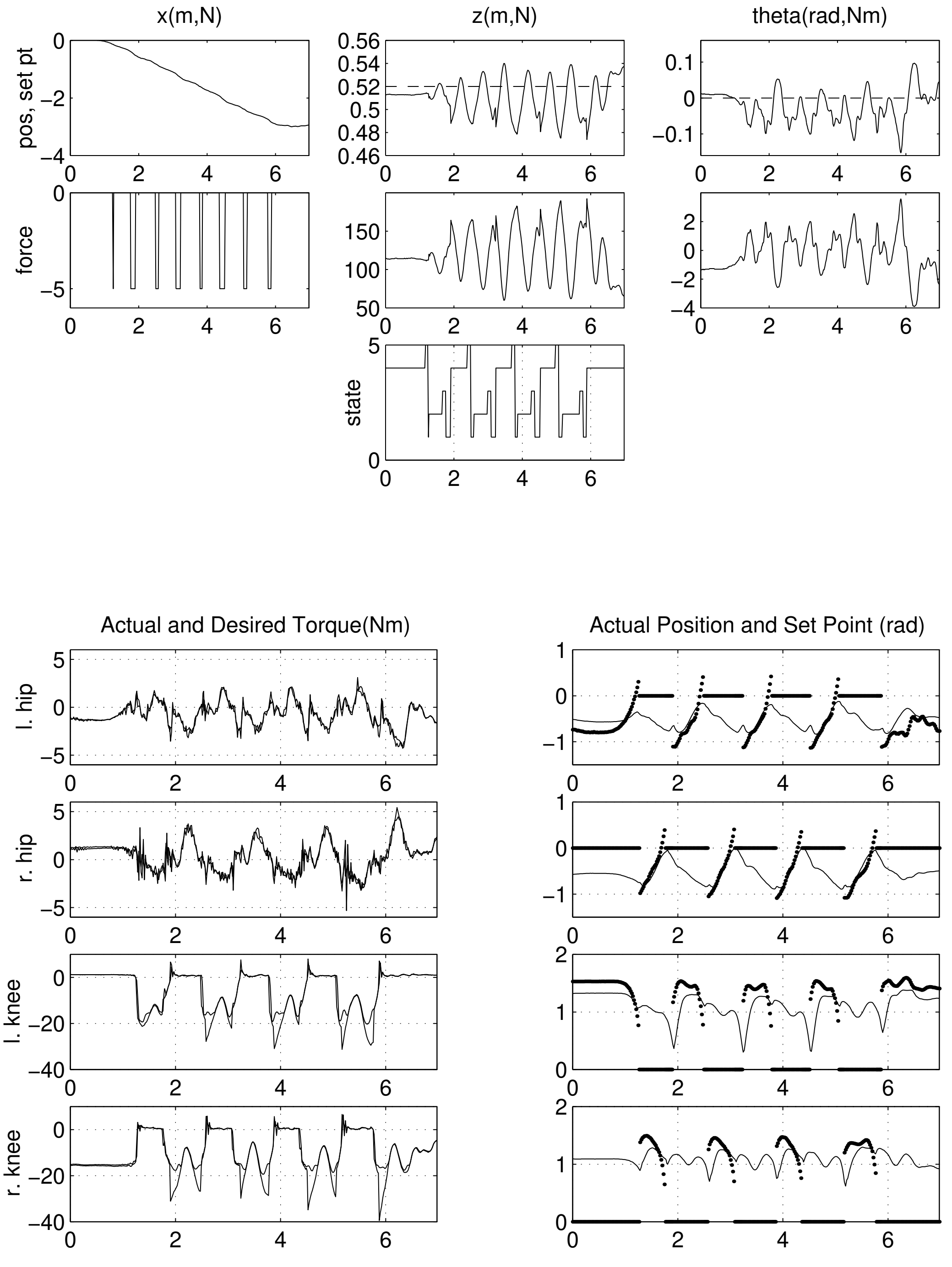
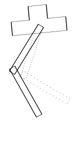


Figure 9-6: Spring Turkey walking data generated using virtual model control. The first row of graphs display the x, z, and θ positions and virtual spring set points, indicated with the dashed lines. The second row of graphs display the resultant forces applied to the body due to the virtual components. The graph in the third row shows the steady, periodic nature of the state machine. The bottom left column of graphs show the resultant desired and actual torques applied to the joints. The bottom right column of graphs display the position of the joints and the spring set point positions when the given leg is in the swing phase.



algorithm contains no speed control or robustness mechanisms and therefore is extremely dependent on initial conditions, ground conditions, etc. We have not attempted to perform an analysis on why the stupid walking algorithm converges to a limit cycle for the appropriate initial conditions. We only speculate that mechanisms similar to those present in McGeer's passive dynamic walker [17] are in force.

In order to perform more robust walking, a speed control mechanism is necessary. This could be accomplished either through foot placement during single support or through modulating the virtual feed-forward horizontal (x) force during double support. Future work will focus on using virtual model controllers to implement walking algorithms which others have found successful and on developing new algorithms.

Chapter 10

Conclusions

We can draw the following conclusions from this project,

1. Implementation of virtual model controllers is relatively straightforward and computationally efficient.
2. Virtual model control may be successful in some instances in which blind application of a local control technique, such as inverse kinematics, fails.
3. A passive robot, controlled with any combination of passive virtual components, will remain passive, assuming perfect actuators.
4. A number of tools exist for analyzing the stability and performance of linear virtual model controllers, connected to linear plants, utilizing non-ideal actuators.
5. It is not clear how to analyze stability and performance of a non-linear robot, connected with non-linear virtual components, while utilizing non-ideal actuators.
6. The higher the bandwidth of the actuator controllers, the more likely a stable virtual model controller will remain stable and the more likely it will implement the desired virtual model.
7. Simple bipedal walking can be achieved by utilizing a simple set of virtual components.

The ease of implementing virtual model controllers is promising. One of the major incentives of developing virtual model controllers is to make designing robot control algorithms easier and more intuitive. The algorithm designer will be given additional incentive to use virtual model controllers since they require minimal computational resources and are straightforward to derive. One of our goals is to automate this process.

The lack of non-linear analysis tools for the non-ideal actuator case is disappointing but was expected. One can only use the general guidelines that analysis of the linear case provides, including the rule of thumb that the higher the bandwidth of the actuator controller, the better and if the bandwidth is limited, one must use "less demanding" virtual components.

It is quite promising that simple bipedal walking was easy to achieve using simple virtual components. We are hopeful that virtual model control will be useful in producing more robust walking. Future work will focus on developing such algorithms.

Appendix A

Matrix Inversion for 3D Example

To solve for the Minimum Force Set for the 3D hexapod example, we must invert the matrix in Equation 3.41,

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

We then can solve the Minimum Force Set in terms of the generalized virtual forces,

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

where,

$$\begin{aligned}
det &= (4s_{63} - 8s_{61})(s_{42}(s_{55} - s_{56}) + s_{45}(-s_{52} + s_{56}) + s_{46}(s_{52} - s_{55})) \\
t_{11} &= \frac{4s_{63}\, det}{4s_{63} - 8s_{61}} \\
t_{12} &= \frac{2s_{64}\, det}{4s_{63} - 8s_{61}} \\
t_{16} &= \frac{-8det}{4s_{63} - 8s_{61}} \\
t_{21} &= 4(s_{53}s_{61} - s_{51}s_{63})(s_{46} - s_{45}) \\
t_{22} &= s_{51}s_{64}(-2s_{46} + 2s_{45}) + s_{53}s_{64}(-s_{45} + s_{46}) + s_{55}s_{44}(-s_{63} + 2s_{61}) + s_{56}s_{44}(-2s_{61} + s_{63}) \\
t_{23} &= (4s_{63} - 8s_{61})(s_{45}s_{56} - s_{46}s_{55}) \\
t_{24} &= (4s_{63} - 8s_{61})(s_{55} - s_{56}) \\
t_{25} &= (4s_{63} - 8s_{61})(-s_{45} + s_{46})
\end{aligned}$$

$$
\begin{aligned}
t_{26} &= (4s_{53} - 8s_{51})(s_{45} - s_{46}) \\
t_{31} &= \frac{-4s_{61}det}{4s_{63} - 8s_{61}} \\
t_{32} &= \frac{-s_{64}det}{4s_{63} - 8s_{61}} \\
t_{36} &= \frac{4det}{4s_{63} - 8s_{61}} \\
t_{42} &= \frac{det}{4} \\
t_{51} &= 4(s_{46} - s_{42})(s_{51}s_{63} - s_{61}s_{53}) \\
t_{52} &= (2s_{51} - s_{53})(-s_{42}s_{64} + s_{46}s_{64}) + (2s_{61} - s_{63})(s_{56}s_{44} - s_{52}s_{44}) \\
t_{53} &= (4s_{63} - 8s_{61})(-s_{42}s_{56} + s_{46}s_{52}) \\
t_{54} &= (4s_{63} - 8s_{61})(s_{56} - s_{52}) \\
t_{55} &= (4s_{63} - 8s_{61})(-s_{46} + s_{42}) \\
t_{56} &= (4s_{53} - 8s_{51})(s_{46} - s_{42}) \\
t_{61} &= 4(s_{63}s_{51} - s_{61}s_{53})(s_{42} - s_{45}) \\
t_{62} &= (2s_{51} - s_{53})(-s_{45}s_{64} + s_{42}s_{64}) + (2s_{61} - s_{63})(s_{52}s_{44} - s_{55}s_{44}) \\
t_{63} &= (4s_{63} - 8s_{61})(s_{42}s_{55} - s_{45}s_{52}) \\
t_{64} &= (4s_{63} - 8s_{61})(s_{52} - s_{55}) \\
t_{65} &= (4s_{63} - 8s_{61})(s_{45} - s_{42}) \\
t_{66} &= (8s_{51} - 4s_{53})(s_{45} - s_{42})
\end{aligned}
$$

Bibliography

[1] Chae H. An, Christopher G. Atkeson, and John M. Hollerbach. *Model-Based Control of a Robot Manipulator.* MIT Press, Cambridge, MA, 1988.

[2] Dalila I. Argaez. An analytical and experimental study of the simultaneus control of motion and force of a climbing robot. Master's thesis, Massachusetts Institute of Technology, June 1993.

[3] E. Bizzi, F. A. Mussa-Ivaldi, and S. Giszter. Computations underlying the execution of movement: A biological perspective. *Science*, 253:287–291, July 1991.

[4] Jim D. Chapel and Renjeng Su. Coupled stability characteristics of nearly passive robots. *Proceedings of the 1992 IEEE Int. Conference on Robotics and Automation*, May 1992.

[5] K. Cleary and T. Brooks. Jacobian formulation for a novel 6-dof parallel manipulator. *1994 IEEE Conf. on Robotics and Automation*, pages 2377–2382, 1994.

[6] J.E. Colgate and N. Hogan. Robust control of dynamically interacting systems. *Int. J. Control*, 48(1):65–88, 1988.

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

[8] S. D. Eppinger and W. P. Seering. Understanding bandwidth limitations in robot force control. *IEEE International Conference on Robotics and Automation*, 1987.

[9] S. D. Eppinger and W. P. Seering. Three dynamic problems in robot force control. *IEEE International Conference on Robotics and Automation*, 1989.

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

[11] 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.

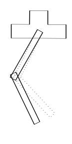
[12] A. A. Goldenberg. Analysis of force control based on linear models. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1348–1353, 1992.

[13] N. Hogan. Impeadance control: An approach to manipulation: Part i - theory, part ii - implementation, part iii - applications. *J. of Dynamic Systems, Measurement and Control*, 107:1–24, 1985.

[14] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE Journal of Robotics and Automation*, 5(1):90–98, 1986.

[15] O. Khatib. A unified approach for motion and force control of robot manipulators: the operational space formulation. *IEEE Journal of Robotics and Automation*, 3(1):43–53, 1987.

[16] Kazuhiro Kosuge and et. al. Input/output force analysis of stewart platform type of manipulators. *IEEE Journal of Robotics and Automation*, 3(1):43–53, 1987.



[17] Tad McGeer. Passive dynamic walking. *International Journal of Robotics Research*, 9(2):62–82, 1990.

[18] Simon Mochon and Thomas A. McMahon. Ballistic walking: An improved model. *Mathematical Biosciences*, 52:241–260, 1979.

[19] David E. Orin and W.W. Schrader. Efficient jacobian determination for robot manipulators. *Robotics Research: The First International Symposium*, 1984.

[20] Gill A. Pratt. A biped robot for real environments. Unpublished grant proposal, 1994.

[21] Gill A. Pratt and Matthew M. Williamson. Series elastic actuators. *IEEE International Conference on Intelligent Robots and Systems*, 1:399–406, 1995.

[22] Jerry E. Pratt. Learning virtual model control of a biped walking robot. Unpublished Project Report for MIT's Machine Learning Class (6.858), 1994.

[23] Jerry E. Pratt. Exploration of virtual model implementation, stability, and performance issues. Unpublished Project Report for MIT's Applied Nonlinear Control Class (2.152), 1995.

[24] M. H. Raibert and J. J. Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102, 1981.

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

[26] K. Salisbury. Active stiffness control of a manipulator in cartesian coordinates. *19th IEEE Conference on Decision and Control*, pages 83–88, Dec. 1980.

[27] K. Salisbury and et. al. The design and control of an experimental whole-arm manipulator. *Proc. 5th Int. Symp. on Robotics Research*, 1989.

[28] K. Salisbury and et. al. Haptic rendering: Programming touch interaction with virtual objects. *Proceedings of the 1995 ACM Symposium on Interactive 3D Graphics*, April 9-12, 1995.

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

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

[31] C. Sunada, D. Argaez, S. Dubowsky, and C. Mavroidis. A coordinated jacobian transpose control for mobile multi-limbed robotic systems. *IEEE Journal of Robotics and Automation*, pages 1910–1915, 1994.

[32] Daniel E. Whitney. Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, 6(1), 1987.

[33] Daniel E. Whitney. Force feedback control of manipulator fine motions. *Journal of Dynamic Systems, Measurement, and Control*, pages 91–97, June 1977.

[34] Matthew M. Williamson. Series elastic actuators. Master's thesis, Massachusetts Institute of Technology, June 1995.