CONTROL TECHNIQUES FOR PHYSICALLY-BASED ANIMATION

by

Michiel van de Panne

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Control Techniques for Physically-Based Animation

Doctor of Philosophy, 1994
Michiel van de Panne

Department of Electrical and Computer Engineering
University of Toronto

Abstract

Determining how an animal should activate its muscles to perform a desired motion is a difficult problem. Solutions to this problem are useful to animators who wish to direct their characters without having to provide all the explicit details of a motion as required in keyframing. This thesis unites some of the ideas from the field of control with the requirements of animators. Four contributions towards solving motion control problems are presented. The first contribution, that of approximating-graph dynamic programming, is an enhancement of a general optimal control technique. It proves useful for problems of low dimension, such as balancing a pole or parking a truck-and-trailer. Our second contribution is a variable terrain walking controller for a human figure. This shows how decomposition of a complex control problem can be used to make it tractable. Our third result consists of two methods of control for the turning motions typically performed by bicyclists and skiers. The control of a simplified physical model can be used as the basis for the more complex motion of the whole body. Our fourth contribution is that of sensor-actuator networks. These are control structures useful for generating walking, hopping, flipping, swimming, and crawling behaviours for a large variety of simulated creatures.

Acknowledgments

It would have been impossible to arrive at the ideas in this thesis without the support of many people. First and foremost, I am greatly indebted to my supervisors, Eugene Fiume and Zvonko Vranesic, who have always encouraged me to pursue my own ideas while at the same time providing invaluable feedback and discussions at every step along the way. As well, the great personal energy and enthusiasm you bring to all your pursuits have not gone unnoticed. I can only hope that you have enjoyed this collective adventure as much as I have!

I wish to acknowledge the contribution of the members of my Ph.D. committee, who thorougly read the thesis and suggested many useful improvements. Thanks go out to Prof. Mark Green, Prof. Demetri Terzopoulos, Prof. Susan Tupling, Prof. Glenn Gulak, Prof. Raymond Kwong, and Prof. Morris Milner.

There are many members of the EECG lab who have contributed greatly to this thesis through their friendship, enthusiasm, and discussions. Steve Jantzi (king of the kitchen seminar series), Bob Francis, John Chappel, Steve Wilton, Keith Farkas, Kevin Chung, Ron Unrau, Orran Krieger, Heather Hinton, Marcel Leblanc, Jane Xin, Duncan Elliott, and Yaska Sankar have all made the time spent in the EECG group very enjoyable and rewarding.

There are equally many members of the DGP lab who have helped throughout this thesis in many ways. Michael McCool, George Drettakis, and Jos Stam have made DGP a very exciting place to be, as have Marc Ouellette, Charles Wuethrich, and Didier Badouel. I thank you all for many interesting discussions.

On a more personal note, I wish to thank Michèle (ma belle!) Danelice for her endless encouragement and support during my Ph.d. work. Your influence on this work is there, even though you may not see it directly! Last, but not least, I have a big thank-you for my parents and Job and Carol, who have helped me take the ups and downs during this work in stride.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

The precise leap of a squirrel between tree branches is an amazing sight to see. Even a grasshopper is capable of exquisite coordination and balance in moving about. The goal of physically-based animation is to be able to control and command such motions, thereby making the animator a director rather than someone who must implement every detail of an animation. Just as a choreographer relies on the ability of the individual dancers to make his or her job easier, the animator will be able to rely on the abilities of the actors involved in any scene. Implementing such a scenario requires endowing the simulated actors (human, animal, or fictitious) with the skills to move about in a natural way. The goal of this thesis is to study how such skills may be synthesized. The human characters in our animations should know how to walk, ride a bicycle, and ski. The creatures should know how to hop, run, swim, and crawl. A key to successfully synthesizing such skills is finding the right representations of the problem to be solved and the possible forms of solution.

It is worthwhile briefly examining where the motion control problem fits in the process of producing animations. The first step of this process is to come up with a summary of the story to be told. This is done in the form of a *storyboard*. A storyboard is a series of sketches of the important scenes to be used in the animation. It is used to give a sense of the structure of the whole animation and the relative length of each of the scenes. A summary of the desired soundtrack is also written at this stage. The next step is to design the characters to be animated and specify their motion. Using traditional animation techniques, this is done by drawing keyframes. Keyframes are frames of the animation where the positions of the characters are completely specified. The remaining frames of the animation are created by *inbetweening,* which involves interpolating for the characters' positions between those shown in the keyframes, as first described in [17].

Using physically-based modelling, the same task of determining character motions is accomplished in several steps. The first step is to create a suitable *physical* and *geometrical* model of the characters. The physical model supplies all the parameters necessary to perform a mechanical simulation of the character. For an articulated character, this means supplying the mass and moment of inertia for each link, as well as defining the constraints in effect at each joint. The geometrical model supplies the geometry that will be used for the graphical display of the character as well as possible for detecting collisions involving the character. The second step is to control the motion of the characters, and the last step is to render the images of the animation. Of these three steps, the first and last steps have been relatively well studied compared to the second step, namely that of controlling the motion of the character in an easy way.

1.1 A Modelling Problem

Motion control can be seen as a modelling problem, where we seek to build a behavioural or motion control model of a person or creature. In doing so, we are in effect modelling part of the nervous system of the character. Such a model could be constructed at various levels of detail. The highest level involves understanding the motivation for a given movement. The lowest level involves understanding the details of how individual neurons and muscle cells operate. The needs of an animator require modelling the running dog at a level between these two extremes. What is the set of rules that govern when one foot should be placed down and another should be lifted? How can these rules be constructed? How can they best be represented?

It is informative to compare the process of conventional modelling in computer graphics with the motion modelling that we wish to perform. Consider generating a sequence of images of what a camera sees when it is flown about through some fixed scene. The end product desired consists of the pixel values in each image for a sequence of perhaps a thousand images. The standard approach for creating such an animation involves creating geometric models of the objects in the scene, modelling the light sources in the scene, modelling the interaction of the light with the objects (e.g., reflection and refraction), and lastly modelling the camera itself. These models lead to a relatively compact description of the phenomena we wish to produce, namely the realistic visual depiction of an environment. The description of the object models and the rendering script taken together are shorter than the description of all the images produced as a result. In a similar vein, we shall seek a compact description of many motions. Consider a person walking over varying terrain for hours on end. A compact representation of this phenomenon is a set of control rules combined with a model of the person and the terrain. We seek a compact representation not only for reasons of efficiency, but also for reasons of insight. Frequently, the most compact representation of a phenomenon also gives the most insight in terms of an understanding of that phenomenon.

In modelling motion control we shall rely primarily on synthesis (of controllers) rather than analysis (of motions). The reasons for this are twofold. First, we wish to be able to control imaginary objects and creatures for which no previous motions exist to analyze. Second, it is a difficult problem to deduce the structure of a controller (i.e., the behavioural model we seek) from a given set of motions. Many different controllers can realize identical motions. Analysis of motions is useful for purposes of comparison, but provides few clues as to the structure of a controller responsible for a given motion.

1.2 The Problem Context

In this thesis, the motion control problem is discussed in the context of creating animations. Some of the results are equally well applicable in the context of other fields, however. In our work we seek to control physically-realistic mechanical simulations of objects, people, and various creatures. This is no different than controlling the real thing, provided that our simulations are sufficiently realistic and the control can be executed in real time. With this in mind, the work presented in this thesis also has relevance to the fields of control, robotics, biomechanics, autonomous systems, and possibly evolution as pertaining to physical structure for locomotion. In many of these fields, simulation already represents an indispensable tool for performing research [35] [56] [83] [109]. We shall choose to present our work largely in the context of animation. The influence of related research in robotics, control, and biomechanics on our work is nevertheless apparent. The goals of such research are ultimately different, but some of the methods and insights are shared with the work we shall present. Lastly, the work presented here provides some contributions to these same fields.

With respect to animation, the physically-based techniques presented here are intended to augment traditional animation techniques, not to replace them. The purpose of most cartoon animations is to provide a caricature of life in a world rather than to accurately reflect it in all its detail. However, the laws of physics have an impact on all real motions in the world, and hence impact caricatures of these motions as well. It is thus useful to first examine how to control objects that are subject to the real laws of physics.

An animator will ultimately need a different interface to motion control models than a roboticist. In particular, the degree of autonomous skill exhibited by any character or creature should be under the animator's control. The animator should also be able to create a particular instance of a motion by specifying a set of desired parameters. For example, these might be the desired walking speed or hopping height. Some of our work will reflect a predisposition to providing the animator with an intuitive interface.

In this thesis we seek to gain an understanding of general control techniques applicable to animation. Each control technique is nevertheless best suited for solving a particular category of control problems. For this reason, we shall present several control algorithms in lieu of restricting our focus to just one. There are many ways of looking at the problems to be solved and it is thus important to examine the alternatives available.

The work presented in this thesis builds upon basic results from several different fields. Although we endeavour to explain as many fundamentals as possible, we shall assume some basic knowledge of the reader in the fields of control and dynamics. A general familiarity with control problems and optimization techniques will be useful. A basic knowledge of dynamics is also helpful. The focus of this thesis lies in the field that has come to be known as *physically-based animation*. Chapter 2 provides an introduction to this field and some of the subproblems involved.

1.3 Thesis Overview

A roadmap to this thesis is given in Figure 1.1. This thesis explores several different representations for motion control problems and their solutions. We begin by discussing the nature of the problem to be solved in Chapter 2. Given that it can be cast as an optimal control problem, the first proposed technique is one which determines globally optimal solutions. This is described in Chapter 3. The major drawback of globally-optimal solutions is that they are prohibitively expensive to use for any but the simplest systems. The focus then becomes one of how to reduce the order of the physical models in order to make globally optimal solution practical. The walking work of Chapter 4 describes this process for a biped walking controller. Chapter 5 describes a how a reduced-order physical model can be used for the synthesis of turning behaviour. Lastly, Chapter 6 presents a stochastic approach to dealing with the control of complex systems. At present, it seems this last method has the most promise.

A more detailed overview of each chapter is as follows. Chapter 2 discusses various classifications of motion control problems and their solutions. One can classify control schemes in many different ways: by their formulation, the type of information they use (sensor or state), the class of systems they are suitable for, and their method of control synthesis. These classification methods are examined with regard to the motion problems we would like to solve, namely balancing a pole, riding a bicycle, human walking, and fish swimming, among others.

Chapter 3 presents an optimization technique for calculating a general, state-based controller. Our goal is to derive a realistic control method that can move objects in real-time. We call this technique *approximating-graph dynamic programming* (AGDP). The controllers

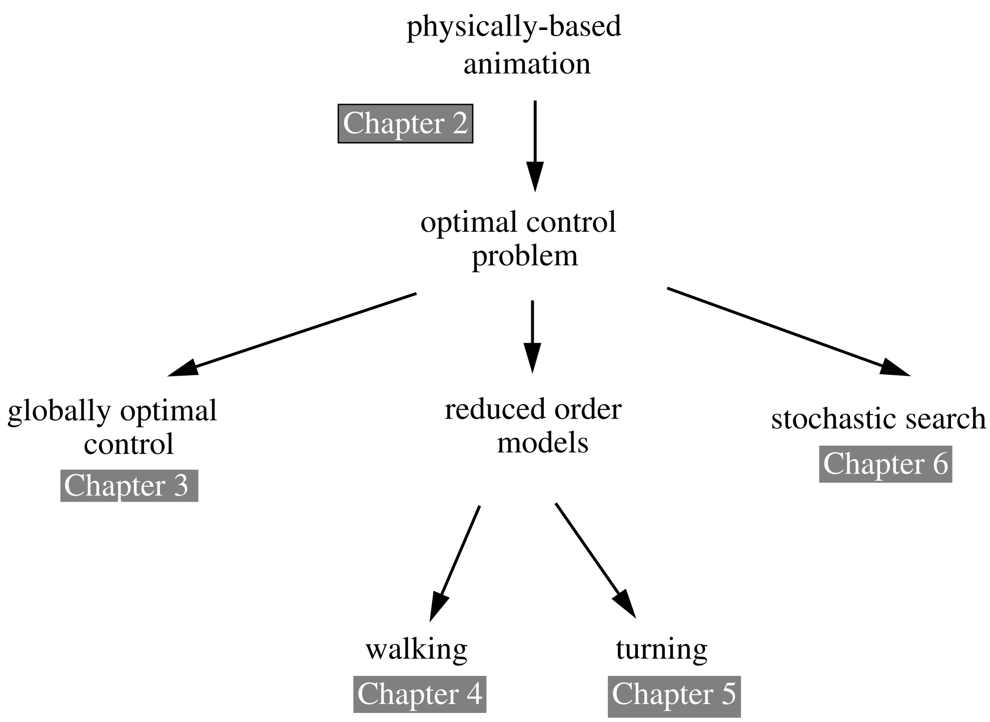


Figure 1.1 Thesis overview.

produced are useful in the control of low-dimensional systems, such as those involved in balancing a pole or parking a truck. The technique is an adaptation of dynamic programming to solve one-point boundary value problems for continuous systems. The resulting controllers are table-based, with the state of the system being used as an index to retrieve desired control values. Results are shown for a variety of nonlinear systems, including two fourth-order systems.

While the technique described in Chapter 3 is general, its order of complexity is exponentially dependent on the order of the system, making it inappropriate for the control of complex systems. In order to control something as complicated as a model of the human body, we need to perform some decomposition of the control functions. Chapter 4 shows how this can be done for a walking controller. The most important subcontroller is calculated using the dynamic programming technique of Chapter 3. The other subcontrollers have a simpler form. All the subcontrollers operate in a coordinated fashion by using a common piece of information we call the *phase*. The control scheme is shown to be capable of generating the internal torques necessary to make our human model walk over varying terrain as well as up stairs.

Chapter 5 presents control solutions for turning motions, such as bicycling, snowboarding, and skiing. All these activities can be modeled using an identical, simplified physical model. Two control techniques are presented. The first gives control in terms of three parameters dictating the turn frequency, sharpness, and heading of the turns. The second allows the user to draw a desired path to be followed while avoiding obstacles. The control solution in this case is calculated by a branch-and-bound optimization technique.

Chapter 6 takes a rather different approach than the preceeding chapters. The aim here is to examine the power of very simple controllers. Whereas the controllers of Chapters 3 and 4 take on the order of 10 kilobytes to specify, the *sensor-actuator networks* (SANs) introduced here take on the order of 70 bytes to specify. The user supplies the configuration of a mechanical system (i.e., a model of the desired creature) that has been augmented with simple sensors and actuators. It is then possible to automatically discover many modes of locomotion for the creature, each having a SAN as a controller. A SAN consists of a small, nonlinear network of weighted connections between sensors and actuators. The control they produce is robust, meaning that many of the creatures are capable of locomoting over bumpy terrain. Results for ten different creatures are presented.

Chapter 7 summarizes the contributions of this thesis. Conclusions about the merits of the various techniques are made. The general optimal control technique of Chapter 3 is useful for systems having few degrees of freedom (less than 6) but is not suitable for direct application to more complex systems. The specific controllers for walking and turning presented in Chapters 4 and 5 give some insight into how the control for more complex systems can be decomposed. The method of Chapter 6 represents perhaps the most exciting approach in terms of the resulting motions and provides some of the most promising future directions.

It is not necessary to read all the chapters of this thesis in the order they are presented. In particular, those with a background in animation will likely find the work of chapters 5 and 6 more directly tied to problems in animation. The work in chapters 3 and 4 is a more suitable starting point for those with some background in control. Portions of this thesis can also be found in [95]- [100].

Chapter 2 Background

Physically-based animation seeks to achieve realistic motion by ensuring adherence to the laws of physics. The simplest such animations involve *passive* objects, which require only simulation to generate their motion. A swinging chandelier or a chair tumbling down a flight of stairs are good examples of this. The initial state of the system and the laws of physics define the motion in the animation. However, more interesting moving objects are usually *active*. These systems have internal actuators or muscles and can thus influence their own motion.

The problem of physically-based animation of active objects such as humans and animate creatures consists of the control and simulation subproblems as shown in Figure 2.1. This thesis shall focus almost exclusively on the control problem. Advances in mechanical simulation have made it easier to simulate the motion of the type of characters we wish to animate, although writing a general mechanical simulator is still an onerous task [87]. The task of synthesizing realistic control for these same characters has not been well solved.

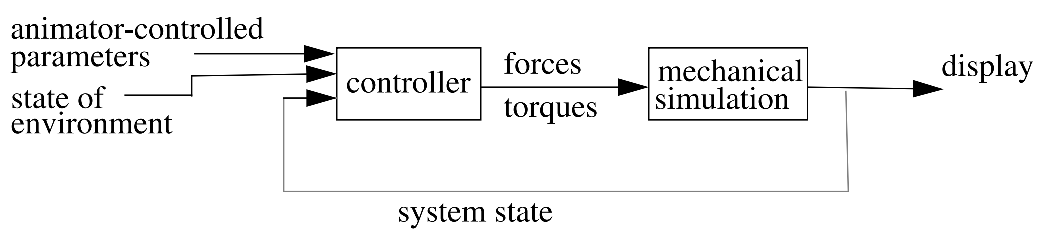


Figure 2.1 The control and simulation problems.
The controller determines how the muscles or motors act, while the simulation applies the laws of physics to determine the actual motion.

A controller directs the actuators, usually in terms of the forces or torques to be applied. The simulation then determines the effect of the actions. The dotted line indicates the possible use of feedback, which can greatly simplify the controller needed to accomplish a given task. The controller can in general have other external inputs that affect its actions. In particular, the animator might like to specify parameters that further define the motion to be achieved, such as the swimming speed for a fish or the sharpness of a turn for a bicyclist. Inputs from the environment are often useful in increasing the autonomy of the controlled object. For example, providing a walking controller with information on the height of the next step allows for autonomous walking over varying terrain.

This chapter begins with a brief historical overview of physically-based animation and the motion control problem. We then examine several different ways of classifying motion control problems. Relevant previous work in the area is summarized and classified according to the type of system being controlled. The four contributions of this thesis are similarly classified. This is followed by a discussion of related problems in other fields.

2.1 A Historical Overview

It is only recently that the control problem has come to be addressed as an important problem to be resolved in creating physically realistic animations. This can be observed by looking at the progression of animation research and how it has made use of results from other fields along the way. The purpose of this section is to give a sense of the history of the motion control problem in the context of animation.

Computer-assisted animation dates back to the early 1970s [17] [27] and is firmly founded upon the use of keyframing. Keyframing is a technique used in traditional cel-based animation. The animator specifies the position of the figure to be animated at key instances in time. These positions are called keyframes. The position of the figure at all instances in time are calculated by performing interpolation between the keyframes. The interpolation of each degree of freedom is usually treated independently and can be performed using linear or higher order polynomials. The motion in most well known works of computer animation to date has been produced using this method, as it places no restrictions on what the animator can do. Its biggest drawback is that the animator has to expend a lot of effort in specifying and modifying keyframes in order to achieve the desired motion. The amount of effort required to create an animation in this way is almost directly proportional to its length. A discussion of how traditional keyframing techniques can be used in computer assisted animation can be found in [62].

A variety of techniques have been developed to simplify the job of specifying keyframes. Examples of this can be found in [8], [18], [19], [29], [55], [84], and [110] among others. It was not until the work of Armstrong and Green [4] and Wilhelms [103] [104] in 1985 that the use of physical models was proposed as an alternative method for producing animations. This early work, and other related work [48] concentrated primarily on creating and simulating the mechanics of human-like articulated figures. The techniques used to derive the equations of motion generally came from the domains of mechanical engineering and robotics. The term *physically-based animation* emerged to describe the animations based upon the motion of underlying mechanical models.

While being able to perform physical simulations of mechanical models approximating human figures was a noteworthy accomplishment, the first physical simulations were not readily usable because of the lack of appropriate control methods. It is a difficult problem to determine how the muscles in a human figure or other creatures should act to achieve a desired motion. Some of the earlier techniques required the animator to directly specify the torques as a function of time [5] [105]. The use of positional constraints has also been suggested to reduce the degrees of freedom to control [5]. While the use of a physical model in these cases ensures that the laws of physics are maintained, it creates some difficulties for the animator. The manipulation of torque functions is less intuitive than directly manipulating positions, as in keyframing. The motions can also still be unnatural, despite the fact that they are physically plausible.

The method of *spacetime constraints,* introduced in 1988, represented a new approach to the control problem [108] [25]. This work is very relevant to the goal of this thesis, as it represents a general technique for controlling physically-based animations. The spacetime constraints technique deals with two-point boundary value problems, where the animator specifies a starting and ending state for the object. A time interval for the motion must be given, as well as a reasonable guess at an initial trajectory. This is shown in Figure 2.2. Lastly, an optimization function is required, which defines how the motion is to be performed. Typically, this is a function of the control energy required to perform the motion, hereby assuming that desirable motions are efficient as well.

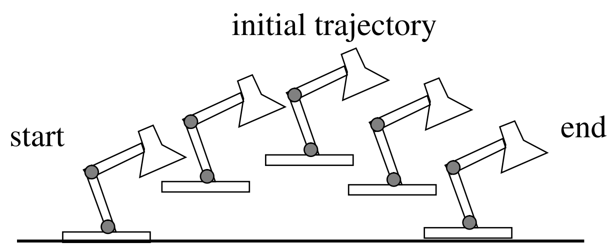


Figure 2.2 Motion specification for the spacetime constraints method. The animator specifies a starting state, ending state, and an initial guess at the trajectory. Preference for how the motion must be performed is conveyed through an optimization function.

The solution technique must accomplish two things. First, the laws of physics must be satisfied. Second, the chosen optimization function must be minimized. The solution proceeds iteratively, where each iteration makes adjustments to the entire trajectory at once. Each iteration minimizes the optimization function while trying to satisfy the physics constraints. This is accomplished using a variant of sequential quadratic programming.

The space-time constraints technique has several shortcomings. One of the problems with trajectory-based solutions is that they are not reusable. The solution can only be used in the specific situation for which it was calculated. A solution in the form of a controller is more general and reusable because controllers make use of feedback to perform the desired task. The solutions supplied by spacetime constraints can be physically implausible in some cases. This can occur in cases where it is impossible for the object to perform the desired transition from the start state to the end state in the given amount of time.

Controllers that are specialized and hand-tuned to their task have appeared in the graphics literature in the past five years. These control hopping motions [82] [83], snake locomotion [73], unicycling [44], pumping a swing [44], cockroach locomotion [71], and walking [91] [97] among others. There is still a lack of a general technique for the construction of such controllers. This makes it difficult to construct controllers for new objects. Frequently these specialized controllers have many parameters that require fine-tuning by hand and do not provide intuitive methods for animators to further influence the motions. A good survey of recent work can be found in “Making Them Move” [7].

Many of the figures that animators would like to control represent difficult systems for conventional control techniques. These difficulties are caused by the non-linearities involved, the discontinuities that occur, and the very general specification of the problem to be solved. Furthermore, the goals of control theory and animation tend to be different. In animation it is the quality of the resulting motion that is of concern, whereas the field of control is also concerned with issues of proving stability and performance under all conditions. The work presented in [15] is an example of the use of results from control theory for animation in a limited way. In general, however, crossover of this nature has been rare because the difficult systems that most animated figures represent are not yet well understood in terms of control theory.

While the behaviour of groups of objects is beyond the scope of this thesis, this has also been an active area of research [29] [39] [84]. Once a better understanding of solutions to low-level control problems has been obtained, this research will provide directions as to how to build the higher layers of behaviour control.

2.2 Classification Criteria for Control Problems

One of the first questions that needs to be addressed is whether control problems for physically-based animation are different from other types of control problems. Can the task of walking or riding a bicycle be compared with that of controlling the mix of chemicals in a chemical reactor? There is a large body of literature dedicated exclusively to control problems and we should seek to make use of it if possible. Our path through the previous literature will be defined by devising orthogonal criteria for classifying control problems and their solutions.

The simplest criterion for classifying a control problem is based upon the *type* of system being controlled. There are many well known control results for linear systems and as a result it is certainly useful to determine whether the system to be controlled is linear or non-linear. Most real systems are not linear in their entire range of operation but many can be treated as being linear in a useful range of operation. This is not the case for the nonlinear systems we are interested in animating, however. These systems can have discontinuous motion due to collisions as well as often being unsuitable candidates for linearization in their useful operating range. For example, some of the objects we shall deal with move by hopping, thus colliding with the floor as well as passing through many internal configurations.

One can classify a control solution based upon its use of *feedback*. A *closed-loop* control solution making use of feedback yields a controller, whereas an *open-loop* solution produces a trajectory. In practice, controllers are more useful and flexible than trajectories. Once a controller is calculated, it can be reused in many different situations. Trajectories require recalculation for every new situation. A controller is thus a more compact representation of a desired motion. The work we shall present is in the form of closed-loop controllers, with the exception of the open-loop control of turning in Chapter 5.

Controllers can be further classified according to the *type of feedback* they use. Many controllers make use of state feedback. The *state* of a mechanical system is the set of numbers required to specify the position and velocity of every point on the object at any instant in time. State feedback is convenient to use because it ensures that all known information about the position and velocity of the object is available to the controller. An alternative to the use of state-feedback is sensor feedback. We can instrument a body with a variety of sensors that can serve directly as useful information to the controller. It is not guaranteed that the entire state of the system can be reconstructed from the sensory information, thus sensor-based feedback only provides a subset of the information available using state-based feedback. We shall see in Chapter 6, however, that this is not necessarily a disadvantage. Examples of sensors might be as simple as touch sensors, which turn on when they are in contact with something. Chapters 3 and 4 make use of conventional state-feedback. Chapter 4 presents an open-loop control algorithm.

We can also classify a controller according to its optimality. A controller can be synthesized to ensure it optimizes a desired evaluation metric. Functions of the time and control energy required to perform a motion are often suitable evaluation metrics for natural looking motions [2]. The main problem with optimization techniques is the difficulty and cost of calculating solutions. Arriving at a globally optimal solution is frequently prohibitively expensive, especially as the system to be controlled becomes more complex. Almost all the results to be presented in this thesis make use of some kind of optimization technique. However, the optimization functions and optimization methods are different for each result. The technique of Chapter 3 searches for globally optimal solutions while the others search for locally optimal solutions.

The order of complexity of a control solution can also serve as a basis for classification. Although in general different control techniques usually solve different problems, it is still possible to examine how the control techniques scale with the complexity of the system being controlled. The cost of the dynamic programming technique introduced in Chapter 3 scales exponentially with the order of the system to be controlled. Other techniques often remain unproven in their ability to control complex systems. We believe that the sensor-actuator networks of Chapter 6 can eventually provide a feasible route to controlling more complex systems. Chapters 4 and 5 show how the complexity of a problem may be reduced for the specific cases of walking and turning sports, respectively.

Another classification criterion relates to the structure of a controller and its biological plausibility. Controllers can have different structures, with the structure defining the operations to be performed in generating the desired control at any instant of time. Some of these structures are more plausible in a biological setting than others. Control solutions resulting in the calculation of fixed trajectories instead of controllers are somewhat implausible, because it is clear that animals have the ability to make compensations and changes to a motion as it is being performed. One could also envision a controller being defined in terms of a set of rules. An example rule might be "If the system is approximately in state A and event B just happened, then apply control C." The extent to which such a rule-based system is biologically plausible or to which it is really just a paraphrasing of the function being performed by some underlying structure in a real animal remains an open question. We will not comment on the biological plausibility of our controllers, but recognize it as an interesting question to be answered.

A last means of classifying a control scheme is in terms of how specific or general a technique it is. Does a given control technique apply only to a blind mouse with three legs, or is it more general? In this thesis we shall arrive at two general control techniques and two specific control techniques. The dynamic programming technique described in Chapter 3 and the sensor-actuator networks described in Chapter 6 are general techniques. The work on walking in Chapter 4 and that on turning in Chapter 5 are more restrictive in their application.

2.3 Related Work

A large body of literature relevant to the control problems we seek to solve exists in the domains of animation, robotics, and biomechanics. We shall discuss only the work which is particularly relevant to ours. We shall use the various *types of systems* being controlled to lend structure to our discussion.

2.3.1 Linear Systems

Linear systems are often the simplest to control. The canonical form for a linear system is given by [23]:

$$\dot{x} = Ax + Bu \qquad [2.1]$$

$$y = Cx + Du \qquad [2.2]$$

where x is the state vector, y is the output vector (i.e., the quantities one wishes to control), u is the input vector (i.e., the quantities one can control), and $\dot{x}$ is the time derivative of x. Many results for controlling linear systems exist. An important result exists for optimal regulation in the form of the linear quadratic regulator (LQR). The goal is to minimize a quadratic function of the control input u and output y while driving the output y to its null state. Specifically, the optimization function is defined as:

$$J = \int_0^\infty (y^T Q y + u^T R u)\, dt \qquad [2.3]$$

where Q is assumed to be positive, semi-definite and R positive definite. The solution of the LQR problem has the form of a feedback matrix F where $u = Fx$. The feedback matrix F can be obtained by solving the Riccati equation [51]. The solution is not completely analytic in that the Riccati equation must be solved through iteration. In [15] it has been shown how this linear control theory can be applied to the animation of some linear systems. Unfortunately, most systems to animate are very nonlinear. Contributions to nonlinearity in the models of humans and creatures include gravity (for rotating objects), collisions, friction, joint limits, and rotary joints placed in series. The problems we shall be working with are all nonlinear in nature, with the exception of one example in Chapter 3.

For the types of control problems we shall be dealing with it will prove useful to classify nonlinear systems into three categories. We shall define a nonlinear system as being *smooth*, *statically stable*, or *statically unstable and not smooth*. These categories are not disjoint; there are smooth, statically-stable systems.

2.3.2 Smooth Systems

We shall define smooth systems as being those that are not necessarily linear, but whose state variables (positions and velocities) are C^0 continuous over time,[1] meaning that there are no jump discontinuities in velocity. Smooth systems form an important category because we can calculate the derivatives necessary for the application of various optimization techniques.

1. If velocities are C^0 then positions are C^1.

Systems that undergo collisions are excluded from this category because of the instantaneous change in velocity. Most optimization techniques cannot properly handle such discontinuities of the system state. Smooth systems have states that follow a continuous trajectory through state space.

Many solutions to smooth control problems typically involve iteratively refining a state trajectory. The problem can be posed as a two-point boundary problem, where the path between the boundary points is optimized with respect to a desired evaluation metric [108], while the laws of physics act as a constraint upon the system. With this view, it is not necessary to separate control from simulation as we have depicted in Figure 2.1. Some solutions take alternating steps in optimizing the evaluation metric and meeting the physical constraints [25] [108]. In this case the physics constraints are solved for explicitly. It is also possible to have the physics constraints expressed implicitly in the optimization process, as exemplified in [91] to create walking animations.

It is also possible to pose the problem as a one-point boundary problem, in effect solving for many two-point problems simultaneously [93] [94]. The solution in this case takes the form of a controller instead of a trajectory, allowing it to be used in many different situations (i.e., from many different starting points). Chapter 3 will introduce an improved method for calculating such one-point solutions. In our work on walking in Chapter 4 we approximate a walking human figure as a smooth system. Walking does not truly represent a smooth motion because the feet regularly impact the ground. The hip and the trunk generally maintain smooth and continuous behaviour, however, and this allows us to determine a controller based upon the assumptions of a smooth system.

Solutions to smooth problems are often useful for animation. The takeoff and aerial portion of many jumping motions fall within this class. A swimming fish, crawling snake, bicycling (turning) person, parking car, and walking person can all be treated as smooth systems. Chapters 3, 4, and 5 deal with smooth systems, while Chapter 6 deals with systems that are not smooth.

2.3.3 Statically-Stable Systems

Statically-stable systems are those that can be effectively controlled through kinematic means.[1] This means that the motion of a creature can be halted in mid-action and then resumed without it falling over or otherwise collapsing. These systems can often be controlled by passing a cyclic motion through the body or legs. Realistic animations of snakes [73] and cockroaches [71] have been performed using suitable controllers. Fish swimming and spiders walking could also be considered as statically-stable behaviours at low speeds. Many creatures are statically stable during some parts of a motion and not during others. We classify these as being statically unstable. The statically-stable systems we deal with in this thesis are a parking truck, swimming fish, and a *crawler* creature. Controllers for statically-stable systems can be relatively simple to construct because the momentum of the body parts does not have a large influence on the motion of the body as a whole. As a result, such controllers can often be position-based. As well, the timing of the various phases of a motion is not as critical as it is for statically-unstable systems. Most robot arms are statically stable. Most of the systems we shall deal with are not, including the walking figure of Chapter 4, the turning figures of Chapter 5, and the walking/hopping creatures of Chapter 6.

2.3.4 Statically-Unstable and Non-Smooth Systems

Many interesting things we would like to animate are nonlinear, not smooth, and statically unstable. One need only consider running, hopping, flipping, or walking motions, among others. There has been little work done in the control of such systems. Systems in this class are not necessarily difficult to simulate, but are difficult to control. The mathematical tools that could be used to calculate control for smooth systems cannot cope with the discontinuities. Similarly, the position-based control techniques applicable to statically-stable systems do not apply. The work we shall present in Chapter 6 presents a method which is successful at determining control for some systems of this class.

1. We use kinematic control to refer to position-based motion control schemes.

2.4 Related Work in Other Fields of Research

The control problem we are trying to solve in the context of animation can be considered to be a superset of the problems at the heart of several other fields of research. Solutions to problems in the fields of robotics, biomechanics, control, and ‘artificial life’ are all of possible use in animation. The reverse is not necessarily true because some control solutions appropriate for animation will be infeasible or impractical to implement on real robots. On the other hand, a physically-based animation environment can provide a good testbed for work in robotics, biomechanics, and control. Given that the animation control problem can be considered as encompassing at least these four areas of research, it is clearly a large and difficult problem.

Work in robotics is of importance to the problems we wish to solve. Forward and inverse kinematics, as well as forward and inverse dynamics are important concepts from robotics that have a direct application to the animation of articulated figures [6]. Work performed on biped locomotion is relevant to the animation of human walking (§4.1). The path planning problems considered in mobile robotics are also possibly useful in animation [61]. The construction of intelligent autonomous systems is one of the goals that robotics and animation have in common. This involves creating systems that can sense their environment (real or virtual) and make appropriate responses.

Biomechanics is the study of the mechanical bases of biological activity, especially as pertaining to muscular activity. It provides results that are useful for the synthesis of natural-looking motions in animations. Muscles have many properties (e.g., strength, elasticity) that influence the type of motion produced. Biomechanics provides a study of these properties as well as investigating schemes for controlling muscles to yield desired motions. For example, biomechanics examines what kind of optimization functions can be best used to model human and animal motions. Examples of the use of physical simulations for biomechanical studies can be found in [1], [2], [72], [81], and [109] among others. In summary, biomechanics can help with the building of appropriate physically-based models, and help with the synthesis of suitable control for the models.

Control theory can provide tools useful in solving the animation control problem [56]. While control theory is not concerned with the natural-looking quality of motions, appropriately natural motions might be obtainable by applying suitable constraints and optimization functions to the problem. Control theory is at the base of much of the work carried out in robotics and biomechanics.

Lastly, an area of research that is loosely referred to as 'artificial life' seeks to solve some of the same problems we do [64]. This field is characterized by simulations of simple creatures operating in an environment, usually coupled with a replication process. Many of the simulated creatures are defined by a compact code, which subsequently determines the creatures' behaviours when they are placed in an environment. Algorithms mimicking genetic reproduction, mutation, and natural selection are often used as a method to synthesize a desired behaviour code. Most creatures being experimented with are still too simple to be interesting for the purposes of animation. However, the goal is to seek general encodings of behaviours and general synthesis algorithms, both of which would be useful for the animation control problem.

2.5 Conclusions

Physically-based animation provides an alternative to the use of keyframing. However, its use has not been widespread. Only the simplest elements of physically-based animation have been used in commercially-available animation systems. There are several reasons for this, including a lack of general control techniques and the difficulty of implementing robust and general mechanical simulation algorithms.

There are many possible ways of defining control problems. There is no clear choice of a best method for posing a control problem in the context of animation. In some cases we may wish to achieve a very specific motion. In others, we may seek a motion that best mimics the natural motion of a real person or creature. We have outlined some of the possible ways of classifying control problems. We have discussed previous work in terms of the type of systems being controlled. For this, we have restricted ourselves mainly to literature directly pertinent to animation problems. In the subsequent chapters we shall refer to work from other fields when it relates specifically to the work being presented.

The work presented in the following four chapters covers a variety of system types. In two of the chapters we present general control techniques, while in two others we present more specific control techniques for the problems of human walking and turning motion in sports. The work on walking is related to one of the general control techniques presented. The work on turning serves as an example of how simplified physical models can be used in animating more complex figures. Lastly, the sensor-actuator networks of Chapter 6 represent a technique very much different from the previous chapters in that stochastic techniques are used to synthesize controllers for nonlinear, non-smooth systems.

Chapter 3 Approximating-Graph Dynamic Programming

Dynamic programming is a well known global optimization technique for the control of smooth systems. We shall show it to be a useful method of control for such tasks as balancing a pole and parking a semitrailer truck. It will also prove useful in creating a walking controller for a human figure in Chapter 4. We use dynamic programming to solve one-point boundary value problems for continuous systems. We call our method *approximating-graph dynamic programming* (AGDP). This method introduces several improvements upon previous methods for this kind of problem. In the context of animation, the control variables being solved for will be the internal forces and torques of the animated figure, corresponding to its muscles or motors.

The AGDP technique has several properties that make it promising for solving animation control problems. A controller produced by AGDP takes the form of a lookup table, which allows an animator to use it in an interactive, real-time environment. There are few restrictions placed on the choice of optimization function. A useful optimization function for the purposes of animation is typically based on the time and energy required to complete a given motion [15] [108]. The AGDP technique is also able to generate controllers for a variety of systems. This allows an animator to treat different kind of problems in a uniform fashion. For example, the problems of backing-up a truck and balancing a pole-and-cart system are treated within the same framework. A last useful property for animators is that motions can be chained together with relative ease by exchanging controllers when desired [93].

Dynamic programming is a very general optimization and decision-making technique that has many applications. In many cases, however, it can be computationally prohibitive. We shall introduce a new stageless version of dynamic programming that produces numerical approximations to optimal control laws for continuous systems. The method creates an approximating graph that models the allowable state transitions in the continuous system. Dynamic programming is applied on the approximating graph to obtain a control table. Forward dynamic programming can be shown to be the exact dual of reverse dynamic programming in this formulation. We evaluate the AGDP approach for a variety of problems, including two fourth order systems. Lastly, some possible extensions to the technique are proposed.

The AGDP method is the result of several improvements made to our previous related work [93] [94]. Although the problem being solved is the same; our solution differs in several respects. The most important of these are the use of the approximating graph and the handling of edge conditions. We also compare our results to known analytical solutions, something which was not done in our previous work.

AGDP views a control problem in terms of a path-planning problem. Two closely-related methods also based upon this viewpoint are presented in [28] and [45]. Our work formalizes the construction of the approximating graph, suggests new solutions to some of the discretization problems, and shows that the technique can be applied to more complex problems than previously considered.

3.1 Introduction

Dynamic programming is known as a useful analytical and numerical technique for determining optimal solutions to control problems. All dynamic programming methods have their basis in Bellman's principle of optimality [11]:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

We shall employ the notation of Larson [59] because of the relevance of his work to the method we propose. We shall set out to solve the particular problem of reaching a desired set of *destination states*, x_D, in an optimal way from any starting state within a given domain S. In the context of animation, the domain represents the space of all possible positions and velocities for all degrees of freedom of the system. The destination state represents the desired end configuration (including velocities) of the motion for any figure. The optimization criterion to be minimized is any monotonically increasing function of the form

$$J = \int j(x, u, t)\, dt$$

The partial differential equations governing the motion of the dynamical system are given by $dx/dt = f(x)$, and can be nonlinear. In the remainder of this section, we describe previous work on dynamic programming for continuous systems, look at some of its problems, and point to where our proposed method will differ.

3.1.1 Review of Dynamic Programming

Before examing how to apply dynamic programming to a continuous system, it is worthwhile to reexamine the operation of dynamic programming in a discrete setting, where the procedure is well defined. The simplest illustration of dynamic programming is effected on a system where the allowable state transitions can be represented by a directed acyclic graph (DAG), as shown in Figure 3.1.

In this example we seek to find the least-cost path from the initial node A to the destination node D. A graph node is uniquely defined by (x, k) where x is the state and k is the stage. Dynamic programming solves the problem without explicit enumeration of all possible paths. Instead, the careful application of Bellman's principle of optimality results in all possible paths being evaluated implicitly with a much lower order of complexity algorithm. This is done by examining the problem as a multistage decision process. The decisions at each stage depend only

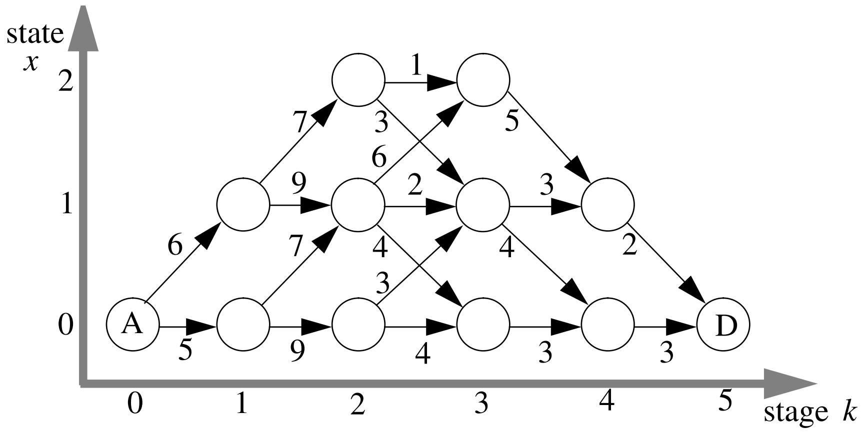


Figure 3.1 A discrete optimization example. The weights on each arc denote the cost of the corresponding transition. The object is to find the least-cost path from node A to node D.

upon the transitions available and the costs at the next stage. This is reflected in the functional equation of dynamic programming, which can be stated as follows:

$$I(x, k) = min_u(L[x, u, k] + I[g(x, u, k), k+1]). \tag{3.4}$$

$I(x, k)$ represents the minimal cost to go from node (x, k) to the final node D; u enumerates the choices (exiting arcs) available at each node; $L(x, u, k)$ represents the cost of taking a given choice, and $g(x, u, k)$ is the state transition function. Thus $x_{k+1} = g(x, u, k)$. We use the notation x_k to refer to the node (x, k). The control variable u will represent the motor or muscle force we wish to determine in the continuous case.

In the discrete optimization problem being posed, we know $I(0, 5) = 0$ and wish to determine $J_{opt} \equiv I(0, 0)$. This can be readily done by applying the functional equation to evaluate $I(x, 4)$, $I(x, 3)$, $I(x, 2)$, $I(x, 1)$, and $I(x, 0)$ in that order. Because the solution propagates away from the final desired state and effectively proceeds in a backwards fashion, the solution process defined by equation 3.4 is often called *reverse dynamic programming* (RDP) [60]. The same solution could also be calculated by having the solution advance in a forward manner using *forward dynamic programming* (FDP). The functional equation for forward dynamic programming can be written as follows [59]:

$$I'(x, k) = min_u(L[g^{-1}(x, u, k), u, k-1] + I'[g^{-1}(x, u, k), k-1]) \tag{3.5}$$

$I'(x, k)$ represents the minimal cost to get from the initial node A to the node (x, k). $g^{-1}(x, u, k)$ is the inverse of the state transition function, namely $g(g^{-1}(x, u, k), u, k-1) \equiv x_k$. To apply equation 3.5 to our problem, we know $I'(0, 0) \equiv 0$ and wish to determine $J_{opt} \equiv I'(0, 5)$. This is done be evaluating $I'(x, 1)$, $I'(x, 2)$, $I'(x, 3)$, $I'(x, 4)$, and $I'(x, 5)$ in that order.

In solving the dynamic programming functional equations given by equations 3.4 and 3.5., we implicitly determine an entire family of solutions. This was called *invariant imbedding* by Bellman [11]. The solutions to our discrete problem are shown in Figure 3.2. For the RDP formulation, we know the optimal path from any state and stage to the final state D. Similarly, for the FDP formulation, we know the optimal path to any state and stage from the initial state A.

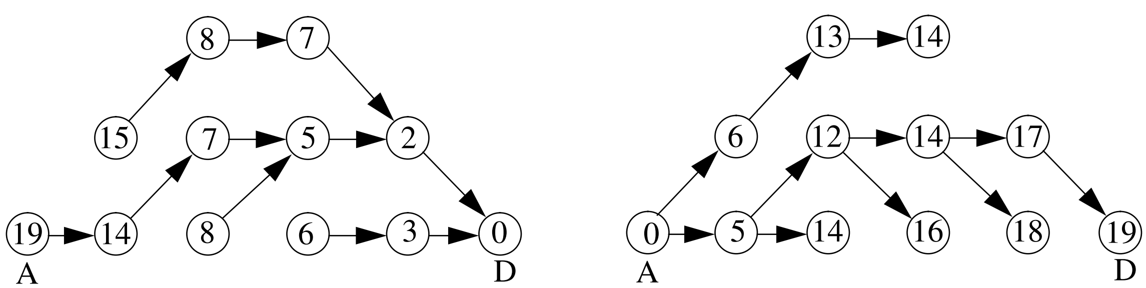


Figure 3.2 Optimal RDP and FDP solutions.
The cumulative costs are indicated at each node.

3.1.2 Optimal Controller Synthesis

The optimal family of solutions obtained from RDP has the particularly nice property that it can be used as a closed-loop controller. For every state and stage, we know the optimal arc to take (i.e. the optimal input to apply). When the set of destination states x_D is fixed and we are dealing with time-invariant systems, the RDP solution provides an optimal state-feedback controller.

Dynamic programming solves two-point boundary value problems by solving one of the associated one-point boundary value problems. The end goal in the two-point case is an optimal trajectory. It has shown that several optimizations can be made when solving two-point boundary problems [66]. These techniques generally use iterative region-limiting strategies.

We shall pursue in depth the use of dynamic programming to solve one-point boundary problems to create optimal controllers. Once an optimal controller for a given process has been calculated off-line, its on-line use requires very few calculations. However, optimal trajectories must be recalculated when the system state deviates from the planned trajectory. The family of solutions obtained using FDP are also useful, although they cannot act as a controller. In this case, we can determine the open-loop control necessary to take a system from a fixed initial state to any desired final state. This can be done with few calculations once the FDP solution has been precomputed and stored in a table.

Dynamic programming is most often described as a multistage decision process [60]. A stage variable is explicit in the description, usually corresponding to a fixed time step Δt for control problems. The optimal control $u(x, k)$ is thus determined as a function of the state and stage. We are interested in time invariant systems, where the optimal control is determined strictly as a function of the system state. In this case, the concept of stages is not as useful as before. We will introduce a new dynamic programming formulation that avoids the use of fixed-length stages. In this respect it is similar to the state-increment dynamic programming proposed by Larson [59].

3.1.3 Problems to Overcome

A difficult problem in dynamic programming is that the functional equations as given by equations 3.4 and 3.5 are not straightforward to apply to continuous systems. Most previous work on the discretization of continuous systems for dynamic programming makes use of *stages*, which represent discrete points in time. Typically the state space, inputs, and time are all discretized into the sets X, U, and K, respectively, giving $x^{(i)} \in X\{x^{(1)}, x^{(2)} \ldots, x^{(N_x)}\}$, $u^{(j)} \in U\{u^{(1)}, u^{(2)} \ldots, u^{(N_u)}\}$, and $k \in K\{0 \ldots N\}$. However, when we try to evaluate the functional equation as before, a problem arises. We note that $g(x^{(i)}, u^{(j)}, k)$ is not necessarily a member of X; it is in fact unlikely to be so. This is illustrated geometrically in Figure 3.3. We need to know how to estimate the optimization costs at states that have not been discretized.

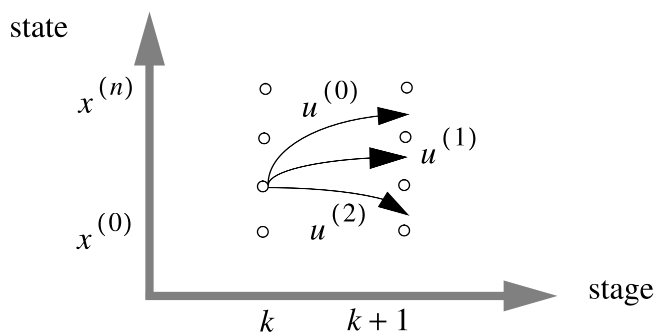


Figure 3.3 The discretization problem.
Each arc represents the path taken for one of the discretized input values. Arcs begin at discretized states, but do not in general end at discretized states.

The usual approach to this problem is to use an interpolation function to generate an appropriate estimate $\hat{I}(g(x, u, k), k+1)$ to be used in the functional equation. Linear interpolation is most common. A secondary problem arises in performing such an interpolation, namely how to treat the boundary conditions when performing the interpolation. Consider stage $N-1$ of an N-stage continuous process that is to end at state x_D. We assume that $x_D \in X$. As in the purely discrete case, we assign $I(x_D, N) = 0$. When we apply the interpolation procedure, however, we will also need to evaluate $I(x^{(i)}, N)$ for some $x^{(i)} \neq x_D$ in order to determine $\hat{I}(g(x, u, N-1), N)$. Figure 3.4 shows this *edge condition problem*. As the figure shows, it is difficult to apply any interpolation procedure to calculate the costs at the endpoints of the arcs because we do not even know the costs at the neighbouring discretized nodes.

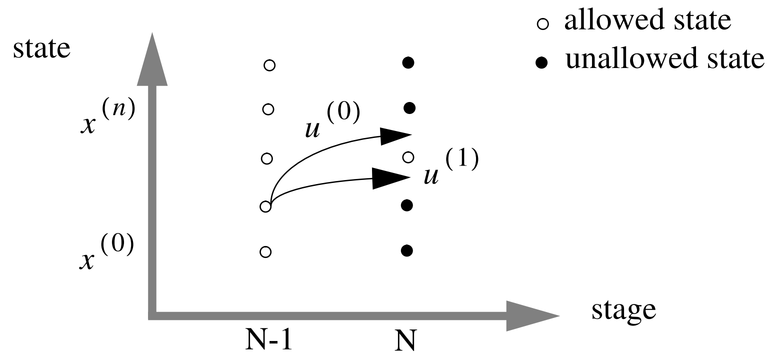


Figure 3.4 The edge-condition problem.
Interpolation procedures cannot be used to calculate the optimization costs at the arc endpoints because of the neighbouring unallowed states.

One solution is to round $g(x, u, k)$ to the closest state in such situations. This is sometimes a reasonable choice, but requires some extra complexity to determine when such rounding is needed [66]. A second solution is to assign $I(x, k)$ a large value for unallowed or unreachable states [60]. This allows for a uniform treatment of all states at the possible expense of the accuracy of the solution. We will propose an alternative solution to this problem.

In the following section we present the fundamentals of our AGDP technique. The method uses an approximating graph and is iterative in nature. In section 3.3 we describe how FDP can be performed using our method and we show it to be the exact dual of RDP. In section 3.4 the method is evaluated by applying it to problems with known analytic solutions, including a bang-bang controller and a linear quadratic regulator. Optimal controllers for two non-linear fourth order problems are also generated: a truck-parking problem and a pole-and-cart problem. Section 3.5 presents some possible extensions to our dynamic programming method. Section 3.6 provides concluding remarks on the AGDP technique and comments on its suitability as a control technique for animation.

3.2 Approximating-Graph Dynamic Programming

When synthesizing controllers for time-invariant, one-point boundary value problems, it is not necessary to introduce a stage variable into the problem formulation. We seek a representation of the control solution that is only a function of the state, thus having the form $u = F(x)$. The discrete optimization example of Figure 3.1 could be considered as being stageless if the nodes in the graph were all unique states of the system. The first step of our method is to capture the possible state transitions of a continuous system and their costs in an approximating graph. The second step will be to apply the dynamic programming functional equation iteratively to the approximating graph. Because the approximating graph is likely to contain cycles, the second step requires some subtleties that are not needed for RDP or FDP on a directed, acyclic graph.

3.2.1 Creating the Approximating Graph

The nodes in our approximating graph are created in the same way as is traditionally done for staged dynamic programming problems. The multidimensional domain of the controller, S, is discretized in each dimension to obtain a set of quantized states X. The destination state, x_D, is defined to lie exactly on one or more of the quantized states. In the examples we shall present, S is always an axis-aligned rectangular hyper-volume. We use a uniform grid to perform discretization of states within S. The discretized states that will form the nodes of the approximating graph lie at the corners of the discretization grid and not at the centre as in some other schemes [45]. Figure 3.5 shows the nodes created for a second-order system.

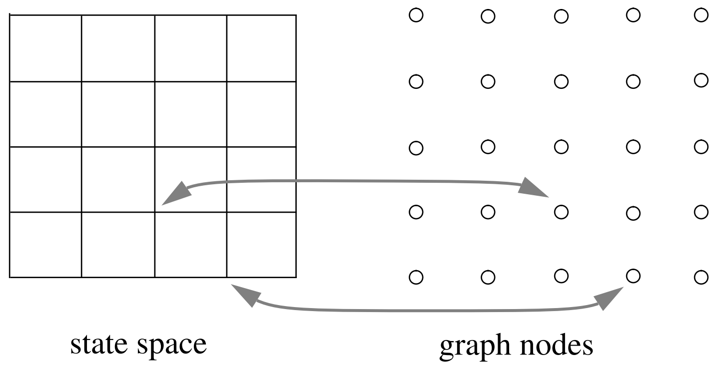
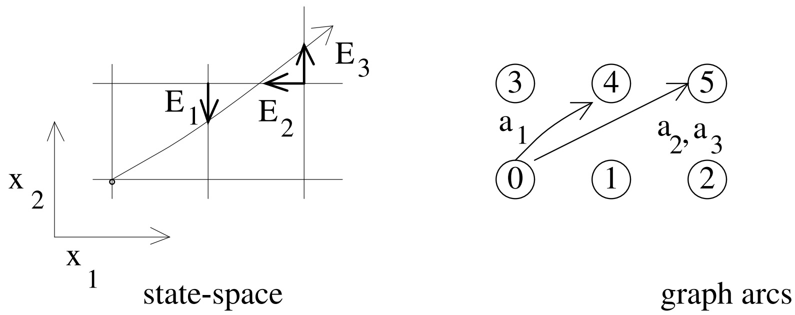


Figure 3.5 Node generation for the approximating graph.
Nodes are generated at the corners of a regular discretizing grid.

The total number of nodes created is $\prod_{i=1}^{d}(n_i+1)$, where d is the order of the system, and n_i is the number of subdivisions chosen for dimension i. The final solution will consist of estimates of the optimal control value and optimization function value at each node.

The arcs for the approximating graph require more effort to create. At least one arc is created for every element of $U \times X$. The procedure for creating an arc is as follows. The system to be controlled is placed in the state corresponding to the node and a constant input vector $u \in U$ is applied. The motion of the system is then simulated until the system state crosses the discretization grid. This marks the end-point of an arc. The ending node of the arc is chosen to be the node closest to the arc endpoint. Associated with the arc is an error vector: $E = x_j^e - x_j^{(c)}$, where x_j^e is the ending state of the arc and $x_j^{(c)}$ is the closest quantized state. It is not an absolute requirement that the arc terminate on the edge of a discretization cell; in practice, it conveniently keeps E small. In our notation referring to states, we shall use subscripts to denote associated arcs and superscripts to indicate the start or end point of an arc. Brackets around the superscript indicate the state is also a node in the corresponding approximating graph.

An arc can also be continued until a second or third intersection with the grid, thus resulting in two or three arcs being created in the graph for each node and input. As we shall see later, the longer arcs allow for a faster propagation of the solution, akin to a multiresolution implementation. Figure 3.6 illustrates the quantities involved in calculating arcs for a second order system.



arcs

arc	start	end	cost	error
a_1	0	4	L_1	E_1
a_2	0	5	L_2	E_2
a_3	0	5	L_3	E_3

Figure 3.6 Arc generation for the approximating graph.
For each arc, the closest node to the endpoint is chosen as an ending node, along with an error vector.

The generation of the arcs for the approximating graph requires $O(|U \times X|)$ short simulation sequences. This is an expensive but unavoidable process. However, because we do not make use of a stage variable, the problem is of one dimension lower than that solved using a stage variable. The arc trajectories need only be determined once for each state. A staged solution requires determining possible trajectories for each stage and state. State-increment dynamic programming (SIDP) [58] uses a similar method of calculating arcs. As in our method, the arcs in SIDP end upon crossing a grid boundary. Because SIDP still uses stages, however, the time-wise length of an arc is not allowed to exceed Δt, the time-step for a stage.

Some parallels can be drawn between the way arcs are generated in AGDP and the work of Hsu presented in [45]. He also defines an approximating graph to represent a dynamical system. AGDP differs in two fundamental respects, however. Hsu uses the arc error only as a *discrimination function* to be taken into consideration when two paths have otherwise equal costs. A second difference is that it is necessary to quantize the arc costs in Hsu's method, introducing further error in the final solution. We shall now describe how we make use of the approximating graph.

3.2.2 Dynamic Programming on the Approximating Graph

Equation 3.1 needs to be rewritten to obtain a functional equation that can be applied to the approximating graph. All references to the stage variable can be dropped. As well, let us assume that we can obtain an estimate of the gradient of the total control cost, $I(x)$, in state space. The new functional equation can be written as:

$$I(x^{(i)}) = min_{j \in A_i} \{L_j + \hat{I}(x_j^e)\} \tag{3.6}$$

where $x^{(i)}$ is the quantized state i, L_j is the cost of taking arc j, $\hat{I}(x)$ is an estimate of $I(x)$, x_j^e is the exact ending state of arc j, and A_i is the set of all arcs starting at node i.

We can calculate $\hat{I}(x_j^e)$ as follows:

$$\hat{I}(x_j^e) = I(x_j^c) + \nabla I \cdot E_j \tag{3.7}$$

where x_j^c is the closest quantized state, ∇I is the gradient of I evaluated at x_j^c, and E_j is the error associated with arc j.

Our final functional equation is

$$I(x^{(i)}) = min_{j \in A_i} \{L_j + I(x_j^{(c)}) + \nabla I \cdot E_j\}. \tag{3.8}$$

The gradient ∇I at state $x^{(n)}$ can be calculated using finite differences. The following equation defines the calculation of ∇I at node $x^{(2)}$ for the second order system shown in Figure 3.7.

$$\nabla I = \left(\frac{I(x^{(1)}) - I(x^{(2)})}{\Delta x_1}, \frac{I(x^{(3)}) - I(x^{(2)})}{\Delta x_2}\right) \tag{3.9}$$

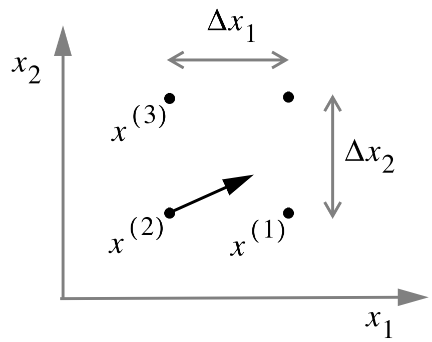


Figure 3.7 Evaluating the gradient of I using finite differences.

In some cases a value of $I(x^{(n)})$ as required by equation 3.9 may be unknown, either because it has not been evaluated yet or because of an edge condition. In these cases, we can safely replace equation 3.8 by

$$I(x^{(i)}) = min_{j \in A_i}\{L_j + I(x_j^{(c)}) + \Psi|E_j|\}, \tag{3.10}$$

where Ψ is a scalar satisfying $\Psi \geq |\nabla I_x|$, $\forall x \in S$. Because $\nabla I \cdot E_j \leq |\nabla I||E_j|$, we know that $\nabla I \cdot E_j \leq \Psi|E_j|$ and hence that we will not be corrupting our solution by using an estimate of $\nabla I \cdot E_j$ that is too small. Any underestimation of the cost can never later be removed or adjusted for, while we shall see that overestimates are corrected for by making the solution iterative. The use of ∇I can provide an underestimate of the cost when the true optimal cost function, I_o, has a concave shape. This is discussed further in § 3.4.

A more common approach in dynamic programming is to perform linear interpolation in state space, and to use a fixed value I_{max} when no estimate of $I(x^{(n)})$ exists yet [60]. The resulting estimate of $\hat{I}(x_j^e)$ will be greater than the upper bound obtained using Ψ. Figure 3.8 illustrates the difference for a one-dimensional case. In comparing the two estimates, it can be seen that the estimate obtained using Ψ provides a better upper bound because $\Psi \ll I_{max}/\Delta x_1$. It is also difficult to provide an estimate of I_{max} without first knowing the solution.

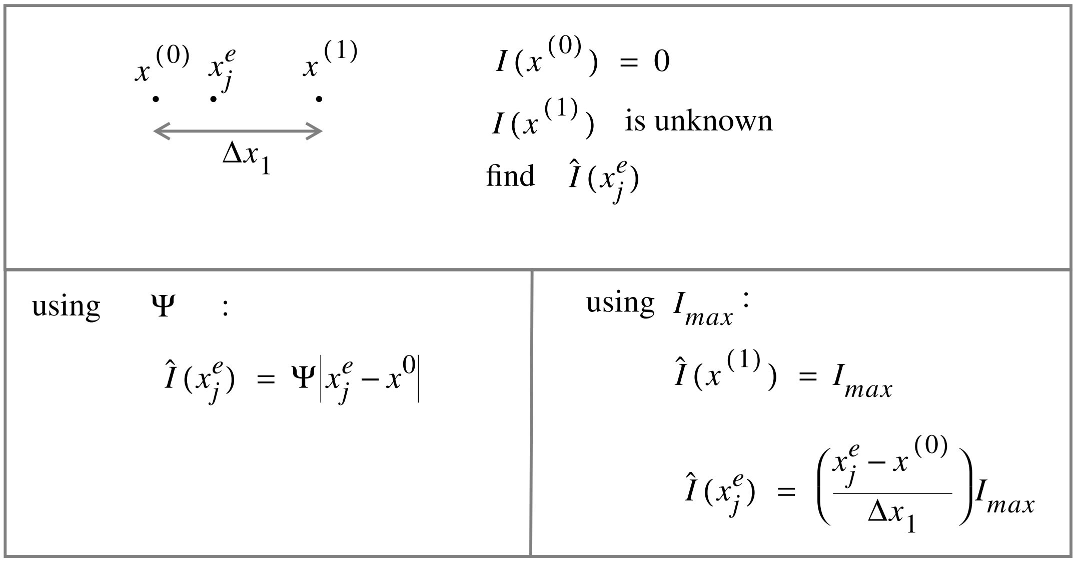


Figure 3.8 Interpolation techniques when using unevaluated nodes.
A better upper bound can be obtained by using the estimate of Ψ.

In our present version of the AGDP algorithm, the user must specify the value of Ψ such that it satisfies $\Psi \geq |\nabla I_x|$, $\forall x \in S$. While this is not too difficult given appropriate knowledge of the system to be controlled and the optimization function, it is an impediment to animators being able to use the AGDP method. An estimate of Ψ can be automatically calculated using $\Psi = max_{c \in C}|\nabla I_c|$, where C is the set of all arcs in the approximating graph, and ∇I_c is the gradient of the cost function as obtained from the arc. Unfortunately, the estimate obtained this way will not always be conservative.

We now have a suitable functional equation to apply to the approximating graph. It remains to determine in what order to apply it to the nodes in the graph. In the traditional staged approach, this is not an issue because the stage variable determines the order of evaluation. All states at stage $N-1$ would be evaluated first, followed by all states at stage $N-2$, and so forth. The order of evaluation within a stage does not matter because each stage is only dependent on the values in the next stage. The approximating graph for a dynamic system will not in general be acyclic, however, and thus there exists no single order of node evaluation that directly leads to a solution. Figure 3.9 shows how a typical cycle can arise[1].

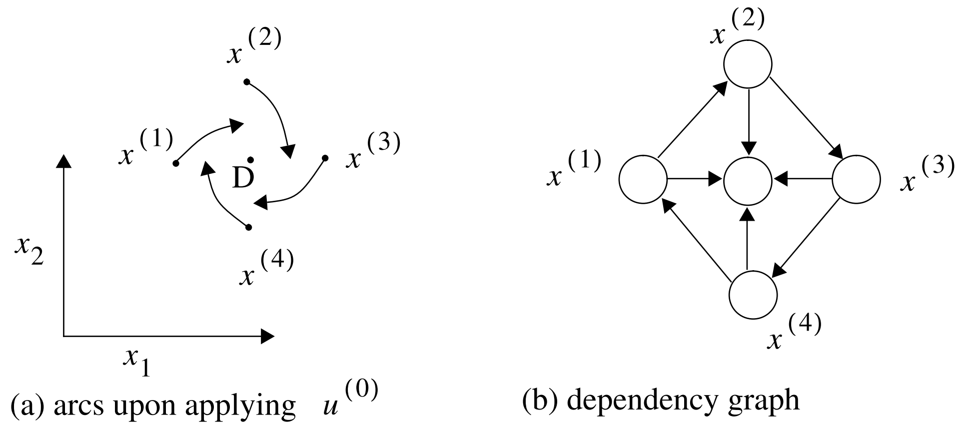


Figure 3.9 Cyclic dependencies in state space.
There is no order in which the functional equation can be applied to the nodes such that they only need to be evaluated only once.

We choose the order of node evaluation to reflect the propagation of changes. Consider node $x^{(a)}$ that has just obtained a new (lower) value of the optimization function, $I(x^{(a)})$ through the application of the functional equation. This will possibly cause a change in value in all nodes that are dependent on node a. Let $\Omega(x^{(a)})$ denote the set of dependent arcs for node a. The starting nodes of this set of arcs must now also be reevaluated using the functional equation.

The reevaluation of a node because of a dependency is simpler than a full application of the functional equation. Because it is known which arc is responsible for the potential change, it is only necessary to reevaluate one arc and compare it to the current minimum optimization value. We shall call this the *propagation functional equation* as applied to node $x^{(a)}$:

$$\forall j \in \Omega(x^{(a)}) : I(\gamma(j)) = min\{I(\gamma(j)), L_j + I(x_j^{(c)}) + \nabla I \cdot E_j\} \tag{3.11}$$

Here j represents a dependent arc and $\gamma(j)$ represents the starting node of arc j.

While the original functional equation evaluates all arcs starting at a given node, the propagational functional equation evaluates all arcs terminating in a given node. By propagating

1. Any system capable of performing a periodic motion will have cycles in its approximating graph

all changes according to equation 3.11 it is ensured that the original functional equation is always satisfied.

A simple scheme wherein the original functional equation is iteratively applied to all nodes only allows a change to propagate one arc hop per iteration. The propagation functional equation propagates the change through many nodes at each iteration, although multiple iterations are still required because of cyclic dependencies. For this reason, equation 3.11 is used to perform the dynamic programming on the approximating graph, although equation 3.8 still embodies the principle that gets satisfied.

In applying equation 3.11 to a node $x^{(a)}$, several nodes dependent on it will change in value. All nodes that have changed in value will also further require the reapplication of equation 3.11. Nodes are processed in order of increasing $I(x)$ by keeping all nodes requiring evaluation in a sorted list. This is the most efficient order of evaluation for the following reason. Given any pair of nodes satisfying $I(x^{(a)}) < I(x^{(b)})$, a change in $I(x^{(a)})$ could eventually cause a change in $I(x^{(b)})$. It is thus best to evaluate all nodes in order of increasing $I(x)$.

Using the propagation functional equation does not resolve cyclic dependencies. Accordingly, a restriction is placed on node evaluation such that each node is only evaluated once on every iteration. Iteration is stopped when

$$max_{x \in X} \{ I_{old}(x) - I_{new}(x) \} \le \varepsilon.$$

Alternatively, if it is known that there is no use for the final optimal costs, namely $I(x)$, iteration can be stopped after there are no more changes in the control values (i.e. the set of arcs chosen as optimal remains constant). This is a useful heuristic because the values of $I(x)$ will continue to change for many iterations after the control decisions have stopped changing. It is only a heuristic, however, because some closely contested decisions could possibly result in a change of optimal control value as $I(x)$ converges.

3.2.3 Using the Solution

In order to use the solution as a controller, we shall make use of the grid positions of the graph nodes in state space. The grid used to define the positions of the graph nodes forms a set of n-dimensional hypercubes in the state space of the system, where n is the order of the system. The graph nodes, at which optimal control values are known, lie at the vertices of the hypercubes. Given the current state x, the corresponding optimal control value is determined through n-linear interpolation of the control values found at the vertices of the hypercube. This is given by:

$$w_c = \prod_{i=1}^{n}\left(1 - \frac{|x_i - C_i|}{\Delta x_i}\right) \tag{3.12}$$

$$u_x = \frac{\sum w_c u_c}{\sum w_c} \tag{3.13}$$

where w_c is the weight assigned to a vertex ($0 \le w_c \le 1$ and $1 \le c \le 2^n$), n is the order of the system, C_i is the coordinate of vertex C in dimension i, Δx_i is the subdivision size along dimension i, and u_c is the optimal control value found at the given vertex. In the normal case, we will have $\sum w_c = 1$. If u_c does not exist (i.e., has not been assigned a value), indicating that the state is uncontrollable, we assign $w_c = 0$ and thus we require the sum found in the denominator of equation 3.13 to renormalize the weights.

There are several reasons for using n-linear interpolation instead of higher-order schemes. The first is that higher order interpolations are not clearly defined for many of the conditions in which they must be used. There is no clear way of dealing with states located in the boundary region between the controllable and uncontrollable regions of the control table. Uncontrollable states are those for which there is no path to the destination states. Second, there is no guarantee that a higher order interpolation will produce a better approximation. For example, a higher order interpolation can introduce a 'ringing' effect if applied to the solution of a bang-bang control problem. The optimal solutions to some biomechanical problems have been shown to be of a bang-bang nature [81]. Third, the use of higher order interpolation can cause the constraints on the range of allowable control inputs to be violated. This is not the case with n-linear interpolation.

The controller samples the current state and issues the corresponding control value at a fixed sampling rate f_c. To ensure continuity of control values, the sampling time step should be smaller than the minimum time for the state to travel the length of a grid division. We require

$$\frac{1}{f_c} \le min_i \left(\frac{\Delta x_i}{max_S |dx_i / dt|} \right),$$

where i is used to enumerate the dimensions of the system. This definition can be used to automatically select f_c.

During the use of a controller, we can calculate a value for the optimization cost function. We shall use I_c to denote the actual cost obtained when using the control table. Note that $I_c(x)$ is not necessarily the same as $I(x)$ because the interpolation as specified by equation 3.13 results in non-discrete control values being used. $I_c(x)$ will prove useful in evaluating the optimality of the control tables produced by AGDP, as we shall see in § 3.4.

3.3 Forward Dynamic Programming

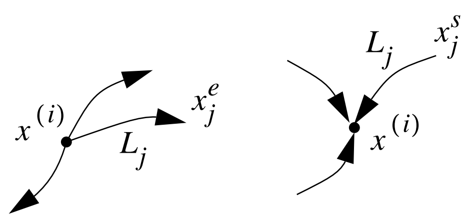
Forward dynamic programming (FDP) [11] [60] solves the dual problem to the more common reverse dynamic programming problem (RDP). Thus far, we have presented how to use AGDP to solve the problem of reaching a fixed a set of desired destination states in an optimal way from any initial state within the controller domain. FDP solves the problem of how to reach any state in a specified domain in an optimal way from a fixed starting state. RDP obtains a solution by working backwards from the end point, while FDP works forwards from the starting point. As with RDP, the problem becomes easier to solve when more constraints are placed on it, because any constraints serve to restrict the size of the approximating graph.

3.3.1 The Functional Equation

Larson presents an FDP algorithm in [60] that makes use of stages. As with the AGDP solution to RDP, we choose to deal only with time invariant systems, thus obviating the need to use a staged solution. Our description of FDP begins with its functional equation. An interpretation of the functional equation of RDP (equation 3.4) is that for any given node, one must always choose the exiting arc having the minimal total cost. This always represents the path of minimal cost to get to the set of nodes in x_D. In the FDP dual we must always choose the entering arc having the minimum total cost because it represents the best way to get to that state. The FDP functional equation has the form

$$I(x^{(i)}) = min_{j \in B_i} \{ L_j + \hat{I}(x_j^s) \} , \tag{3.14}$$

where B_i is the set of all arcs ending exactly at node i, $\hat{I}(x)$ is an estimate of $I(x)$, x_j^s is the starting state of arc j, and L_j is the cost of arc j. Figure 3.10 shows a geometrical interpretation of the functional equations for RDP and FDP.



RDP: $j \in A_i$ FDP: $j \in B_i$

Figure 3.10 Functional equation application to node x(i). Reverse dynamic programming finds the minimal cost to exit a node, while forward dynamic programming finds the minimal cost to reach a node.

The functional equations for the RDP and FDP problems are very similar (equations 3.6 and 3.14). x_j^e is replaced by x_j^s, and A_i is replaced by B_i. We will now examine if we can use an identical functional equation for both problems. An arc for the RDP approximating graph is obtained by beginning at state $x^{(i)}$ and using a numerical simulation of the dynamical system with a constant discretized control value being applied. This numerical simulation uses a time step

Δt. In the case of the FDP problem, we wish to generate arcs terminating exactly in a given state. This can be accomplished by using a time step of $-\Delta t$ and tracing the state trajectory backwards in time. The end point of this backwards trace is x_j^s. We thus find that changing the sign of the time step determines x_j^s instead of x_j^e, and produces the set of arcs B_i instead of A_i. We have shown earlier that making this substitution in the RDP functional equation produces the FDP functional equation, thus we can conclude that using a negative time step transforms an RDP problem into an FDP problem.

In order to be able to make use of this dual property, it is necessary that the dynamical system be time-reversible. This encompasses a large class of systems, but not all dynamical systems. An inelastic collision is an example of a process that is not time-reversible. It is impossible to recover the initial velocities before such an inelastic collision took place. For systems that are time-reversible, however, the FDP and RDP problems are duals of each other, requiring only a change in the sign of the time step when calculating the arcs in the approximating graph for their AGDP solution.

3.3.2 Backtracking

The solution produced by FDP is not a controller. It produces the control necessary to get from the fixed initial state to any desired final state within the given domain in an open-loop fashion. In order to obtain the open-loop control input required to reach a desired state x', it is necessary to invoke a back-tracking procedure [60]. The back-tracking procedure begins at state x' and finds the immediately preceeding optimal path (and control value) taken to arrive at state x'. The process proceeds backwards in time in this manner until the initial state is reached. The control function is stored as the back-tracking proceeds.

The FDP-RDP dual extends nicely in that the backtracking procedure for FDP amounts to using the control values of the solution as if it were a controller for an RDP problem. The plant in this case is the dynamical system operating backwards in time. This means that the control interpolation procedures described by equation 3.12 can be used. Previous work mentioning the

FDP problem [60] does not present specific details on how the back-tracking should be performed.

3.4 An Evaluation of the Method

Dynamic programming has been used in the past predominantly to calculate optimal trajectories [7] [66] [81]. The approximating graph technique is directed towards calculating optimal controllers and as such needs to be evaluated differently. We shall first examine several possible error measures that can be used for comparing the calculated controllers with true optimal control solutions. We shall then describe and evaluate the generation of optimal control tables for four different problems. It is unfortunately difficult to perform direct comparisons of the accuracy and computational efficiency of AGDP with that of state-increment dynamic programming as presented by Larson [60], or the technique presented by Hsu [45]. The results presented by Larson are for two-dimensional systems only and are based on knowing apriori the sequence in which to evaluate nodes. No error measures with respect to the true optimal solution are given. Similarly, there are no error results given for Hsu's work.

3.4.1 Error Measures for Controllers

The controller obtained using AGDP is an approximation to the true optimal controller. There are several possible ways of measuring the quality of this approximation. The first is to compare the control values provided by the controller with the known optimal control values. This could be done everywhere in the domain of the controller. This is not always a useful error measure. In the case of a bang-bang controller, a slight error in the position of the switching curve will result in a large error as measured in terms of control values. A better measure in this case would be one that measures the total error in the position of the switching curve.

A more useful criterion is the optimization function itself. There are two ways this can be used. The first is to compare $I(x)$ as calculated at the nodes with their true values for the same states, denoted by $I_{opt}(x)$. However, this ignores the fact that in using the controller, the linear interpolation scheme of equation 3.12 is used to determine the control. As a result, a better evaluation criterion is to use $I_c(x)$, which represents the optimization function as evaluated during the actual use of the controller. In our case, $I_c(x)$ is evaluated by using a simulation of the system and controller, integrating the cost of the optimization function until the destination state is reached. A relative error measure using $I_c(x)$ is given by:

$$E_1(x) = \frac{I_c(x) - I_{opt}(x)}{I_{opt}(x)} \tag{3.15}$$

We shall also use

$$E_2(x) = \frac{I(x) - I_{opt}(x)}{I_{opt}(x)} \tag{3.16}$$

as a secondary error measure.

3.4.2 Minimal-Time Control of a $1/s^2$ System

As our first example, we choose a system that is simple and has a well-known analytic solution. The state equations for a $1/s^2$ system can be written as:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = u.$$

This corresponds to a physical system consisting of an object whose acceleration we can control. The time taken to reach the destination state, $x_D = \begin{bmatrix} 0 & 0 \end{bmatrix}$, is to be minimized. This means we wish to bring the mass to rest in a desired position as fast as possible. An additional constraint bounds the input of the system, namely $-1 \le u \le 1$. We consider the solution over the domain $-1 \le x_1 \le 1$ and $-1 \le x_2 \le 1$. This problem is treated in [51] among others. The known optimal solution is shown in Figure 3.11. The corresponding minimum time to reach the origin can be given as:

$$-0.5x_2|x_2| - x_1 > 0 \qquad I_{opt} = -x_2 + 2\sqrt{0.5x_2^2 - x_1}$$

$$-0.5x_2|x_2| - x_1 \le 0 \qquad I_{opt} = -x_2 + \sqrt{0.5x_1^2 + x_1}$$

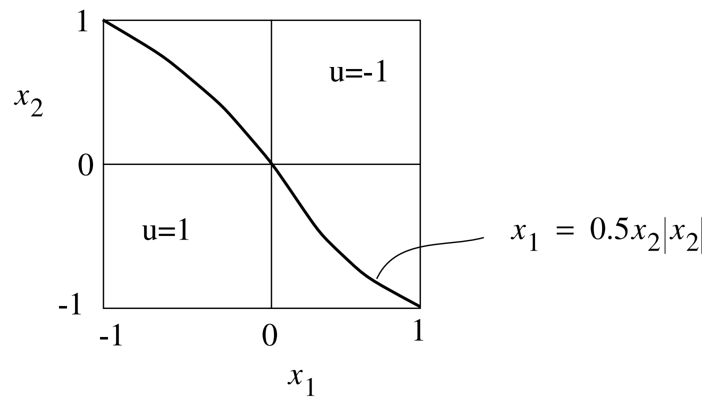


Figure 3.11 Minimal-time control of a $1/s^2$ plant.
The switching curve defines two regions, one where a maximum input of +1 is applied, another where a minimal input of -1 is applied.

Larson [59] solves this minimum-time control problem with dynamic programming by fixing the evaluation order of the nodes to be counter-clockwise around the destination state. This evaluation order avoids cyclic dependencies for this specific problem, thus each node only needs to be evaluated once. Such problem-specific knowledge cannot be expected in the general case, however. No method for evaluating the quality of the final solution obtained is given.

In applying AGDP to the problem, we shall first present a nominal solution that discretizes state space using an 13x13 grid We shall then show how the solution is affected by using different grid densities. All of the solutions have the input range discretized into 7 samples, generate two arcs for each element of $U \times X$, corresponding to the first and second intersections with the discretization grid. We use $\Psi = 4$ as an upper bound on the gradient of the optimization function. This is a choice that reflects some knowledge of the problem. The resulting approximating graph has 781 arcs and requires 18 iterations to solve. The terminating condition was chosen to be an iteration with no changes in $I(x)$ because we will be comparing these values to the true optimal values. In this case, we effectively consider $\varepsilon = 0$.

Figure 3.12 shows a table of the optimal control values at all points on the grid. This can be compared with the true optimal solution as shown in Figure 3.11. The resulting control is mostly of a bang-bang nature. The uncontrollable states are marked with an ‘x’, indicating there is no sequence of inputs to get from that particular state to x_D without passing outside of the controller domain or violating one of the imposed constraints. Some control values near the corners of the grid do not have the expected optimal control values. This is as a result of the finite size of the domain being considered; the true optimal solution in these cases passes outside of the domain S at some point in time. Figure 3.13 shows a table of the final calculated optimization values, I. These values are thus the time required to reach the origin from the corresponding states. From the figure it can be seen that $|\nabla I| < \Psi$, as desired.

x_2											
1	-0.7	-0.7	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	x	x	x
	0.3	0.3	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	x	x
	1.0	1.0	1.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	x
	1.0	1.0	1.0	1.0	-0.7	-1.0	-1.0	-1.0	-1.0	-1.0	x
	1.0	1.0	1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	x
0	1.0	1.0	1.0	1.0	1.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0
	x	1.0	1.0	1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0
	x	1.0	1.0	1.0	1.0	1.0	0.7	-1.0	-1.0	-1.0	-1.0
	x	x	1.0	1.0	1.0	1.0	1.0	0.3	-1.0	-1.0	-1.0
-1	x	x	1.0	1.0	1.0	1.0	1.0	1.0	0.0	-0.3	-0.3
	x	x	x	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.0
x_1	-1					0					1

Figure 3.12 Optimal control calculated using AGDP (13x13x7 discretization).

1.93	1.70	1.47	1.71	2.47	2.96	3.29	5.80	x	x	x
1.95	1.72	1.49	1.24	1.56	2.35	2.81	3.12	5.17	x	x
1.99	1.76	1.52	1.27	0.94	1.65	2.36	2.71	2.99	5.25	x
2.07	1.84	1.59	1.30	1.16	0.93	1.97	2.36	2.66	3.07	x
3.47	1.96	1.71	1.46	1.29	0.34	1.64	2.06	2.39	2.63	x
4.07	2.14	1.88	1.65	1.40	0.00	1.40	1.83	2.16	2.41	3.06
x	2.53	2.11	1.87	1.62	0.34	1.29	1.67	1.99	2.23	2.47
x	3.93	2.44	2.14	1.90	0.93	1.16	1.61	1.87	2.11	2.35
x	x	3.38	2.49	2.22	1.61	1.24	1.57	1.79	2.02	2.25
x	x	6.11	3.18	2.61	2.24	1.55	1.51	1.76	1.99	2.22
x	x	x	5.99	3.32	2.77	2.37	1.74	1.74	1.97	2.19

Figure 3.13 I(x) from the approximating graph.

Error measures E_1 and E_2 were applied at all grid points. The results are shown in Figure 3.14 as percentage errors. The errors are relatively large given that the actual control values seem to reasonably approximate the desired switching curve. It is interesting to observe that E_2 is occasionally negative, implying that the optimal cost calculated using the approximating graph was less than the true optimal solution. This occurs because the function $I_{opt}(x)$ is convex in some regions. In this case, the linear interpolation using ∇I results in $\hat{I} < I_{opt}$ for convex regions.

	23	20	27	2	17	17	16	x	x	x	x
	23	23	24	30	31	20	18	17	x	x	x
	21	23	27	34	50	23	22	20	17	x	x
	19	21	28	39	41	3	24	23	21	23	x
	17	20	27	38	41	13	27	28	24	20	x
E_1	16	18	24	33	34	0	34	33	27	22	19
	x	16	21	28	27	13	41	38	31	24	21
	x	20	17	22	24	3	41	41	34	26	23
	x	x	15	17	22	23	50	39	34	28	26
	x	x	x	14	17	20	31	24	30	28	27
	x	x	x	x	13	16	17	2	24	25	26

	33	33	34	5	18	23	23	100	x	x	x
	30	31	34	39	5	22	25	25	90	x	x
	26	27	30	38	48	14	29	28	26	104	x
	23	24	28	32	77	-4	35	32	30	35	x
	91	22	24	33	75	-29	44	38	34	31	x
E_2	104	20	22	30	56	0	56	45	40	35	53
	x	26	19	25	43	-30	75	53	45	39	36
	x	73	19	20	30	-4	77	63	50	43	40
	x	x	43	17	21	11	97	70	53	47	43
	x	x	125	27	16	16	4	68	57	51	48
	x	x	x	107	24	15	13	7	58	54	51

Figure 3.14 Percentage error measures for bang-bang controller using metrics E_1 and E_2.. These performance error measures show that small errors in approximating the switching curve can yield large errors in the resulting performance for minimum-time problems.

It is evident that the quality of the solution will be somehow dependent upon the quantization levels used. The optimal switching curve can only be approximated in the resulting control table, as shown in Figure 3.12. A better approximation can possibly be constructed using a finer quantization grid. In order to explore this, we present the E_1 error metric for several

	32	28	15	1	20	19	16	x	x	x	x
	31	30	31	19	31	22	21	17	x	x	x
	29	29	34	43	36	24	26	21	17	x	x
	25	27	34	48	48	3	27	26	21	23	x
	22	24	32	45	45	13	29	33	25	20	x
11x11x7	20	22	29	39	36	0	36	39	29	22	20
	x	20	25	33	29	13	45	45	32	24	22
	x	23	21	26	27	3	48	48	34	27	25
	x	x	18	21	26	24	36	43	34	29	29
	x	x	x	17	21	22	31	19	31	30	31
	x	x	x	x	16	19	20	1	15	28	30

	17	15	14	0	10	8	6	6	x	x	x
	15	14	15	14	6	11	8	7	7	x	x
	12	12	14	18	27	11	11	8	7	x	x
	11	11	14	19	28	-0	14	10	8	7	x
	9	10	12	18	26	-0	17	13	9	8	x
21x21x7	8	9	11	16	20	0	20	16	11	9	8
	x	8	9	13	17	-0	26	18	12	10	9
	x	7	8	10	14	-0	28	19	14	11	11
	x	x	7	8	11	11	27	18	14	12	12
	x	x	7	7	8	11	6	14	15	14	15
	x	x	x	6	6	8	10	0	14	15	17

	14	13	12	0	8	6	6	5	x	x	x
	11	11	12	16	9	8	6	6	5	x	x
	10	10	12	15	14	13	8	7	6	x	x
	8	9	12	14	23	-0	11	8	7	6	x
	10	9	10	13	20	-0	14	10	8	7	x
	11	8	9	11	17	0	19	12	9	8	11
41x41x7	x	7	8	9	13	-0	22	13	11	9	11
	x	6	7	8	11	-0	25	14	12	10	8
	x	x	6	7	8	11	14	15	13	10	10
	x	x	5	6	6	8	8	16	13	11	12
	x	x	x	5	5	6	7	0	12	13	14

	15	15	15	0	7	6	6	6	x	x	x
	13	13	14	18	8	8	7	6	6	x	x
	11	12	14	17	20	10	8	7	7	7	x
	10	11	13	15	22	-0	10	9	8	7	x
	9	10	11	14	19	-0	13	10	9	8	x
81x81x7	8	9	10	12	16	0	16	12	10	9	8
	x	8	9	10	13	-0	19	14	11	10	9
	x	7	8	9	10	-0	22	15	13	11	10
	x	7	7	7	8	10	20	17	14	12	11
	x	x	6	6	7	8	8	18	14	13	13
	x	x	x	6	6	6	7	0	15	15	15

Figure 3.15 Effect of grid quantization on the E1 error metric on the plant. The error decreases with a finer grid, but not indefinitely. The large state-space grids have been subsampled.

solutions calculated with varying discretizations of state space. Because we are solving a bang-bang control problem, performing a finer discretization of the input space does not have much impact on the quality of the solution.

Figure 3.15 shows how the quality of the solution improves when a finer grid is used to quantize the state-space. Although the solution does improve with a finer quantization, it does not do so indefinitely. As the grid quantization becomes finer, the optimal solution can be represented with increasing fidelity. The improvement is limited, however, because of errors that are made in approximating $I(x)$ as the solution is being calculated. These errors cause the calculated switching curve to differ from the optimal one. The errors in approximating $I(x)$ arise from the use of an estimate of the gradient to perform interpolation, as described by equation 3.7. This ignores the higher-order curvatures present in $I(x)$. A finer discretization grid decreases the errors for individual approximations, but does not necessarily improve the global approximation. A finer grid results in the use of more approximations because the effective number of stages involved in propagating the solution outwards from the destination states is larger.

3.4.3 Linear Quadratic Regulator

Another class of optimization problem with a well-known analytic solution is that of the linear quadratic regulator (LQR). Treatment of this class of problems can be found in [51] among others. We shall briefly restate the problem as follows. Given the linear system modeled by

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

minimize

$$J = \int_0^\infty [y^T Q y + u^T R u]\, dt,$$

where x, y, and u are all functions of time, Q is assumed to be positive semidefinite, and R is assumed to be positive definite.

The optimization function J implies regulation, that is $y(t) \to 0$ as $t \to \infty$. The known optimal solution is obtained by solving the algebraic Riccati equation for the symmetric matrix P:

$$A^T P + PA + Q = PBR^{-1}B^T P. \qquad (3.17)$$

The optimal control is given by the feedback matrix F, such that $u = Fx$, where $F = -R^{-1}B^T P$. The optimal cost is given by $I_{opt}(x) = x^T P x$.

In the following example, we consider the following second order problem:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$R = 0.2 \qquad Q = \begin{bmatrix} 3 & 0.5 \\ 1 & 1 \end{bmatrix}$$

The solution obtained using the algebraic Riccati equation is:

$$P = \begin{bmatrix} 4.4456 & 0.4718 \\ 0.4718 & 1.5740 \end{bmatrix} \qquad F = \begin{bmatrix} -2.3589 & -7.8701 \end{bmatrix}$$

An approximating graph is constructed using a 13x13 grid of nodes centred about the origin and covering the domain $-1 \le x_1 \le 1$, $-1 \le x_2 \le 1$. Two arcs for each element of $U \times X$ are used. An input space of $-10.2 \le u \le 10.2$ is discretized uniformly into 13 levels, and we choose $\Psi = 3$. The approximating graph for this case has 169 nodes, 3422 arcs and takes 13 iterations for $I(x)$ to become fixed. The calculated control values are shown in Figure 3.16 and the corresponding calculated values of $I(x)$ are shown in Figure 3.17.

-5.1	-5.1	-5.1	-5.1	-5.1	-6.8	-8.5	-8.5	-8.5	-8.5	-8.5	-10.2	x
-5.1	-5.1	-5.1	-5.1	-5.1	-6.8	-6.8	-6.8	-6.8	-6.8	-6.8	-10.2	x
-3.4	-3.4	-3.4	-3.4	-3.4	-3.4	-5.1	-5.1	-5.1	-5.1	-5.1	-8.5	x
-1.7	-1.7	-1.7	-1.7	-1.7	-3.4	-3.4	-5.1	-5.1	-5.1	-5.1	-6.8	x
-1.7	-1.7	-1.7	0.0	0.0	-1.7	-3.4	-3.4	-3.4	-3.4	-3.4	-3.4	x
-1.7	0.0	0.0	0.0	0.0	0.0	-1.7	-1.7	-1.7	-1.7	-1.7	-1.7	x
3.4	1.7	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.7	-1.7	-3.4
x	1.7	1.7	1.7	1.7	1.7	1.7	0.0	0.0	0.0	0.0	0.0	1.7
x	3.4	3.4	3.4	3.4	3.4	3.4	1.7	0.0	0.0	1.7	1.7	1.7
x	6.8	5.1	5.1	5.1	5.1	3.4	3.4	1.7	1.7	1.7	1.7	1.7
x	8.5	5.1	5.1	5.1	5.1	5.1	3.4	3.4	3.4	3.4	3.4	3.4
x	10.2	6.8	6.8	6.8	6.8	6.8	6.8	5.1	5.1	5.1	5.1	5.1
x	10.2	8.5	8.5	8.5	8.5	8.5	6.8	5.1	5.1	5.1	5.1	5.1

Figure 3.16 Control calculated using AGDP for LQR problem (13x13x13 discretization).

5.77	4.53	3.51	2.63	2.09	1.84	1.98	2.46	3.14	3.96	5.03	6.76	x
5.48	4.20	3.18	2.34	1.68	1.36	1.43	1.90	2.57	3.37	4.38	5.97	x
5.20	3.89	2.86	2.07	1.35	0.95	0.96	1.43	2.09	2.88	3.85	5.30	x
5.16	3.76	2.73	1.87	1.26	0.69	0.59	1.05	1.70	2.48	3.41	4.73	x
5.01	3.67	2.60	1.80	1.17	0.50	0.30	0.75	1.40	2.17	3.06	4.26	x
5.26	3.68	2.59	1.77	1.08	0.47	0.10	0.54	1.19	1.95	2.81	3.97	x
5.46	3.78	2.66	1.82	1.08	0.44	0.00	0.44	1.08	1.82	2.66	3.78	5.46
x	3.96	2.82	1.95	1.19	0.54	0.10	0.47	1.08	1.77	2.59	3.68	5.26
x	4.26	3.06	2.17	1.40	0.75	0.30	0.50	1.17	1.80	2.59	3.67	5.01
x	4.73	3.41	2.48	1.70	1.05	0.59	0.69	1.26	1.87	2.73	3.76	5.16
x	5.30	3.85	2.88	2.09	1.43	0.96	0.95	1.35	2.07	2.86	3.89	5.20
x	5.98	4.38	3.38	2.57	1.90	1.42	1.36	1.68	2.34	3.18	4.20	5.48
x	6.77	5.03	3.96	3.14	2.46	1.98	1.84	2.08	2.63	3.51	4.53	5.77

Figure 3.17 I(x) calculated by AGDP for LQR problem (13x13x13 discretization).

	3	4	4	5	4	3	3	6	10	7	5	7	x
	3	4	5	5	4	3	2	7	11	8	6	6	x
	3	4	6	8	6	4	2	8	13	10	7	6	x
	3	4	7	11	9	1	3	10	16	12	7	6	x
	4	5	8	15	18	4	6	13	19	15	8	6	x
	7	5	8	17	23	18	5	18	21	17	8	6	x
	7	5	8	17	22	20	0	20	22	17	8	5	7
E_1	x	6	8	17	21	18	5	18	23	17	8	5	7
	x	6	8	15	19	13	6	4	18	15	8	5	4
	x	6	7	12	16	10	3	1	9	11	7	4	3
	x	6	7	10	13	8	2	4	6	8	6	4	3
	x	6	6	8	11	7	2	3	4	5	5	4	3
	x	7	5	7	10	6	3	3	4	5	4	4	3

	14	17	20	19	19	20	26	33	32	25	20	24	x
	15	19	25	29	27	26	30	41	39	30	22	24	x
	15	19	27	39	38	33	38	54	49	36	24	23	x
	18	22	33	47	73	58	50	76	63	43	27	22	x
	16	22	34	60	108	104	70	113	80	50	30	21	x
	21	23	35	64	123	235	124	181	101	58	33	22	x
E_2	23	22	35	64	118	259	0	259	118	64	35	22	23
	x	22	33	58	101	181	123	235	123	64	35	23	21
	x	21	30	50	81	113	69	104	107	60	34	22	16
	x	22	27	43	63	76	50	58	73	47	33	22	18
	x	23	24	36	49	54	38	32	38	38	27	19	15
	x	24	22	30	39	41	30	25	27	29	25	19	15
	x	24	20	25	32	33	26	20	19	19	20	17	14

Figure 3.18 Percentage error measures for LQR problem.
The performance of the controller with interpolation is significantly better than the performance estimated directly from the approximating graph.

The error measures for the calculated solution are shown in Figure 3.18. The error results for this problem are particularly interesting because it is possible to represent the true optimal solution in the control table form that AGDP produces. The optimal control is given by $u = Fx$, which can be represented in the control table without making any approximations. Error measure E1 shows that when the control table produced by AGDP is used, it produces results within 20% of optimal over 95% of the controller domain. However, error measure E2 shows that the errors in the cost estimates produced by the AGDP method are in general much larger, especially near the origin. This can be attributed to the discretized input space U. When the control table is actually in use, the linear interpolation between control table entries yields non-discrete control values. The AGDP cost estimates are produced using only discrete control values, however. This produces particularly bad cost estimates near the origin, where many small optimal control values are required. These are not well represented in a regularly sampled discretization of the input space.

The analytic optimal solution defines a plane over the state-space of the object. Given the approximating-graph control solution, we can calculate a feedback matrix from the results by using linear regression to find the equation of the best-fitting plane passing through the origin. In applying this procedure, the nodes influenced by edge effects were ignored. The resulting feedback matrix is given by $F = \begin{bmatrix} -2.147 & -7.406 \end{bmatrix}$. This is reasonably close to the true optimal feedback matrix of $F = \begin{bmatrix} -2.3589 & -7.8701 \end{bmatrix}$ as calculated earlier. The local approximations and the various discretizations necessary for the AGDP method are responsible for the errors in the true optimal solution.

The AGDP solution for the LQR problem can benefit from a finer discretization of both state-space and the input space. Figure 3.19 shows the result of progressively finer discretizations of the state space, while Figure 3.20 shows the result of making the input discretization finer. For both cases, the finer discretizations generally results in better solutions. As with the bang-bang controller, however, there are limitations to the increase in accuracy achieved. The optimal cost for the LQR problem as calculated by analytic means is not a linear function of the state. This means that using ∇I to produce estimates of $I(x)$ will yield errors that cannot be accounted for by increasing the discretization of state space or the inputs. The process of using a finer discretization is equivalent to extending the controller domain while maintaining the same discretization intervals. Any errors already existing in the control solution are further magnified when extending the controller domain.

	x	x	x	170	28	21	34	90	x	x	x	x	x
	x	x	x	42	25	19	28	68	136	x	x	x	x
	x	x	147	37	23	15	25	53	86	138	x	x	x
	x	x	54	31	18	13	22	42	56	75	x	x	x
	x	x	45	23	12	9	14	32	34	42	62	x	x
	x	x	38	17	6	2	8	19	17	24	42	x	x
3x3x13	x	x	36	16	6	4	0	4	6	16	36	x	x
	x	x	42	24	17	19	8	2	6	17	38	x	x
	x	x	62	42	34	32	14	9	12	23	45	x	x
	x	x	x	75	56	42	22	13	18	31	54	x	x
	x	x	x	138	86	53	25	15	23	37	147	x	x
	x	x	x	x	136	68	28	19	25	42	x	x	x
	x	x	x	x	x	90	34	21	28	170	x	x	x

	2	3	3	5	6	2	3	7	9	7	8	x	x
	3	3	4	4	3	2	3	6	10	8	7	x	x
	3	4	5	5	4	1	2	5	13	9	7	23	x
	3	4	5	7	8	2	2	4	17	11	8	13	x
	3	4	5	10	18	6	2	4	21	12	8	7	x
	3	4	6	11	25	1	2	5	24	13	9	5	4
7x7x13	3	4	7	12	23	3	0	3	23	12	7	4	3
	4	5	9	13	24	5	2	1	25	11	6	4	3
	x	7	8	12	21	4	2	6	18	10	5	4	3
	x	13	8	11	17	4	2	2	8	7	5	4	3
	x	23	7	9	13	5	2	1	4	5	5	4	3
	x	x	7	8	10	6	3	2	3	4	4	3	3
	x	x	8	7	9	7	3	2	6	5	3	3	2

	2	3	4	5	3	3	4	7	4	3	3	11	x
	2	3	4	6	5	2	4	8	6	4	3	7	x
	2	3	4	6	7	3	5	11	8	4	3	5	x
	2	3	4	6	5	3	6	15	10	5	4	4	x
	2	3	4	6	7	7	3	21	12	6	4	3	x
	3	3	4	7	11	27	1	28	14	6	4	3	5
11x11x13	4	3	4	6	12	29	0	29	12	6	4	3	4
	5	3	4	6	14	28	1	27	11	7	4	3	3
	x	3	4	6	12	21	3	7	7	6	4	3	2
	x	4	4	5	10	15	6	3	5	6	4	3	2
	x	5	3	4	8	11	5	3	7	6	4	3	2
	x	7	3	4	6	8	4	2	5	6	4	3	2
	x	11	3	3	4	7	4	3	3	5	4	3	2

	2	3	4	5	4	2	3	7	8	5	4	3	x
	2	3	4	4	5	3	3	7	10	6	4	4	x
	3	3	4	6	5	4	2	8	12	7	5	4	x
	3	3	5	8	9	2	3	9	14	8	5	4	x
	3	3	5	10	16	4	5	12	18	9	6	4	x
	5	4	6	11	23	16	5	18	22	11	6	4	x
15x15x13	5	4	6	11	23	22	0	22	23	11	6	4	5
	x	4	6	11	22	18	5	16	23	11	6	4	5
	x	4	6	9	18	12	5	4	16	10	5	3	3
	x	4	5	8	14	9	3	2	9	8	5	3	3
	x	4	5	7	12	8	2	4	5	6	4	3	3
	x	4	4	6	10	7	3	3	5	4	4	3	2
	x	3	4	5	8	7	3	2	4	5	4	3	2

Figure 3.19 Effect of state-space discretization on the E1 error measure for the LQR problem.

13x13x7

6	12	13	11	6	5	6	5	17	13	8	7	x
5	7	16	17	10	5	7	5	20	15	10	8	x
5	7	10	13	11	8	8	7	24	18	11	8	x
6	8	13	21	25	6	12	9	27	22	13	8	x
6	9	16	29	37	5	28	14	15	26	15	8	x
7	9	17	32	15	7	48	15	11	30	16	9	x
7	9	17	32	9	7	0	7	9	32	17	9	7
x	9	16	30	11	15	48	7	15	32	17	9	7
x	8	15	26	15	14	28	5	37	29	16	9	6
x	8	13	22	27	9	12	6	25	21	13	8	6
x	8	11	18	24	7	8	8	11	13	10	7	5
x	8	10	15	20	5	7	5	10	17	16	7	5
x	7	8	13	17	5	6	5	6	11	13	12	6

13x13x11

4	5	7	8	8	5	3	5	12	8	5	7	x
4	6	7	9	9	7	2	5	14	10	6	6	x
5	6	8	11	12	6	4	6	16	12	7	5	x
5	6	9	14	19	9	7	8	19	14	8	5	x
5	6	10	19	26	2	14	12	21	17	9	6	x
6	6	10	21	26	12	11	14	22	20	10	6	x
6	6	10	21	21	15	0	15	21	21	10	6	6
x	6	10	20	22	14	11	12	26	21	10	6	6
x	6	9	17	21	12	14	2	26	19	10	6	5
x	5	8	14	19	8	7	9	19	14	9	6	5
x	5	7	12	16	6	4	6	12	11	8	6	5
x	6	6	10	14	5	2	7	9	9	7	6	4
x	7	5	8	12	5	3	5	8	8	7	5	4

13x13x15

6	7	9	10	7	2	4	5	10	9	8	8	x
6	8	10	12	11	3	3	5	12	11	9	8	x
6	9	12	15	14	6	2	6	14	12	10	8	x
7	9	14	19	15	3	2	8	17	14	11	8	x
8	10	15	20	19	5	2	12	18	17	13	9	x
9	10	15	20	17	15	2	19	16	18	14	9	x
9	10	14	19	13	22	0	22	13	19	14	10	9
x	9	14	18	16	19	2	15	17	20	15	10	9
x	9	13	17	18	12	2	5	19	20	15	10	8
x	8	11	14	17	8	2	3	15	19	14	9	7
x	8	10	12	14	6	2	6	14	15	12	9	6
x	8	9	11	12	5	3	3	11	12	10	8	6
x	8	8	9	10	5	4	2	7	10	9	7	6

13x13x19

2	2	3	7	7	3	5	4	3	3	2	5	x
2	2	3	5	9	4	5	5	4	3	3	4	x
2	2	3	3	6	5	3	7	5	3	3	3	x
2	2	2	3	4	6	4	10	6	3	3	3	x
3	2	2	4	7	7	4	17	8	3	3	2	x
4	2	2	4	9	24	2	28	10	4	2	2	x
4	2	2	4	10	31	0	31	10	4	2	2	5
x	2	2	4	10	28	2	24	9	4	2	2	5
x	2	3	3	8	17	4	7	7	4	2	2	3
x	3	3	3	6	10	4	6	4	3	2	2	2
x	3	3	3	5	7	3	5	6	3	3	2	2
x	4	3	3	4	5	5	4	9	5	3	2	2
x	5	2	3	3	4	5	3	7	7	3	2	2

Figure 3.20 Effect of input discretization on the E1 error measure for the LQR problem. Increasing the number of input discretization levels greatly reduces the error for this LQR problem.

3.4.4 Truck Parking

The problem of backing up a truck-and-trailer configuration has traditionally been treated as a motion-planning problem with nonholonomic constraints [61]. Our representation of the problem is shown in Figure 3.21. The goal is to back the truck to the loading dock by steering the cab appropriately. The speed of the rear-wheels of the trailer is assumed to be constant. The speed of the motion is irrelevant to our formulation of the problem, assuming the steering and speed can be controlled independently.

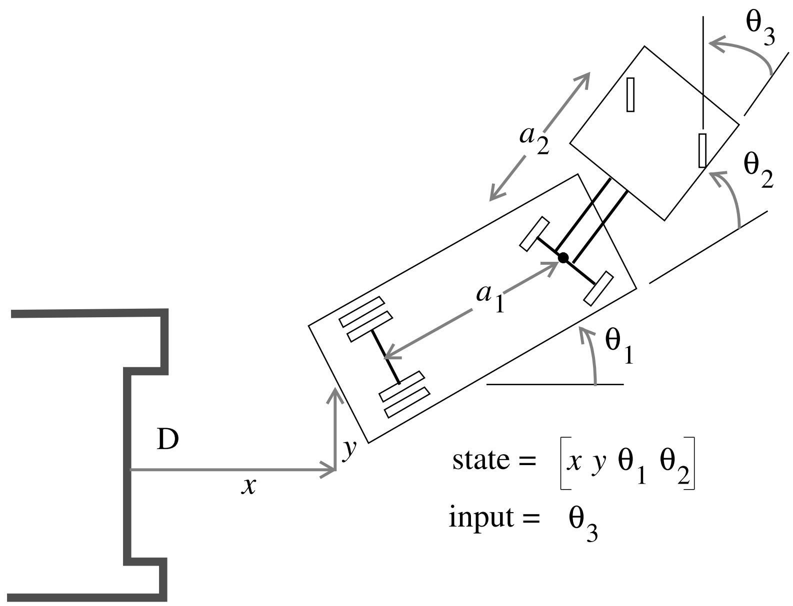


Figure 3.21 The truck-parking problem.
The goal is to park the trailer in the loading dock by controlling the steering angle, θ_3.

The system has a four dimensional state vector given by $[x\ y\ \theta_1\ \theta_2]$ and has the equations of motion as given in [61] and [79]. The state space is approximated using a grid of $11 \times 13 \times 13 \times 9$ nodes. The steering angle is quantized to -40, 20, 0, 20, and 40 degrees. Two arcs per input are generated. In this problem, x_D is of a set of states and not just a single state. The angle of the cab with respect to the trailer is allowed to take on any value in the final desired state, as long as the trailer itself is straight in the dock. The function to be minimized is time. The final approximating graph consists of 16731 nodes, 97088 arcs, and requires 12 iterations.

Figure 3.22 shows the controller in operation from one starting configuration. When no control values are available for a given node, it means the parking dock is unreachable from the current configuration. In effect then, the controller knows immediately if it is possible to park from a given configuration. Obstacles can be handled by eliminating all nodes from the graph (and their associated arcs) that represent collision configurations. The resulting controller works quite well over a large range of starting positions. From a few starting positions, however, there can be a relatively large final docking error. This is a result of the discretization of the states and inputs necessary to create the approximating graph. Relatively large final errors in arriving at the destination state can occur when applying AGDP to nonholonomic systems such as the truck-and-trailer because final corrections in the desired direction are often impossible to make.

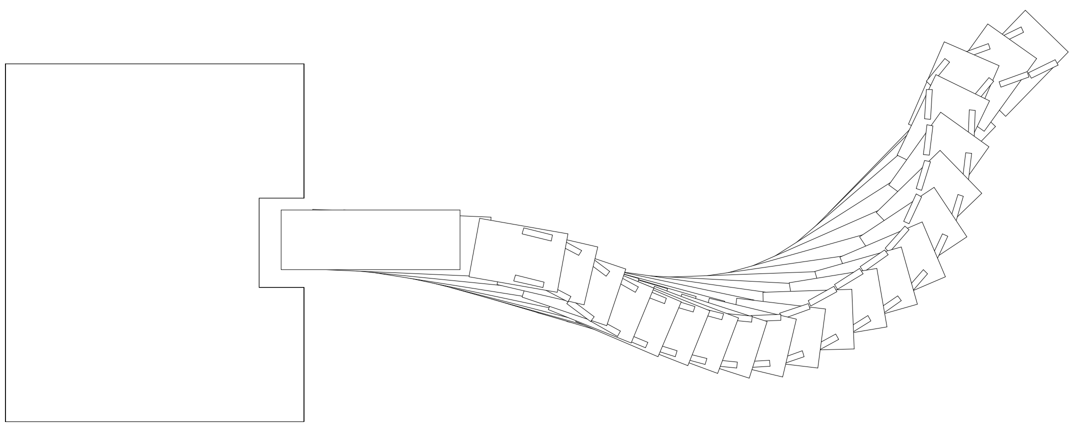


Figure 3.22 Truck parking using controller.

3.4.5 The Pole-and-Cart Problem

A classic problem in control theory is that of balancing an inverted pendulum placed on a cart, as shown in Figure 3.23. The goal is to balance the pole, as well as to drive the cart to the desired centre position. Using AGDP to create a controller requires no linearization of the system about an operating point and requires very little knowledge about the system. The equations of motion for the system are the same used in [9] and [21]. As our controller domain we choose $-2 \le x \le 2$ (m), $-1 \le \dot{x} \le 1$ (m/s), $-30° \le \theta \le 30°$, and $-100 \le \omega \le 100$ (deg/s). The input force is constrained to $-10N \le F \le 10N$. These constraints, as well as large pole angles, pose problems for a linear solution.

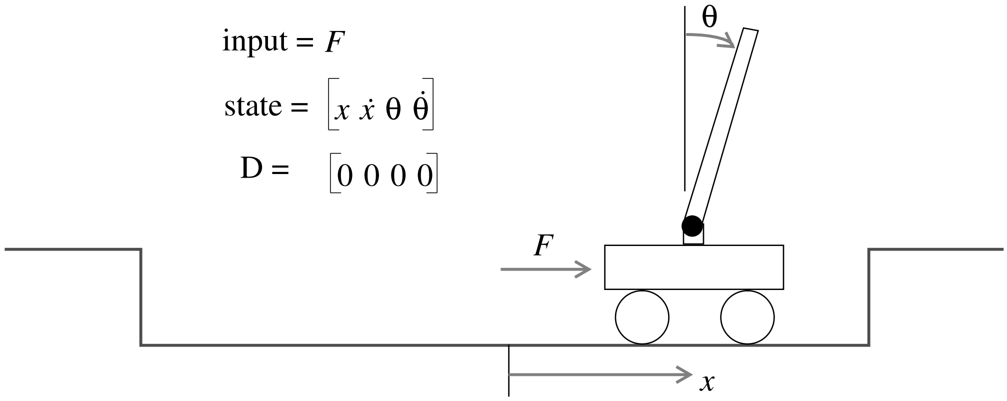


Figure 3.23 The pole-and-cart problem.
The goal is to centre the cart on the track while balancing the pole on the cart. The control variable is a force F applied to the cart.

The approximating graph for this problem has a grid of $10 \times 10 \times 10 \times 10$ nodes. The cart force is quantized into forces of -10, -5, 0, 5, and 10N. Two arcs are generated for each combination of input and node, producing a total of 62580 arcs. The optimization function is given by:

$$J = \int_0^T (1 + 0.005F^2(t))\, dt .$$

The solution requires 19 iterations before the control values for all nodes become fixed. This takes a short time to compute on a modern workstation[1].

Figure 3.24 shows the result of a trial run. The controller functions well across a large range of initial states in the controller domain. As in the previous problems, the cart is uncontrollable from some initial states because of the constraints placed on the cart's position and the forces that can be applied.

Because the AGDP controller is a sampled solution, a small limit cycle results around the destination state, although this is not discernible from Figure 3.24. An improvement to AGDP would be to increase the density of graph nodes in regions of state-space close to the destination

1. e.g.: 4 minutes on a Sun SPARCstation IPC (12 megabytes main memory)

state in order to keep limit cycles like these as small as possible. An adaptive discretization scheme would also be useful in regions known to have large errors.

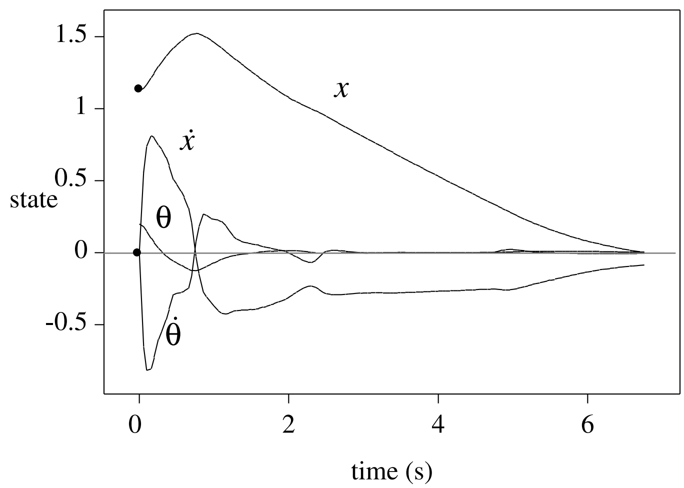


Figure 3.24 Controller operating pole and cart.
The optimization function is a weighted function of time and control energy.

3.5 Extensions

In order to make controllers calculated using AGDP suitable for a wider range of problems, several possible extensions are discussed in this section.

3.5.1 Combining Forward and Reverse Dynamic Programming

For large two-point boundary problems, it may be advantageous to use both FDP and RDP formulations of the same problem. In using RDP to determine the optimal trajectory for a two-point problem, the set of controllable states grows larger as the solution proceeds away from the terminating state. At some point, it will encompass many states that are not reachable from the starting state. This results in needless calculations being performed. A similar argument applies to using FDP alone. Calculations performed on states which are not controllable (cannot reach the end-state) are useless. The problem may be envisioned as shown in Figure 3.25.

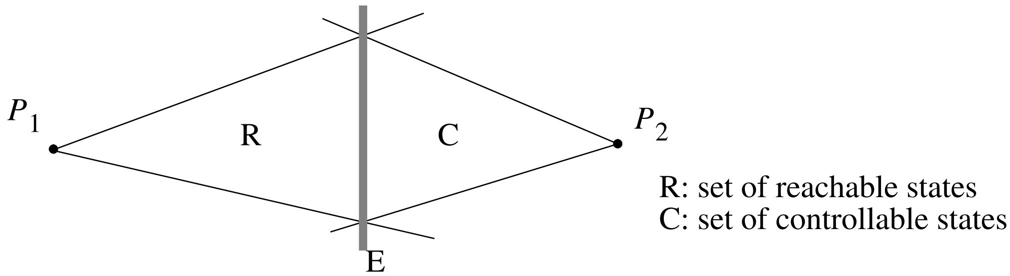


Figure 3.25 Set of reachable and controllable states in a two-point problem. The work in solving for the optimal path from P_1 to P_2 can be minimized by combining FDP and RDP solutions at the dividing hyperplane E.

Both FDP and RDP can be used together to solve a two-point problem by using a dividing hyperplane E, as shown in Figure 3.25. Allowable state trajectories should cross E in one direction only. This allows the portion of the optimal trajectory from P_1 to E to be calculated using FDP and the final portion from E to P_2 to be calculated using RDP. After solving the individual FDP and RDP problems, the point at which the optimal trajectory passes through E can be found by evaluating $I_{opt} = min_{P_E \in E}(I_{FDP} + I_{RDP})$. The point P_E at which I_{opt} is found defines the desired ending state for the FDP portion of the trajectory and the beginning state for the RDP controller.

3.5.2 Periodic Controllers

An interesting class of controllers can be constructed by giving the state-space a cylindrical topology. This leads to the creation of controllers that always drive the system towards an optimal trajectory that passes through a designated non-stationary state. This technique has been used to create a set of controllers for making an 11-link human model walk across variable terrain. This will be described in the next chapter.

3.5.3 High Order Systems

One of the major limitations of dynamic programming is that the problems grow exponentially in terms of memory and compute-time with respect to the dimension of the system to be controlled. Bellman called this the *curse of dimensionality* [11]. This problem is more serious in terms of calculating controllers than for calculating optimal trajectories. When determining optimal trajectories, multigrid techniques can be used [66]. In calculating a controller, one cannot use a similar scheme because we wish to know the optimal control throughout the entire controller domain. There appears to be no easy solution to the dilemma posed to dynamic programming by high-order systems.

There do exist some possibilities for obtaining small reductions in the size of dynamic programming problems. For many problems, a rectangular shape is not the best shape for the controller domain. Specific knowledge about the problem to be solved can be useful in constraining the shape of the controller domain, and hence the size of the resulting problem. Some problems can be decomposed into simpler ones by using appropriate independence assumptions. The 11-link walker described in the following chapter is an example of this.

3.6 Discussion and Conclusions

The results shown have illustrated the applicability of AGDP to a variety of problems that are usually solved by using numerical or analytic techniques more specific to each problem. Solutions to problems such as the truck parking and pole-and-cart problems have typically involved the use of controller synthesis techniques that contain elements of dynamic programming [9] [79]. We have shown that these kinds of problems can be solved in a single framework, namely that of AGDP. In this framework, control problems are effectively treated as path-planning problems through state-space.

AGDP represents a useful approach to generating optimal controllers for a variety of problems while placing few restrictions on the optimization function, system dynamics, and possible constraints. It does not require subdivision of the problem into fixed-length stages and deals effectively with arcs that do not terminate exactly on a node. It is different than its most similar predecessor, state-increment dynamic programming, in that the dependency relationships that the grid points have with each other do not have to be known a priori. The solution is shown to be an iterative process, the final steps of which correspond to a fixed-point iteration.

The error measures show that the resulting controllers are only approximations to the true optimal solutions. Error measure E_1 showed that the control table calculated for the $1/s^2$ system was far from optimal in some states close to the origin. This reflects the fact that the switching curve can only be approximated in the control table entries. This approximation becomes poor in regions close to the origin. The control table for the LQR problem provides a reasonable approximation to optimal control, as reflected by the E_1 error measure. The E_2 error measure shows that the estimated optimal costs, $I(x)$, were consistently higher than both the true optimal cost and the actual cost of using the controller. This is a result of the discretization that takes place in generating the approximating graph. Fortunately, the interpolation employed when actually using the controller results in a continuous range of control values being applied and thus a better approximation of the optimal control. The errors in approximating the optimal solution are not necessarily a problem in terms of the resulting animation because they do not in general have a large visual impact on the motion. As well, it is difficult to determine what the optimization function itself should be in order to lead to a desired motion.

There are several contributions that our work makes over previous work of a similar nature [28] [45] [59] [93]. We present a new way of dealing with the discretization and edge-condition problems. The accuracy of the resulting controllers has been compared against known optimal solutions. It has been shown that forward and reverse dynamic programming are exact duals of each other using AGDP. Periodic controllers have been generated that drive the system onto an optimal periodic limit cycle for walking, as will be shown in Chapter 4. Lastly, controllers are generated for more complex one-point boundary value problems (fourth order) than most examples in the literature (second and third order) [28] [45] [59]. For two-point boundary value problems, higher order problems can be solved [66]. However, the solution to two-point boundary value problems take the form of trajectories instead of controllers.

Given the general applicability of this technique and the ever increasing computational power of computers, AGDP represent a feasible method of calculating controllers for low order (5 or less) systems. Once an optimal controller has been calculated, it can easily be used in a real-time control process, requiring only interpolation of control values in a multi-dimensional table in order to use the solution. Dimensionality remains a problem, but we hope to promote the use of dynamic programming for systems more complex than previously considered using this technique.

Despite the flexibility of the AGDP method to deal with a variety of systems and its ability to approximate global optimal solutions, it has several serious drawbacks from the point of view of an animator. The most serious of these is that the technique is limited to objects of low complexity, currently two or three degrees of freedom which corresponds to a fourth or six order system. This is very restrictive from the point of view of an animator, who would likely be interested in controlling human and animal models of much greater complexity. A second drawback is that the animator is required to specify optimization functions, which is not an immediately intuitive method of influencing motions. Other aspects that are likely difficult for the animator to deal with are the specification of the maximum optimization function gradient Ψ, the state-space domain S, the destination state(s) D, as well as the discretization necessary to create the graph nodes X and quantized inputs U.

Chapter 4 A Variable-Terrain Walking Controller

Walking is a control action that most of us perform everyday without giving it much conscious thought. Despite the apparent simplicity, it has proven to be a difficult task to model. The control problem for bipedal locomotion has received attention from many researchers in the past. In this chapter we shall begin by summarizing some of the large body of work existing in this area. We then present a new control scheme capable of walking a human-like figure over variable terrain. The animator can design the terrain to be traversed and then tell the human character to walk across it.

Walking is a motion that animators are likely to want to model [16]. Unfortunately, the AGDP control technique of Chapter 3 does not directly lend itself to the walking problem. Even the simplest complete model capable of walking is too complex for this technique. One of the goals of this chapter is thus to examine how we might go about solving the more complex control problems that easily arise.

We shall use a planar model of a human figure with 11 links and 13 degrees of freedom. Because this is too large a system to handle using the general technique presented in Chapter 3, it is necessary to perform some decomposition of the control. The most important controller in this decomposition is that used to control the stance leg. The dynamic programming technique of Chapter 3 is used to calculate a series of control tables for the stance leg corresponding to various slopes of terrain. Our control scheme is shown to be capable of generating the internal torques necessary to make the biped model walk across smoothly varying terrain as well as climbing up stairs.

4.1 Related Work

There is an extensive body of literature on the problem of controlling biped walking motions. This is indicative of both the interest in the problem and its difficulty. We shall classify related work according to the type of control solution offered, and give a brief discussion on the merits and problems associated with each type of approach.

The earliest work on biped locomotion consisted of an analysis of human walking motions rather than their synthesis. The photographic studies performed by Muybridge [76] in the 1880's was one of the first attempts to accurately measure and characterize human walking motions. A more recent classification of human walking patterns can be found in [75]. Other recent analysis work on human walking is presented in [72], [1], [2], [46], and [41]. McGeer has performed an interesting analysis of *passive* walking in [69] and [70]. Appropriately designed mechanical walkers can sustain an unpowered walk on terrain having a shallow slope. This has been suggested as a good starting point for the study of powered walking systems. Analysis by itself does not yield motion controllers, however.

In the animation community there have been several papers that try to capture the essence of walking in a kinematic model. [110] presented some of the first kinematic walks using a skeletal animation system. A parameterized walk is presented in [16] which uses experimentally-derived kinematic relationships and a dynamic simulation of the swing leg to derive a hybrid kinematic-dynamic walking model for animation. In [36] a mix of kinematic and dynamic models are proposed to design legged animal motion. The system presented in [12] also uses experimental data to be able to produce sets of parameterized walking trajectories. Kinematic synthesis techniques have the problem that they are difficult to modify to reflect additional constraints. For example, changing a gait to reflect variable terrain has not yet been performed in kinematic techniques. Some notion of balance needs to be imposed on the motion control model. This is difficult to do in a purely kinematic technique.

Statically-stable walks have been used as a way to simplify the complexities of a dynamic walk. These systems move forward (albeit slowly) by always maintaining the centre of balance over the supporting foot. A variety of functional walking robots have been built upon this principle [80] [85] [86] [111]. Many six-legged robots have been constructed using statically-stable tripod gaits. While statically-stable gaits are perhaps a good starting point for creating more realistic dynamic walks, there has been no work detailing how such a progression might be achieved.

A motor-program model has been proposed as a control method for many types of motions [48] [5] [105]. This approach views control as a function of time. A given motion is achieved by playing back the desired control function. Unfortunately, the success of this method is sensitive to the initial conditions. Generating the control functions is often a trial-and-error process. Although no results for walking are known, this technique has been successfully used to control the flipping motion of a real biped robot [43] and a simulated articulated figure [94]. Motor programs unfortunately provide no means to react to their environment, restricting their use to specific situations. They represent purely open-loop control.

The observation that biped walking is similar to the motion of an inverted pendulum has been exploited in many schemes [26] [34] [35] [37] [53] [74]. In most cases the inverted pendulum model is linearized about the upright position in the control equations. In some work, ankle torques are required to provide suitable control during disturbances. In general, the inverted pendulum simplification is not useful for variable-terrain gaits and biped systems that bend significantly at the knee.

Another set of solutions makes use of prescribed nominal trajectories for all or parts of the system [42] [50] [101]. The necessary control can then be calculated using inverse dynamics and inverse kinematics. If the biped system begins to deviate from the planned trajectory, the control is calculated to drive the system back towards the nominal trajectory. This method has difficulties if the ankles are considered unpowered degrees of freedom. It also requires the a priori construction of a nominal trajectory.

Lastly, optimization techniques are a popular basis for the calculation of walking control. A control solution is generated on the basis of its optimality. It is assumed that human walking is optimal with respect to a fixed criterion (e.g., energy consumed). Because biped walking figures generally have at least 5 degrees of freedom and 4 control variables, simplifications must be made in order to make the non-linear optimization tractable. This can be done in several ways. In [91], the walking cycle is divided into several phases and the constraints specific to each phase are used. The control variables are calculated to optimize the motion of the remaining degrees of freedom at each time step. A similar approach is taken in [24]. Motions are optimized within each phase while ensuring that continuity is maintained between phases. In [22] the trajectory of the hip is restricted to those defined by degree 3 polynomials and an optimization is then carried out within this solution subspace. Optimal control methodologies are presented in [49] and [102], although no results are presented.

Our goal is to obtain an understanding of how a robust walking gait may be automatically generated, so we choose not to rely on any reference trajectories. We are also interested in producing a controller that is general enough to be able to walk a biped model across varying terrain. An inverted pendulum model is not well suited for modelling ascending and descending motions of a biped because the leg length varies greatly throughout such motions, and the motions are not symmetrical about the point of support. As a result, we choose to avoid restricting ourselves to an inverted pendulum for our analysis and synthesis. Lastly, we are interested in calculating a controller that can react to external disturbances. This precludes the use of costly optimization methods that determine only a single optimal trajectory.

4.2 The Biped Model

The biped model we shall use is shown in Figure 4.1. The various parameters for the links are given in Table 4.1. These parameters are approximately proportional to human biological data, with the exception of the torso and head, which are exceptionally heavy in our model [107]. It should be noted that the heavy torso is likely advantageous to our control decomposition, as will be seen shortly. The origin of the local coordinate system used for defining the location of the centre of mass (given as x and y in the table) is located at the point of attachment of the link to its parent link. For example, the origin of the coordinate system for links 9 and 11 is located at the elbow joint.

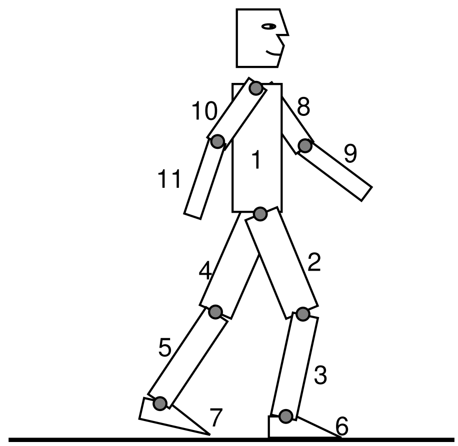


Figure 4.1 The biped model.
The planar figure is composed of 11 links, connected together with 10 pin joints.

Table 4.1 Parameters for the biped model

link	name	length (m)	mass (kg)	I (kgm^2/s)	x (m)	y (m)
1	head, trunk	0.80	63	1.48	0	0.24
2	upper leg	0.45	5	0.09	0	-0.225
3	lower leg	0.45	4	0.07	0	-0.225
4	upper leg	0.45	5	0.09	0	-0.225
5	lower leg	0.45	4	0.07	0	-0.225
6	foot	0.20	1	0.01	0.04	-0.03
7	foot	0.20	1	0.01	0.04	-0.03
8	upper arm	0.26	2	0.02	0	-0.13
9	lower arm	0.30	1	0.01	0	-0.15
10	upper arm	0.26	2	0.02	0	-0.13
11	lower arm	0.30	1	0.01	0	-0.15

The model is restricted to motion in the sagittal plane. All the joints are pin joints. Each of the 11 links has a degree of freedom in its angular position and the trunk, which we use as our root link, has an additional two degrees of freedom in its location, (x,y), yielding a total of 13 degrees of freedom. The state of the system can be represented by a 26-tuple, as each degree of freedom requires both a position and a velocity specification.

The foot has the dimensions shown in Figure 4.2a. The toes are not explicitly included in our model. Their function is assumed to be one of keeping the ball of the foot in stable contact with the ground as the heel lifts off. Our explicit modelling of the floor contact forces ensures this stable contact, and therefore we make the simplification of not modelling the toes as extra links. When the heel of the foot lifts off the ground, the foot is free to rotate about its front point, representative of the ball of the foot.

The biped figure is able to exert internal torques at each pin joint. We wish to determine how to modulate these torques to obtain a stable, natural walking motion. Gravity and the forces exerted by the floor are the only external forces involved. During the simulation, the forces exerted by the floor on the foot are calculated using a spring-and-damper model. Figure 4.2b illustrates the principle.

External forces exerted by the floor are calculated for two points on each foot, indicated

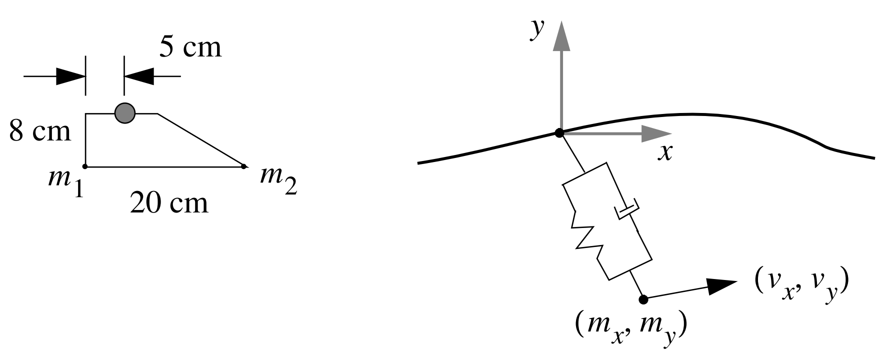


Figure 4.2 Modeling of the foot.
The ground reaction forces on points m_1 and m_2 are modelled using springs and dampers between the original point of contact on the ground and the points on the foot.

by points m_1 and m_2 in Figure 4.2a. The positions and velocities of these *monitor* points are used to calculate the external forces applied at these points as follows:

$$F_x = -m_x k_p - v_x k_v$$

$$F_y = -m_y k_p - v_y k_v$$

Typical spring and damper constants are $k_p = 10^5 N/m$ and $k_v = 10^3 Ns/m$. This creates a suitably stiff floor that functions effectively when used in a simulator with an adaptive time step. The advantage of modelling the biped as a free body is that the same formulation of the equations of motion can be used throughout the entire walking motion. There is no need to calculate the magnitude of any impulsive forces that occur upon impact of the feet with the ground. These are effectively approximated by the spring-and-damper model with a suitably small time step. A limitation of this simple floor contact model is that it does not exhibit slippage.

The equations of motion for our biped are too complex to derive by hand. In addition, we wish to have an efficient implementation to allow for experimenting with the walking controller in a reasonably interactive fashion. In previous work we have written a *dynamics compiler* [93] that produces a linear set of equations of the form $Ax = b$ given a physical description of the articulated figure as input. A and b are dependent on the internal torques, external forces, the physical properties of the links, and the state of the links. The linear set of equations are solved at every time step to yield the instantaneous accelerations, x. The accelerations are numerically integrated to determine the new velocities and positions of the links. The Euler integration procedure we use requires a small time step. Interesting results comparing the efficiency of various integration techniques can be found in [38]. Similar simulators to ours have been written by other researchers [33] and are also available commercially [87].

The output of the dynamics compiler gives the symbolic value of each of the elements of the matrix A and the vector b. Values of common subexpressions are precalculated to avoid unnecessary duplicate calculations. In our implementation, we use the recursive Newton-Euler formulation [6], and the values of A and b are output as lines of code in the ‘C’ programming language so that the equations of motion can be compiled. LU decomposition is used to solve the linear system of equations for the accelerations. An adaptive time step allows for maximum simulation speed while retaining numerical stability during collisions with the floor. The time step is allowed to vary between 0.00025 and 0.02 seconds.

4.3 Control Decomposition

A simplified block diagram of the biped system and its controller is shown in Figure 4.3. The controller uses the current state of the biped as well as information from the environment, namely the height of upcoming terrain, to calculate the 10 controllable inputs to the biped system. The control vector is calculated at the same rate as the simulation of the biped system. A typical time interval is 0.0006 seconds.

Some form of decomposition is necessary to simplify the creation of a suitable controller. Figure 4.4 shows the decomposition used. We shall begin with some general comments on the decomposition, and then proceed with more detailed descriptions of the specific subcontrollers involved.

We shall use the term *modular* to describe controllers that operate independently and that determine their output only as a function of their own states, which are non-intersecting substates of the entire system state. It can be seen from Figure 4.4 that the body controller is modular. We shall also define the term *hierarchical* to describe controllers that use common information to determine their output. In Figure 4.4 the controllers for the stance leg, swing leg, and arms all make use of a quantity we call the *phase* of the motion and thus qualify as being hierarchical. The

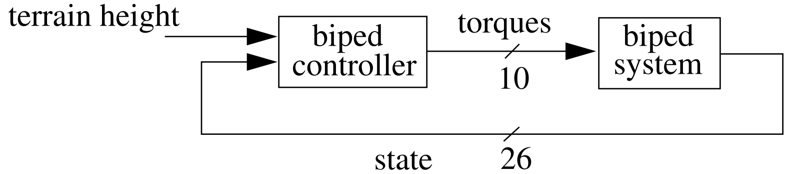


Figure 4.3 Simplified block diagram.
The biped controller uses the relative terrain height of the next step as well as the system state to calculate the internal torques to be exerted.

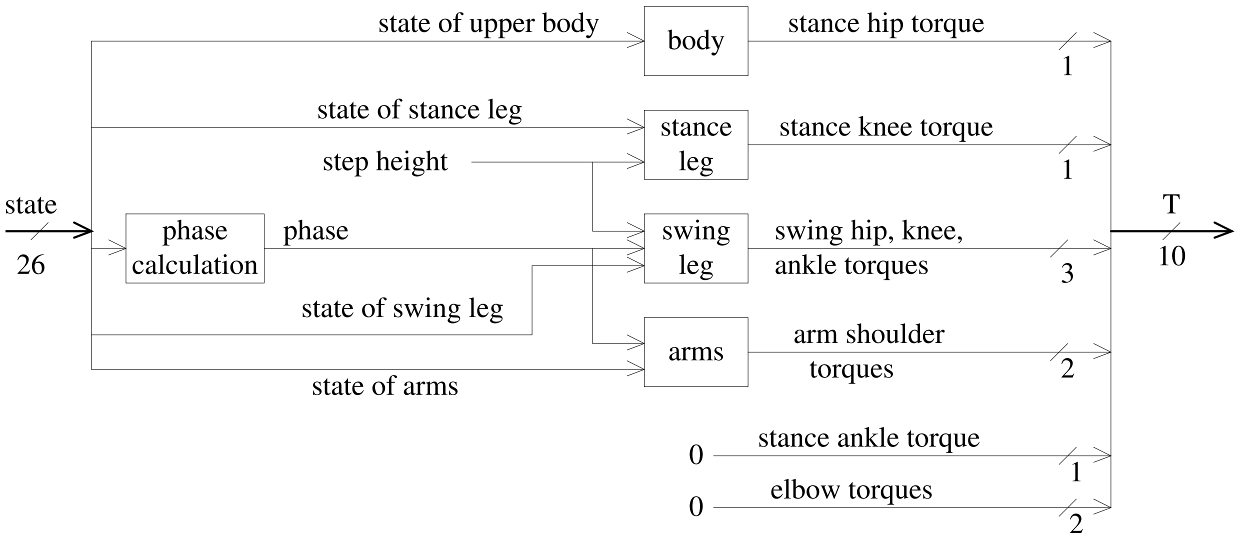


Figure 4.4 Control decomposition.
The motion of some of the subcontrollers are coordinated by sharing phase information.

phase is a number between 0 and 1 describing the progress of a step. It supplies the necessary coordination among all the links. The stance and swing leg also share information on the height of the next step to be taken. If the step length were to be made variable, the stance and swing leg would also share this information. We shall restrict ourselves to a fixed step length here.

In designing such a control system such as that of Figure 4.4, we require that the interference of controllers with each other is not significant enough to prevent the control system from working as a whole. We are not able to prove the stability of our controller decomposition in a formal sense, but we will show that it functions well in practice. Proving the stability of our control decomposition remains an interesting area for future research, although it is not crucial to its use in animation.

The most important subcontroller in our case is the stance-leg controller. In previous work on the dynamic walk of bipeds, the use of inverted pendulum models usually dictated the use of a straight stance leg. In this case the swing leg is used to control the step length and ultimately provides the necessary control over the gait. We choose to fix the step length and use the knee of the stance leg to provide the necessary gait control. While this has the disadvantage of creating somewhat awkward gaits over level terrain, it allows for control of gaits involving motion over variable terrain.

4.4 Stance Leg Control

The stance leg controller uses information from the portion of the state representing the state of the stance leg. In effect, the stance controller sees only a simplified model of the biped, shown in Figure 4.5. Mass M represents the combined masses of the trunk, arms, and swing leg. Masses m_2 and m_3 are as defined in Table 4.1. We shall choose cartesian coordinates to represent the state of the model of Figure 4.5. The state of this simplified system is given by (p_x, p_y, v_x, v_y) and is calculated using the substate of the stance leg in the biped.

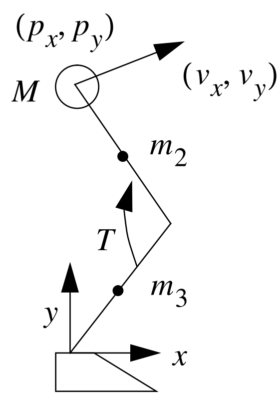


Figure 4.5 Model used for stance-leg control.
The masses of the trunk, head, arms, and swing leg are added onto the hip to form mass M, which is placed at the hip..

The state space of the simplified model is 4-dimensional. We shall define a region of interest in this space and call it our controller *domain*. Figure 4.6 shows the x and y dimensions of a 4-dimensional controller domain. The bounds placed on the x-dimension will serve to define the step length as well as the point at which the step and swing legs are exchanged. This exchange happens as soon as the hip exits the right-hand side of the controller domain (i.e. $p_x > x_2$), giving a step length of $x_2 - x_1$. The hip heights of interest are bounded by y_1 and y_2. Similarly, the velocities (v_x and v_y) of interest must also be bounded, although these two dimensions cannot be shown in the figure.

A *destination* state, shown as point D in Figure 4.6, must be specified. An optimization problem to be solved may now be informally stated as follows: from any state within the controller domain, move the system state from the current state to the destination state in an optimal way. The optimization function we use is $f_{opt} = \int_0^t |T| dt$, where T is the input we can provide, namely the torque on the stance knee, as shown in Figure 4.5.

To capture the fact that we wish to perform a periodic optimization, the edges of the controller domain at x_1 and x_2 should be connected to make the topology of the state space cylindrical in the x-dimension. This allows optimization over the entire period while only fixing one point on the period, namely the destination state.

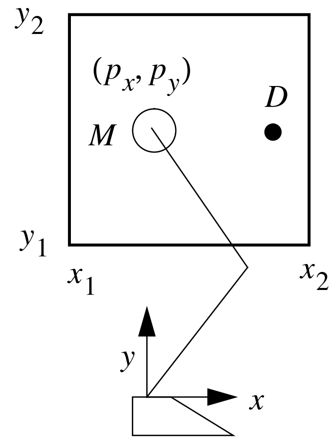


Figure 4.6 Defining the controller domain.
The hip of the model is driven to pass through the destination state D on every cycle. The edges at x_1 and x_2 are connected so that the periodic motion is optimized.

We use the dynamic programming technique outlined in Chapter 3 to solve the optimization problem just defined. This proves to be a convenient method of finding the optimal solution for non-linear systems such as the one in this problem. The optimal solution for the controller domain need only be calculated once and then stored in a table. This control (lookup) table then provides fast access to optimal control values when the controller is being used.

An interesting feature of this method is that it can be readily used to generate control values for ascending or descending steps. In this case, the topology of the state space becomes cylindrical with an offset. Figure 4.7 illustrates this principle. Consider a walk on ascending terrain. At the end of a step, the foot of what will be the new stance leg is a step height h above the foot of the old stance leg. Thus, when the exchange of the swing and stance legs occurs, the body will be lower with respect to the new stance foot by the step height h. This leads to a cylindrical topology with an offset, as shown in Figure 4.7. In this way, generating controllers for ascending and descending walks is no different than generating a controller for a level walk, except for a slight alteration of the topology of the controller domain.

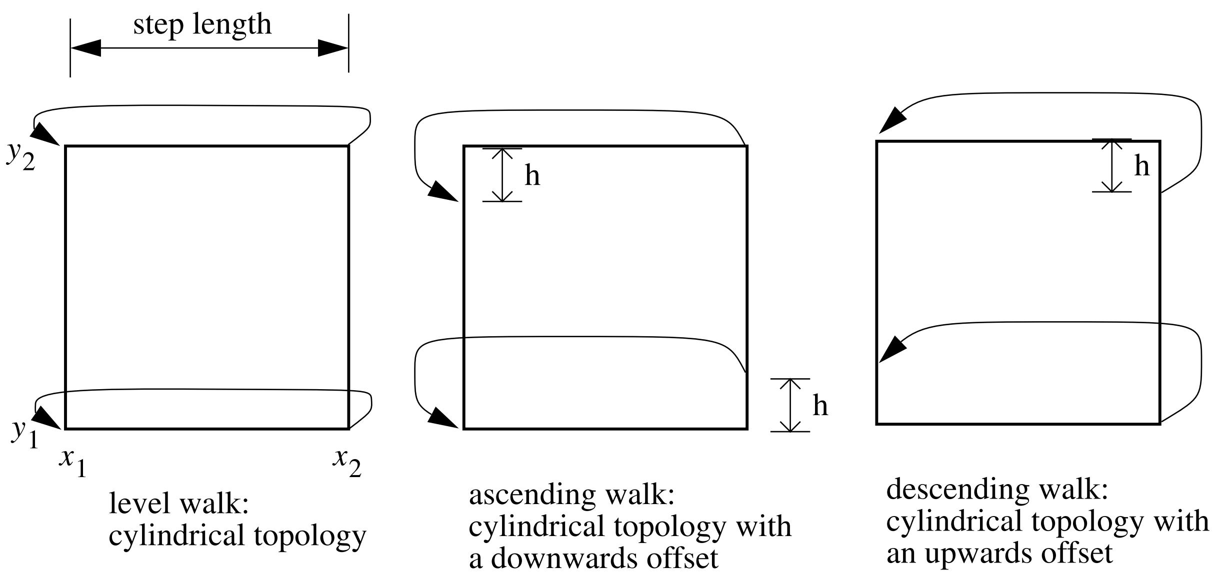


Figure 4.7 Controller domains for various slopes.
A vertical offset in the topology can be used to model walking up or down slopes.

The control tables for the stance leg are discretized into 10 steps along each of the four dimensions, creating a total of 10^4 entries. A series of 7 control tables are calculated for terrain slopes ranging from -0.3625 to +0.3625. Linear interpolation is used within control tables to retrieve appropriate control values and also between control tables to calculate suitable controls for arbitrary slopes. The stance-leg control tables are the key to the capabilities of the variable terrain biped walker.

4.5 Other Subcontrollers

The function of the swing-leg controller is to bring the foot of the swing-leg down one step length past the current supporting foot, while maintaining suitable clearance over the ground. The height of the ground one step length ahead is obtained from the environment and is used in helping to plan the trajectory. No specific trajectory is planned for the swing leg. Instead, two points are specified as being desired points for the ankle of the swing-leg to pass through. These are shown in Figure 4.8.

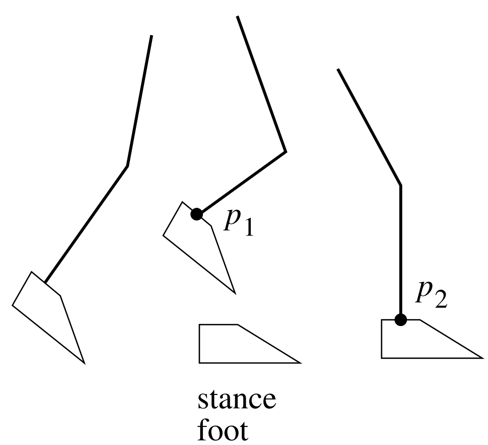


Figure 4.8 Swing leg control positions.
The swing control is governed by two key positions towards which the ankle is driven.

Point p_1 is a point in midstep with suitable ground clearance. Point p_2 is the desired location of the ankle at landing. Point p_1 should be reached halfway through the given step (phase = 0.5), and point p_2 should be reached just before the end of the step (phase = 1.0). Using the current phase, a desired velocity of the swing ankle can be determined such that the ankle will reach the desired point at the right time. Inverse dynamics of the swing leg is used to obtain the torques necessary to achieve the desired velocity of the swing-leg ankle. *Inverse dynamics* solves for the torques necessary to produce a given acceleration. This is complementary to forward dynamics (i.e., simulation), where the torques are known and the accelarations are unknown. Note that the inverse dynamics of the swing leg assumes that the motion of the swing leg is independent of that of the rest of the biped. This has proven to be a reasonable assumption for small step lengths, as our simulation results have verified. The swing-leg ankle is prepared for the slope of the coming step by using a simple proportional-derivative (PD) controller to generate the appropriate ankle torque. A PD controller is equivalent to a virtual spring and damper pulling the joint from the current position to the desired position.

The shoulders of the arms are driven by simple PD controllers to swing in phase with their 'opposite' leg. Because our model is restricted to the sagittal plane, the arm motion does not serve the same purpose as in real walking, namely to alleviate twisting motions of the body.

The trunk is maintained vertical by a PD controller with the hip of the stance leg being used to exert the necessary torque. Movement of the swing leg has a small but visible reactionary effect on the motion of the trunk, which is stabilized using the PD controller. The small oscillations of the trunk appear reasonably natural, however, when animations of the biped motion are viewed.

4.6 Results

The biped is capable of walking over both smoothly varying terrain and up stairs of various pitches. The step length is at present fixed to 25 cm, although we expect that the controller will also work for other relatively short step lengths. This step length of 25 cm is typical of steps taken while climbing stairs, but is much shorter than a typical normal step length of 80cm [16]. For large step lengths it is necessary to expand the stance-leg control model (Figure 4.5) to include the state of the swing leg. This would introduce difficulties in calculating an appropriate controller because it would increase the dimension of the problem to be solved, unless the motion of the swing leg was explicitly coordinated with the motion of the stance leg in advance.

Figure 4.9 shows the results of six walks. For clarity, the biped model is reduced to a stick figure, the arm motion is not shown, and only the motion of one leg is shown. The time step between positions is 0.083 seconds for all six walks. The calculation of the control, simulation, and display requires 1.5 seconds of real time for every second of simulated time on a Silicon

Graphics IRIS Indigo 2 workstation. This speed allows the system to be used in an interactive fashion. The user can design the terrain to be traversed in a drawing program and then immediately view a walk across the terrain. Figure 4.10 gives an example of the results.

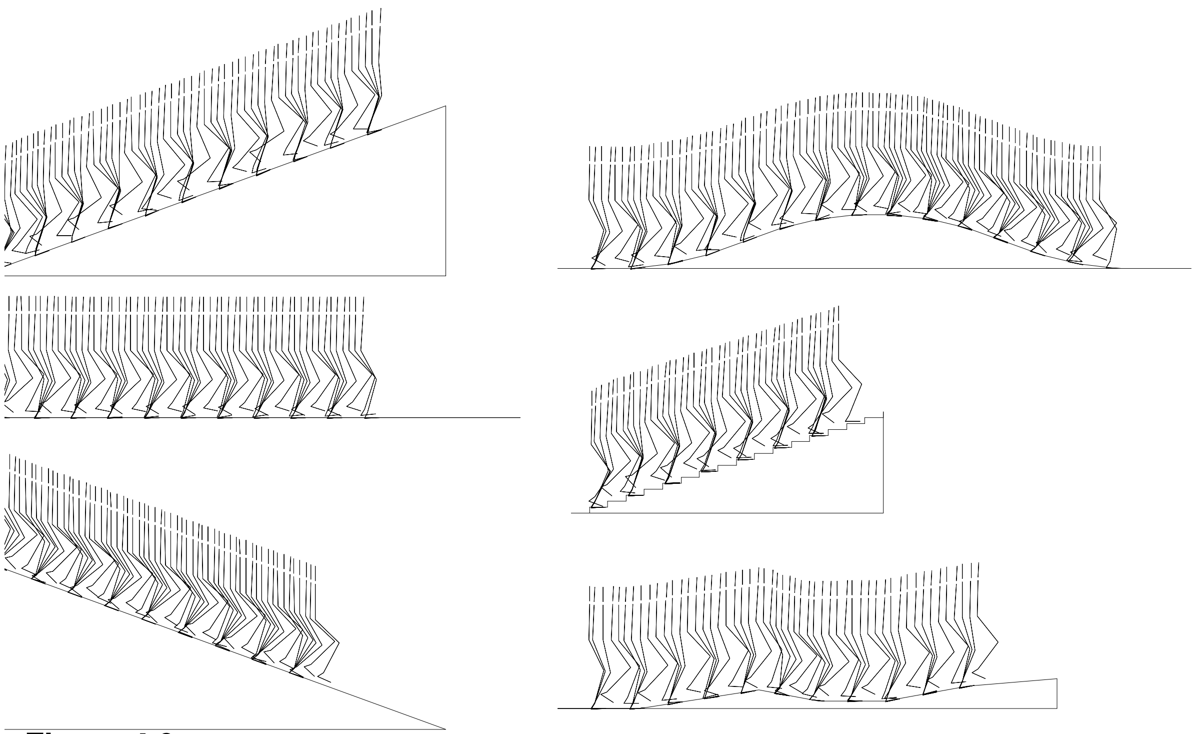


Figure 4.9 Some variable-terrain walks.
The motion of the arms and one of the legs has been removed for the sake of clarity.

Figure 4.10 The graphical display.

As mentioned previously, a stance-leg control table is generated for each of seven different slopes. The walks shown on the left of Figure 4.9 show three of these slopes. Linear interpolation is used between the seven control tables in order to be able to calculate stance-leg control for arbitrary slopes. The step length is constant across the variable terrain. The walking speed is slightly lower for ascending and descending than for level terrain. This occurs as a result of the chosen destination state and because of the optimization function that was chosen.

Our biped model has some unnatural artifacts when the results are compared to a human gait. We shall comment on some of these in a qualitative way. When walking steeply uphill on smooth terrain people tend to use their ankles so as to place their weight over the ball of the stance foot for most of the duration of the step. In contrast, our model has passive (zero torque) ankles during the stance phase, meaning the heel is likely to remain in contact with the ground during the entire stance phase. The trunk also remains more erect than in a comparable human making such a steep ascent. These two artifacts are likely related because placing the weight further forward over the ball of the foot requires the trunk to lean forward to keep the centre of mass over the support point.

Walks across level terrain occur with a larger flexion of the knees than in the equivalent human gait. The origin of this artifact is not known yet.

For downhill walks, the gait resembles that of a person descending on a slippery surface. This occurs as a result of the way the stance-leg control table is calculated. It is assumed that the velocity of mass M (i.e., the system momentum) in Figure 4.5 remains unchanged when the exchange of the swing and stance legs occurs. When the state passes through the right-hand side of the controller domain in Figure 4.6, it reappears on the left side unchanged except for the new value of p_x. No double stance phase is modelled. This is not necessarily the case for humans descending a hill. The instant at which the swing leg hits the ground to become the new stance leg proves to be a convenient point to dissipate some of the forward and downward momentum in real walking motions. This is not possible when the terrain is slippery and hence our model closely represents this case.

The controller drives the biped up a set of stairs in a reasonably natural fashion, as shown in Figures 4.9 and 4.10. The step width of the stairs was chosen to be the same as the step length of our controller. The good results for this terrain are attributable to the fact that the ankle in humans remains largely passive when climbing stairs, as in our controller.

Some of the awkwardness that appears in our walking gaits is a manifestation of the way we have structured the control problem and its solution. The AGDP technique of § 3 requires the system state to be continuous, something that is not true in general for walking. Adding ankle torques into the motion is difficult in the current problem formulation because this would lead to a 6-dimensional, 2-input problem to be solved using AGDP. Problems of such dimensions are unwieldy to solve using AGDP.

Some features of the walking gaits could be improved within the current control structure. Sets of control tables could be generated corresponding to various step lengths. step adjustments to control speed could be accomplished by switching between the appropriate sets of control tables. True three-dimensional walks and turns might be achieved by adding a modular controller to control the foot placement in the frontal plane.

4.7 Conclusions

A controller capable of driving a fully dynamic simulation of an 11-link biped across variable terrain has been presented. The control decomposition makes several assumptions about the independence of the motion of various links of the biped. These assumptions are verified experimentally when the biped motion is simulated as a whole. The results show that the control decomposition used is a useful one.

The stance-leg controller is the most important subcontroller in giving our biped model the ability to walk across variable terrain. The method used for generating the stance-leg control table is able to deal with ascending and descending gaits in the same fashion as level gaits. The control tables calculated for the stance leg are a useful technique for using the results of a family of optimal control solutions in real time. Dynamic programming is a useful technique for calculating the control tables. In effect, the control problem is converted into a path-planning problem through the state space.

Additional research needs to be performed into determining the changes necessary to make the resulting motion more natural. A more realistic mechanical model including a model of the muscle biomechanics is likely a useful step in the right direction. A useful series of mechanical models for walking of increasing complexity is presented in [46]. A study of the stability of the model across a range of terrain slopes, step lengths, and optimization functions is also necessary. A more quantitative comparison with human walking gaits could also be used to isolate specific aspects of the modelling that require improvement. It would be interesting to investigate transitions from walking gaits to running gaits and vice-versa.

At present the interaction of the animator with the walking controller is limited to the construction of the terrain that is to be traversed. If the walking controller were parameterized in terms of step length, walking speed, and perhaps the optimization metric, the animator could influence the motion by controlling these parameters as well. The animator would be ‘keyframing’ in this parameter space instead of directly keyframing the motion. For a true three-dimensional walking model, an animator will likely want to draw a path for the walking motion to follow.

The construction of the controller detailed in this chapter is complex enough that it is unreasonable to expect an animator to understand its intricacies. The details of the control mechanism are best hidden from the animator by providing a set of precalculated, parameterized controllers for walking. While this makes the resulting animation system easy to use, it also imposes strict limitations on the walking motions available to the animator. As a result, the walking controller is best used together with traditional keyframing techniques.

Chapter 5 Control for Turning Motions

This chapter provides another strategy for simplifying the control of complex systems we wish to animate. We shall deal with the modelling of turning motions such as those found in alpine skiing and bicycling. As in the previous chapter, we make some simplifications in order to be able to deal with complex systems. We make use of a simplified physical model, whose motions are then *amplified* to control the motion of a more complex display model. We will show that four turning sports can be effectively modelled using an identical simplified physical model.

The act of turning requires steering, which performs two functions: changing the direction of motion and ensuring that balance is maintained. We shall analyze and control turning motions for our simple physical model. A phase-diagram description of periodic turning motions (as in slalom skiing) is shown to be a useful representation. The phase diagram is used to construct a control algorithm parameterized in terms of frequency, sharpness and heading of the turns. A second method of control allows an animator to draw an arbitrary path for the turning figure to follow while avoiding obstacles. A finite-time optimization is used to find the best physically-feasible motion that closely follows the desired path. Examples of alpine skiing, snowboarding, bicycling, and telemark skiing are given.

5.1 Introduction

The turning motion of a bicycle is a graceful manoeuvre that is somewhat more complicated than it looks. A turn involves controlling the direction of motion and balancing at the same time. Similar turning motions exist in sports such as skiing or surfing. The goal of this chapter is to provide a suitable physically-based model for this class of motions, and more importantly, to come up with general control techniques for turning motions. These control techniques should be easy and intuitive for an animator to apply.

The next section briefly discusses previous related work. The physical model we use and its equations of motion are given in section 5.3. We present a phase diagram useful for analyzing and synthesizing periodic turning motions in section 5.4. A parameterized controller for the model is given in section 5.5. A path-following controller is described in section 5.6. Examples of how to map display models onto the physical model are given in section 5.7, and a discussion concludes this chapter.

5.2 Related Work

The difficulty of creating a suitable controller increases greatly with the complexity of an object. As a result, it is often useful to work with a simplified physical model. This is evident in the work presented in [83], where the locomotion of various types of hopping figures is controlled. In this work, specific constraints are placed upon the roles of the various body parts. This reduces the number of control variables and makes them largely independent of each other, thereby simplifying the control problem. The controllers generated are hand-tuned for each creature, but they are parameterized at a high level.

A mix of dynamic and kinematic methods are used in [16] for the animation of walking. This work is similar to the work to be presented here in that we shall also make use of a simplified physical model and amplify its motion to control a more complex display model. The physics of walking and turning are very different, however. As well, the derivation of our parameterized controller is performed analytically rather than through the use of experimental data. Lastly, our path-following algorithm provides an intuitive method of control.

Little work has been done on a mechanical analysis of turning motions. [90] presents an experimental study on the displacement of a skier's center of gravity during a turn. It is not clear how this experimental data can be used as the basis for actually controlling a turning motion.

Some qualitative analysis of turns can be found in popular ski magazines. Filmed skiing motions are used to recreate animated skiing motions in [63].

5.3 A Physical Model for Turning

In order to understand our choice of simplified physical model, it is helpful to begin with some simple intuition on how turning works for bicyclists, skateboarders, and other turning figures. It is clear that it is necessary to 'lean into a turn' in order to remain properly balanced throughout the turn. It is not so clear whether the turning occurs as a result of leaning, or if leaning occurs as a result of turning. We shall argue for the latter.

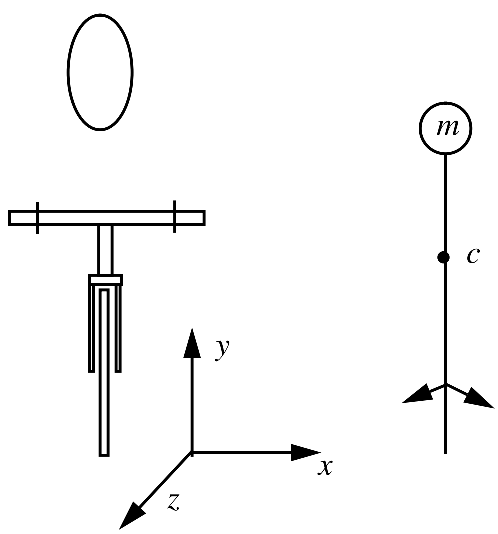


Figure 5.1 A cyclist and a simplified inverted pendulum model.

We shall use the cyclist in Figure 5.1 to illustrate that the lean angle is not something that can be controlled directly, and therefore that leaning occurs as a result of turning. In order for the cyclist to attain a leaning position, we require a torque acting around the z axis. No mechanism for applying such a torque exists. It is possible to effect small changes in the lean angle using body motion alone, as demonstrated by skilled cyclists who can remain balanced while stationary. It becomes much easier, however, to retain ones balance when moving forward on a bicycle, even if only moving slowly. At low speeds, the effect of gyroscopic forces is minimal, and we can conclude that it is the ability to turn that allows for easy balancing and control of the lean angle. A lean is thus obtained by moving the support out from underneath the body. This is easily done by making the point of support follow a curved path (i.e. turning). We shall refer to the point of support of our turning figuress as the *supporting foot.* Thus in the case of a bicyclist, this refers to the points of the tires in contact with the road.

As an aside, it is worthwhile to explain how one can ride a bicycle without having hands on the handlebars, in which case we have no direct control over the steering. One can exert indirect control over the steering by bending sideways at the hips, shown as point *c* on the simplified model in Figure 5.1. Bending in this fashion will not directly change the lean angle of the mass with respect to the support, but it will tilt the bicycle, which forms the lower part of the supporting 'leg'. Bicycles are constructed such that when they are tilted, the front wheel turns, even when stationary. This mechanism allows for indirect control of steering by bending sideways at the hips.

All the figures we shall deal with have the ability to control the curvature of the supporting 'foot'. Bicyclists can turn their front wheels. Skiers place their skis on edge, which causes the skis to bend into an arc shape because of their *sidecut*. Sidecut in a ski causes the ski to be thinner at the waist than at the tip and tail, when seen from above. Skateboards turn because of the *trucks* that cause the wheels to twist when the board is tilted. Trucks are the mechanical joints placed between the wheels and the board. They transform a tilt of the board into a twisting motion that alters the direction of the wheels. Telemark skiers, loosely equivalent to going downhill on cross-country skis, separate their skis and place the front ski at an angle with respect to the rear ski in order to turn. Figure 5.2 illustrates these configurations.

Figure 5.2 The physical model and 4 turning sports.
The turning motions of snowboarding, bicycling, telemark skiing, and alpine skiing can be modelled using the same physical model with different display models.

In order to make the control algorithm as simple and general as possible, our physical model of the turning figure is the inverted pendulum shown in Figure 5.3. The model has its own coordinate system located at its foot. The location of the foot is further defined in a world

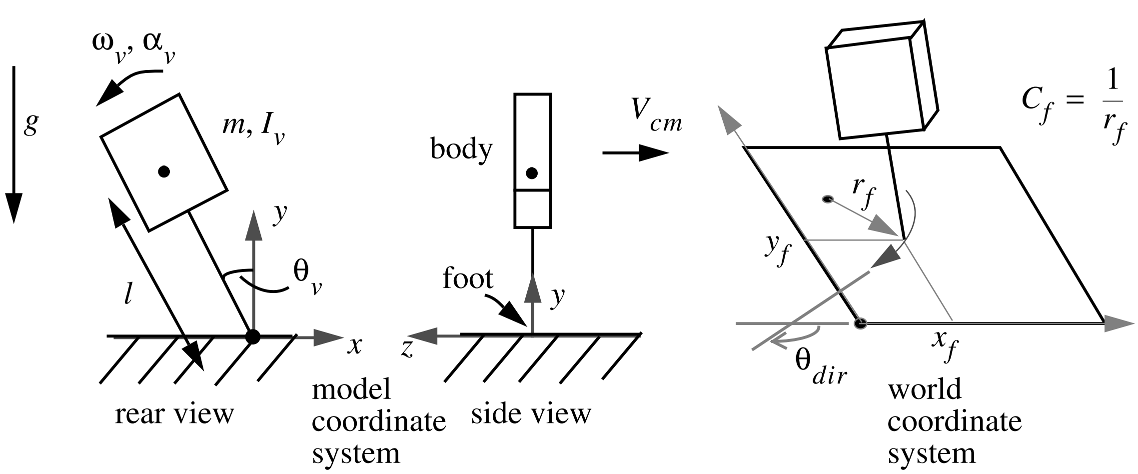


Figure 5.3 The physical model used for turn synthesis.
The model consists of an inverted pendulum with a steerable 'foot'.

coordinate system. The body has a known mass m and moment of inertia about the z-axis, I_v. The body moves forward along its z-axis with a velocity V_{cm} over a planar surface. The centre of mass is located at a distance l above the foot. The origin of the coordinate system is fixed to the 'foot' of our model, with the y-axis perpendicular to the ground and the negative z-axis aligned with the direction of motion. The state of the physical model is given by the 6-tuple $(\theta_v, \omega_v, \theta_{dir}, x_f, y_f, V_{cm})$.

The moment of inertia about the x axis of the model is ignored because we assume the leg to be always vertical in the *yz*-plane. This is sufficient for the turning figures we shall consider; it is a simple matter for bicyclists and skiers to avoid falling forwards or backwards when moving across smooth terrain. The moment of inertia about the y axis is ignored because rotation about the y axis is caused by the constraint placed on the motion by the curvature of the foot, and not some direct torque about the y axis. A constant velocity assumption is not crucial in our development of the equations of motion. It is, however, necessary for obtaining our parametric controller in an analytic form in section 5.5. The model freely rotates about the z-axis of its foot. The curvature of its foot in the xz-plane, given by C_f, is the controllable input. That is, the model can be steered. C_f is defined as $1/r_f$, where r_f is the instantaneous radius of curvature.

Because of the simplicity of the physical model, a mapping relation is necessary to relate the state of the physical model to the more complex *display* model. The mapping relation allows for the kinematic coordination of various body parts with that of the physical model, even though they are not explicitly included in the physical model. In this way subtle motions such as appropriate twists of the upper body can be generated. These motions would be difficult and prohibitively expensive to synthesize in a fully-dynamic solution to the same motion. For example, optimal control techniques such as dynamic programming have an exponential complexity with respect to the size of the state vector.

The problem of using a more complex physical model is one of calculating the control to generate the appropriate motion. If one were to use an optimization technique to generate the necessary control, not only is the optimization itself difficult to perform for complex systems, but it is also not clear what the optimization function should be. Our approach makes use of a physical model to generate the dominant part of the motion. Kinematic relationships are used to generate the motion of parts whose movements are less dependent on constraints imposed by physics and can also be used to capture more aesthetic or irregular aspects of the motion. A simple physical model also allows for a more general control solution because, as is the case here, a group of diverse sports can make use of the same simplified physical model, and hence the same controller.

Example mappings of several display models onto the physical model are shown in Figure 5.2. The mapping procedure itself is discussed in greater detail in section 7. The equations of motion for the physical model of Figure 5.3 are derived in Appendix A. We are interested in the relationship between the curvature of the foot, C_f, and the angular acceleration about the z-axis of the model, α_v. Because C_f is controllable, we shall define the forward dynamics as $\alpha_v = f(C_f)$ and the inverse dynamics as $C_f = f^{-1}(\alpha_v)$. The inverse dynamics is governed by the quadratic equation

$$0 = aC_f^2 + bC_f + k, \tag{5.18}$$

where C_f is the curvature of the foot, and

$$a = l\sin\theta_v k^2 + lV_{cm}^2 \sin\theta_v \cos\theta_v$$

$$b = -2l\sin\theta_v k - V_{cm}^2 \cos\theta_v$$

$$k = g\sin\theta_v - \frac{\alpha_v(I_v + l^2 m)}{ml}.$$

In the expressions for a and b, k is a quantity useful in simplifying the equation. The negative root of the quadratic is used when solving for C_f, as seen in Appendix A. We can use equation 5.18 to calculate the curvature of the foot needed to achieve a given angular acceleration of the body, α_v.

Because it is unrealistic for the curvature of a ski or the steering angle of a bicycle to be unlimited in value, it is necessary to place upper and lower bounds on C_f. As we shall see later, we shall be calculating a desired value of α_v, then using equation 5.18 to solve for the corresponding C_f, bounding the value of C_f to a reasonable one if necessary, and lastly recalculating the α_v corresponding to the new bounded C_f. This process is analogous to kinematically planning a motion (i.e., planning the position over time), using inverse dynamics to calculate the forces required to produce the motion, limiting the forces to being reasonable ones, and then recalculating the resulting motion. An expression that calculates the forward dynamics can be obtained by rearranging equation 5.18:

$$\alpha_v = \frac{lm\,(g\,sin\theta_v - k)}{I_v + l^2 m} \tag{5.19}$$

where k (the same quantity as in equation 5.18) is calculated as

$$k = \frac{C_f V_{cm}^2 cos\theta_v}{1 - C_f l\,sin\theta_v}.$$

Once α_v and C_f are known, all that is needed for a simulation of the motion of the system is a method for numerically integrating the state of the system. Euler integration is used. x_f and y_f are integrated knowing the current speed and direction. θ_v and ω_v are integrated using the known value of α_v. The current direction, θ_{dir}, is integrated knowing the current curvature and speed.

5.4 Phase Diagrams for Periodic Turns

It is readily observable that slalom-type turns result in an oscillation of the body about the vertical. This means that θ_v in our model should vary sinusoidally. We shall use this as the basis of one of our methods for controlling physically-based turning motions. By controlling the turns of the foot of our model so as to generate body oscillations about the vertical axis, we generate realistic slalom turns as a desirable side-effect. It is thus easier to analyze and synthesize turns in terms of the oscillation of the body about the vertical axis rather than in terms of the path of the foot. In this section we analyze how this oscillation can be described using a phase diagram and how to use the phase diagram for analysis and synthesis of turns. In the following section we shall describe how to use the phase diagram for generating conveniently parameterized turn controllers.

We shall begin by assuming that θ_v varies sinusoidally over time, as shown in equation 5.20. θ_{max} defines the amplitude of the oscillation and thus represents the maximum lean angle. Let ω_0 be the frequency of oscillation, or alternatively, the turning frequency.

$$\theta_v = \theta_{max} sin(\omega_0 t) . \tag{5.20}$$

Differentiating with respect to time gives expressions for the angular velocity and angular acceleration about the foot:

$$\omega_v = \omega_0 \theta_{max} cos(\omega_0 t) , \tag{5.21}$$

$$\alpha_v = -\omega_0^2 \theta_{max} sin(\omega_0 t) . \tag{5.22}$$

Equation 5.22 is an expression for calculating α_v, which can be used as a controlling function through the inverse dynamics equation. Thus, we can control the curvature of the foot so as to create a desired sinusoidal body oscillation of a given amplitude and frequency. Although the body is made to oscillate sinusoidally, the path taken by the foot is determined by the inverse dynamics and is thus not a perfect sinusoid.

The state of this system is represented by θ_v and ω_v, and moves clockwise along an elliptical trajectory as shown in Figure 5.4. We shall call this the *phase diagram* of a turn. The ellipse is defined by the constants θ_{max}, the maximum lean angle, and ω_0, the turning frequency.

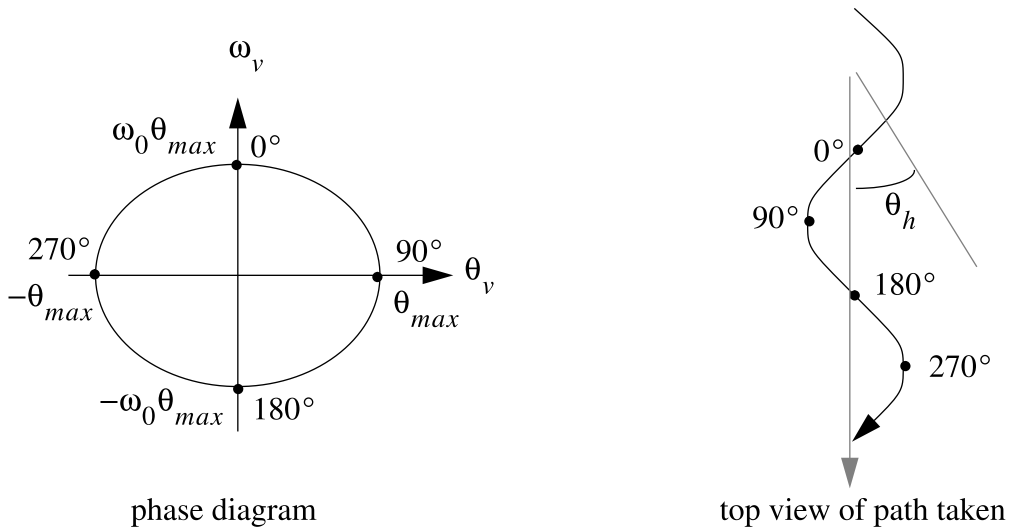


Figure 5.4 The phases of a turn.
The phase diagram describes the sinusoidal oscillation of the physical model about the vertical position. The steering required to create this motion creates the approximately-sinusoidal track for the foot, as shown on the right.

The current *phase* of the body is given by $\phi = \omega_0 t$. It can also be calculated directly from θ_v and ω_v as follows: $\phi = \omega_0 t = \arctan(y_\theta / x_\theta)$ where $y_\theta = \theta_v / \theta_{max}$ and $x_\theta = \omega_v / \omega_0 \theta_{max}$. The phase is dependent on both the state of the body as well as θ_{max} and ω_0, which define its planned motion. The phase represents useful information for the mapping of display models onto the physical model. The simulation and open-loop control cycle using equation 5.22 is described by the pseudocode in Figure 5.5..

choose (ω_0, θ_{max})
while(TRUE) {
  $\alpha_v = -\omega_0^2 \theta_{max} sin(\omega_0 t)$
  $C_f = f(\alpha_v)$
  numerical integration of state
  display $(C_f, state)$
  $t = t + \Delta t$
}

Figure 5.5 Pseudocode for open-loop control of turn.

The correspondence of various points in a turn to the phase diagram is illustrated in Figure 5.4. The shape of the path taken by the foot is not an exact sinusoid, as will be further explained in the next section. The general direction of the turning figure is given by its θ_h, which defines its heading with respect to a fixed axis.

Figure 5.6 illustrates how the various parameters of the model vary over a complete period for a typical turn. F_{leg} represents the compressive force experienced by the leg and is a useful parameter in the mapping process. The amplitudes have been scaled as necessary for this figure. This figure also gives an insight as to why turning sports can look so effortless. The angle of body lean, θ_v, and the curvature of the foot, C_f, must be in phase to cause the appropriate turning motion. This happens naturally in turning sports because of the construction of skis, surfboards, and skateboards. Tilting any of these devices causes a curved motion proportional to the angle of tilt. Because θ_v and C_f are always naturally in phase, the motion requires little effort to sustain, although it may in practice be difficult to initially achieve. This means that on a skateboard or snowboard, very little ankle motion is required to tilt the board; the tilt angle between one's body and the ground is close to giving the board the desired tilt angle already.

.

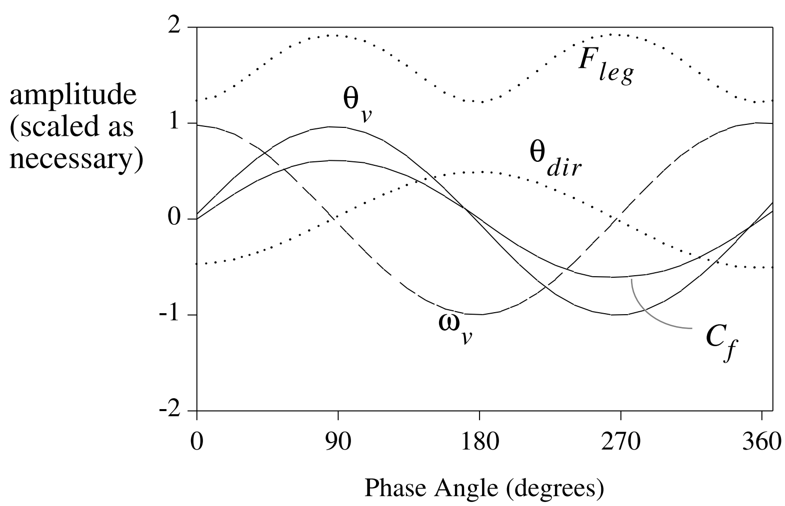


Figure 5.6 Phase relationships for a typical turn. Refer to the text for a description.

Figure 5.7 shows several tracks left by the 'foot' of the physical model for various values of ω_0 and θ_{max}. Phase ellipse 1 has $\omega_0 = 2\pi/2.0$, $\theta_{max} = 23°$; phase ellipse 2 has $\omega_0 = 2\pi/3.0$, $\theta_{max} = 23°$; and phase ellipse 3 has $\omega_0 = 2\pi/2.0$, $\theta_{max} = 7°$.

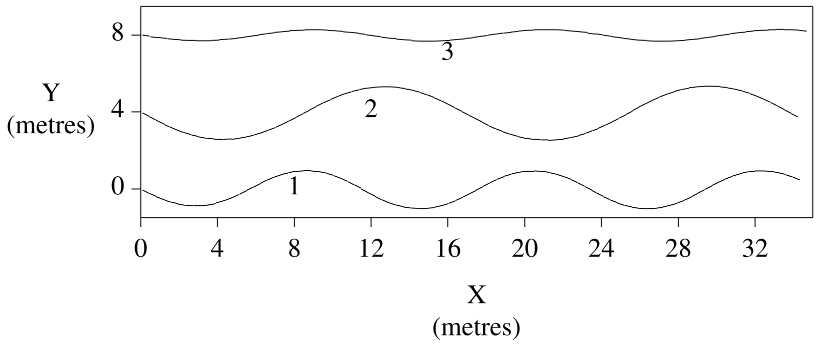


Figure 5.7 Turning examples.
Tracks 1 and 2 have the same turn amplitude, as specified by θ_{max}.
Tracks 1 and 3 have identical turning frequencies.

To summarize, the control variable C_f is calculated by determining the curvature necessary over time to oscillate the body in a sinusoidal fashion.

5.5 Parameterized Control

In this section we look at how to move from one phase-ellipse to another and how to control the general direction of motion for slalom-type turns. The control scheme that we propose will place three variables under the animator's control: ω_0, θ_{max}, and θ_h. The first two define the phase ellipse, and the last defines the general direction of motion. Using the parameterized controller requires specifying the values of these three parameters over time, namely: the turn frequency, turn sharpness, and general heading of the turns.

5.5.1 Control of turn frequency and sharpness

We shall first present a method of taking the system from any current state onto a desired phase ellipse. The control has the purpose of allowing the turning body to move from one phase ellipse to another as well as achieving closed-loop control once on a phase ellipse. A simple solution can be obtained if we look at the sets of phase ellipses that share the same major and minor axes, as shown in Figure 5.8. These ellipses can be used to take the system onto target phase ellipses, shown in bold.

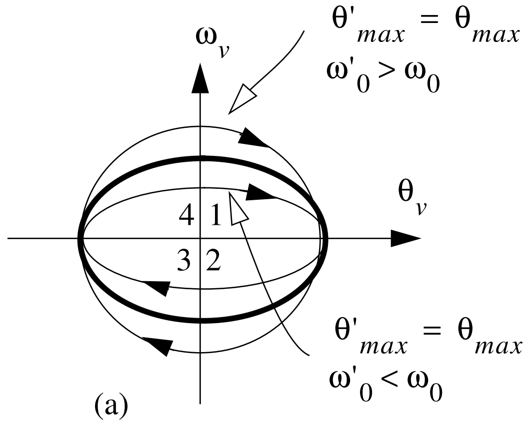


Figure 5.8 Sets of phase ellipses sharing an axis with the desired phase ellipse. These ellipses can be used to bring the system back onto the target phase ellipses, shown in bold.

The ellipses in Figure 5.8a can be used for states in quadrants 1 and 3 to bring the turning figure onto the target ellipse defined by ω_0 and θ_{max}. Similarly, the ellipses in Figure 5.8b can be used for states in quadrants 2 and 4. We thus need to calculate the values of ω'_0 and θ'_{max} that define the controlling phase ellipse that will bring the state of the turning figure onto the desired ellipse.

For quadrants 1 and 3, $\theta'_{max} = \theta_{max}$, and $\omega'_0 = \omega_v / (\theta_{max} cos\phi)$. For quadrants 2 and 4, $\omega'_0 = \omega_0 / \beta$ and $\theta'_{max} = \beta\theta_{max}$ where the value of β can be calculated from equation 5.20 as $\beta = \theta_v / (\theta_{max} sin\phi)$.

Once the controlling ellipse is known, equation 5.22 is used with these newly calculated values to determine α_v and hence the desired control value C_f, as before. The controlling ellipse is recalculated at every time step because θ_{max} and ω_0 are changing functions of time specified by the animator.

Although Figure 5.8 implies that the desired ellipse can always be reached within 90 degrees of phase, bounds placed on acceptable values of C_f and also dC_f / dt usually lengthen the time required to reach the desired phase ellipse. These bounds reflect the physical bounds on the curvature and rate of change of curvature of skis and wheels.

The preceding solution does not work equally well in all regions of the ω_v-θ_v plane. States having $|\theta_v| > \theta_{max}$ or $|\omega_v| > \omega_0\theta_{max}$ do not lie on any of the sets of ellipses shown in Figure 5.8. For these regions one can directly assign values of α_v that drive the system towards the desired ellipse: $\alpha_v = \alpha_{max}$ for quadrants 2 and 3, and $\alpha_v = -\alpha_{max}$ for quadrants 1 and 4.

The origin, representing a perfectly balanced vertical state, is a singularity where there are also no proper controlling phase ellipses. A fixed arbitrary value is assigned to α_v for a small region around the origin.

5.5.2 Directional Control

The last useful parameter is the direction. Figure 5.9 shows an example of a change in direction of $\Delta\theta_h$, where θ_h represents the desired direction that the animator specifies as a function of time. It indicates the direction with respect to some fixed axis.

In Figure 5.9 we see that a change in direction to the left is performed using the short right-hand turn found at $x = 26m$. Similarly, a small change in direction to the right can be effected by making a sharper, longer right-hand turn or alternatively, making a shallow, shorter left-hand turn. We shall use these methods to perform our directional control.

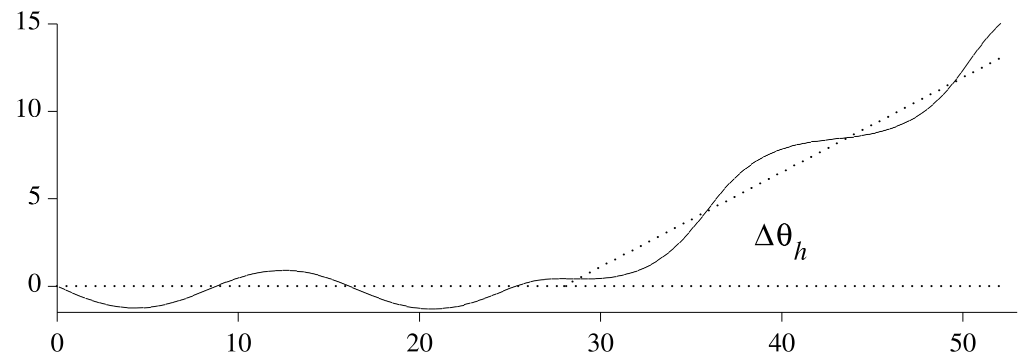


Figure 5.9 An example of direction control.
A top view is shown of a change in direction to the left. This is accomplished with the short, shallow right turn found at x=26m.

Let the instantaneous direction be given by the function $\theta_{dir}(\omega_0 t)$, which is modulated by foot turning, as specified by C_f. In our control scheme thus far, we use the turning of the foot solely to control the body lean (e.g. θ_v and ω_v), and not to purposely change direction. The maximum change in direction over a complete left turn is given by:

$$\Delta\theta_{dir} = \theta_{dir}(180°) - \theta_{dir}(0°) \tag{5.23}$$

In order to effect a change of heading of $\Delta\theta_h$ in a single turn, as shown in Figure 5.9, we require

$$\Delta\theta'_{dir} = \Delta\theta_{dir} + \Delta\theta_h. \tag{5.24}$$

The modified turn should be continuous with respect to the turns performed before and those that must be resumed afterwards. We shall assume here that the turns before and after the change in direction are the same, and are specified by ω_0 and θ_{max}. Figure 5.10 shows the phase diagram of a modified turn. As with the ellipses of Figure 5.8b, the modified turn is defined by a number β. $\beta > 1$ results in a sharper, longer turn than the original. Similarly, $\beta < 1$ results in a shallower, shorter turn than the original.

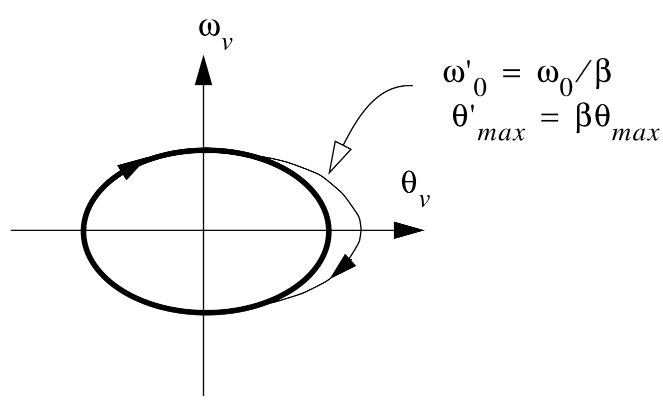


Figure 5.10 Phase diagram for a change of direction.
The departure from the nominal phase ellipse results in a chnage of direction to the left in this case.

$\Delta\theta_{dir}$ is a function of ω_0 and θ_{max}. Rewriting equation 5.24 to reflect this gives

$$\Delta\theta_{dir}(\beta\theta_{max}, \omega_0/\beta) = \Delta\theta_{dir}(\theta_{max}, \omega_0) + \Delta\theta_h. \tag{5.25}$$

The only unknown variable in this equation is β, which specifies how to modify the turn such that the appropriate change in direction is made.

Obtaining the function $\Delta\theta_{dir}(\theta_{max}, \omega_0)$ in an analytical form is difficult. Using first-order approximations for *sin* and *cos*, a simplified expression for the curvature, C_f, can be obtained from equation 5.18. It can be verified in Figure 5.6 that the curvature does indeed approximate a sine function, as indicated by equation 5.26. $C_f(t)$ can be integrated to obtain $\theta_{dir}(t)$, as shown in equation 5.27. The velocity of the foot of the model, V_f, is assumed to be constant in order to be able to evaluate this integral.

$$C_f = \frac{\theta_{max} sin(\omega_0 t)}{V_{cm}^2}\left(g + \frac{\omega_0^2}{lm}(I_v + l^2 m)\right) \tag{5.26}$$

$$\theta_{dir}(t) = V_f \int C_f(t)\, dt \tag{5.27}$$

We can now solve equation 5.25 directly for β, giving the analytical expression that we seek:

$$\beta = \sqrt{1 + \frac{\theta_h V_f \omega_0}{2g\theta_{max}}}. \tag{5.28}$$

By inspection, we can see that to change direction to the left during a left-hand turn $(\Delta\theta_h > 0)$, a value of $\beta > 1$ is required. This is the intuitive result expected, namely that a longer $(\omega'_0 = \omega_0 / \beta)$ and sharper $(\theta'_{max} = \beta\theta_{max})$ turn is required. To change direction during a right-hand turn, β should be calculated using

$$\beta = \sqrt{1 - \frac{\theta_h V_f \omega_0}{2g\theta_{max}}}\ . \tag{5.29}$$

During sharp turns, defined by $\omega_0 > 2\pi / 2.0$ or $\theta_{max} > 10°$, the approximations made during the above derivation cease to hold, and it is necessary to build a two-dimensional table storing $\Delta\theta_{dir}$ for various values of ω_0 and θ_{max}. β is then calculated by solving equation 5.25 by Newton-Raphson iteration.

In applying the directional control, a modified turn is calculated every time the state of the turning figure passes through the 0° or 180° points. The phase-ellipse control then uses the phase ellipse of the modified turn as being the target phase ellipse. Because of our assumption that the phase ellipses before and after the modified turn are the same, large instantaneous changes to the ω_0 and θ_{max} parameters should be avoided.

Some examples of results obtained using the parameterized control are shown in Figure 5.11. Track 1 shows a set of figure-8 tracks created by two figures turning out of phase and is similar to those seen in skiing. Track 2 shows turns of gradually decreasing frequency. Track 3 shows turns of increasing sharpness. Lastly, track 4 shows the direction being controlled. It should be noted that the directional control is important for tracks 2 and 3 as well, because the process of moving onto a new phase ellipse would normally result in an unwanted change of direction. Figure 5.12 shows an animation sequence with a snowboarder mapped onto the physical model. A videotape of our animation results is also available [96].

.

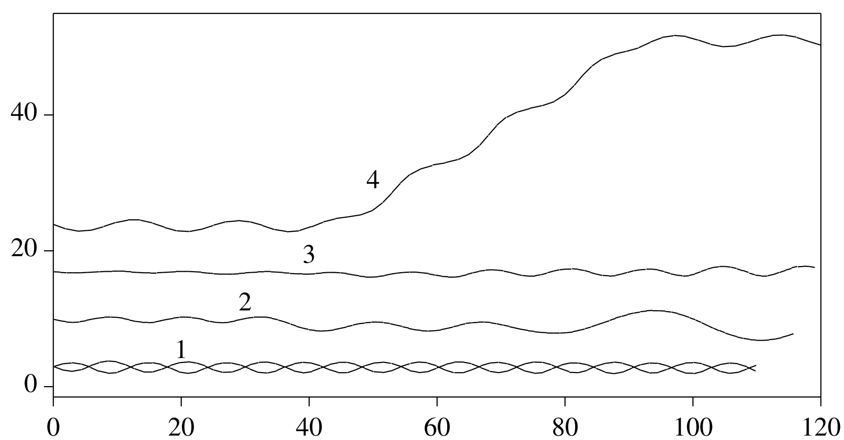


Figure 5.11 Tracks for turning using parameterized control.
Track 1 shows two tracks 180 degrees out of phase with each other.
Track 2 shows a track of decreasing frequency. Track 3 shows a track of increasing turn sharpness. Track 4 is the result of changes in direction.

Figure 5.12 Parameterized control with display model.

The parameterized controller is computationally inexpensive and can be used in real time. The bounds placed on C_f, the foot curvature of the turning figure will depend largely on the activity being simulated. A bicycle front wheel can be made to change direction very quickly and turn sharply as well. A ski or a surfboard, however, can neither change direction as quickly, nor turn as sharply.

5.6 Path-Following Control

An alternative method of control is to have the animator draw a path for the turning body to follow. It is impossible, however, for a turning body to follow an arbitrary path. If the foot is made to exactly follow a drawn path, this fixes the curvature, which leaves no freedom to control the body balance.

Our solution is to treat the motion synthesis as a constrained finite-time optimization problem. The optimization function to be minimized is the sum of the distances from points on the track of the turning body to the closest point on the desired path. The optimization problem is solved using a branch-and-bound tree search. During the branch-and-bound search, any branches of the search tree resulting in collisions with obstacles are also eliminated, meaning that obstacle avoidance constraints can be satisfied if this is desired.

The problem to solve is specified by the animator in a graphical fashion, as shown in Figure 5.13. The desired path and any obstacles are drawn using an interactive editor. We choose to use a piecewise cubic Hermite spline to draw the desired path, but a series of connected straight-line segments could be used equally well. The green vectors in the figure represent the tangent vectors at the endpoints of each spline segment, assuming continuity of both the endpoints and the tangents at the endpoints. All the red objects represent obstacles. A minimum distance can be specified for avoiding obstacles. The speed can be made to vary as a function of the terrain if desired. The desired path influences the solution indirectly through the optimization function.

Before describing the optimization algorithm in detail, we shall describe how the optimization function is evaluated. Let d_i be the shortest distance from the position of the foot at time step i to the spline path. Instead of explicitly solving for the shortest distance, we approximate it by sampling the spline segments at regular intervals and using the minimum distance to one of the samples. Thus we have $d_i = min_j \| P_i - S_j \|_2$, where P_i is the position of the foot at time step i and S_j is point sample j of the spline segments.

Figure 5.13 User specification of path to follow.
In this case a bicyclist must perform a high-speed turn through an intersection. The desired path to follow is given by the blue spline curve, while obstacles are shown in red. Obstacles have been placed across inappropriate exits of the intersection to speed up the solution process.

The value of the optimization function is determined by sampling the position of the track of the turning body and summing over all n such samples, namely $f_{opt} = \sum_{i=1}^{n} d_i$. Figure 5.14 illustrates the points and distances involved. Typically 12 sample points per spline segment and a time-step of 0.1 seconds for sampling the track has produced good results.

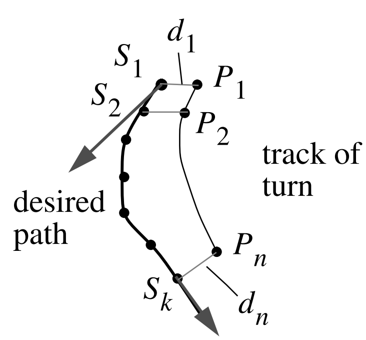


Figure 5.14 Evaluation of the optimization function.
The sum of the distances of the foot to the closest point on the desired path, sampled over a finite time interval, is the optimization function to be minimized.

5.6.1 The optimization algorithm

For the parametric controller discussed in §5.5, we calculate α_v and use equation 5.18 to calculate C_f. For our path-following controller, however, we shall be determining C_f directly. The algorithm uses a branch-and-bound search to explore how C_f should be controlled by planning ahead for a fixed interval of time. This fixed interval of time (typically 3 seconds) is divided into N discrete stages, where the control variable is maintained constant during each stage. We shall use dC_f/dt as our control variable because we wish to have C_f be C^0 continuous, implying that the steering foot cannot change direction abrubtly.

The branch-and-bound technique determines the optimal values of C_f to be used during the next N stages. However, only the value for the first stage is actually used. When the end of the first stage is reached, the branch-and-bound process is repeated. In this way each control decision is based upon a plan looking N stages into the future.

The branch-and-bound search works as follows. In the first stage, the effects of several values of dC_f/dt are simulated for t_{stage} seconds, beginning at the current state. In the second stage, the same process is repeated for each of the states produced by the first stage. This process repeats until the desired maximum number of stages have been evaluated. Let N be the maximum number of stages. Without bounding, the branching process will result in q^N possible paths and ending states, where q is the number of values of dC_f/dt tried at each stage. Fortunately, we shall see that many branches can be clipped or bounded at an early stage.

Let the starting state and ending states of stages be called nodes. Each node contains the information shown in Table 5.1. The pseudocode for the branch-and-bound algorithm as it pertains to the path-following problem is shown in Figure 5.15. The algorithm is used once every t_{stage} seconds to calculate the required control for the next stage. The nodes are kept in an evaluation queue sorted by their f_{opt} values. The algorithm always expands the node having the minimum value of f_{opt} first. Its children are placed in the node queue if they are not clipped.

Nodes are clipped if their f_{opt} value exceeds that of the current upper bound, found by keeping the current best (lowest) f_{opt} value of all completed paths.

Table 5.1 Node information

minseg	spline segment to begin search for shortest-distance points
x,y	coordinates of the foot
θ_{dir}	direction
θ_v, ω_v	state of lean
f_{opt}	value of optimization function
stage	number of stage
C_f	foot curvature
dC_f/dt	value taken at stage 1

Nodes can also be clipped as a result of a collision with an obstacle or the body falling over (a bound is placed on θ_v for this purpose). Obstacle avoidance is obtained by checking the track segments for collisions as they are produced. A collision is said to have taken place if a produced track segment intersects any obstacles.

Typical values used for the algorithm are as follows: $t_{stage} = 0.3s$, N = 10 stages, θ_v bound = 30°, and q = 5. The clipping is generally effective in restricting the number of nodes that need to be searched.

The algorithm uses various values of dC_f/dt to expand the nodes in the tree and thus produces an optimal value of dC_f/dt to use. Using the derivative of the curvature allows for C^0 continuity of the curvature and will thus produce smooth motions. In generating values of dC_f/dt to apply, bounds are placed on dC_f/dt and C_f to ensure realism.

```
enqueue (current state)
while (nodes in queue)
        dequeue (node)
        calculate inputs to be applied
        for (each input)
                end_node = simulate (node,input,t_stage )
                if (hit obstacle) continue
                if (fell over) continue
                if (exceeded bound) continue
                if (at last stage)
                        update bound if necessary
                        continue
                enqueue (end_node)
return dC_f/dt taken at stage 1
```

Figure 5.15 Pseudocode for branch-and-bound algorithm.
Branches in the search tree are bounded by simulation events, such as falling over or hitting an obstacle, or by paths exceeding the present upper bound on the solution.

5.6.2 Calculating the phase

The current phase, ϕ, of a motion is very useful in creating proper motions for the display model. Because the process of determining the phase requires information about the future motion of the turning figure, we produce the phase in a post-processing pass after the optimal path has been determined. Using an estimate of ω_0 and θ_{max} as defined for the parametric controller, we can calculate the phase as before.

Estimates of ω_0 and θ_{max} can be obtained by noting the points at which the state of the turning body enters and exits the current phase quadrant. This is illustrated in Figure 5.16. The magnitude of the intercept along the θ_v axis serves as an estimate of θ_{max}. Similarly, the magnitude of the intercept along the ω_v axis serves as an estimate of $\theta_{max}\omega_0$, which allows us to estimate ω_0.

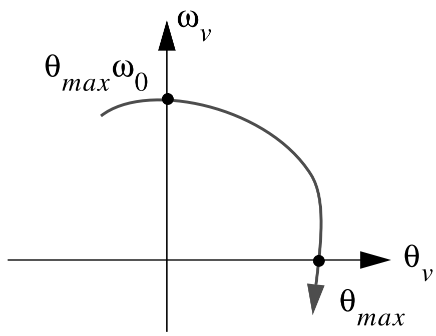


Figure 5.16 Estimating phase information.
A phase ellipse is estimated by using the points at which the axes of the phase diagram are crossed.

5.6.3 Results

Various examples of the use of the path-following algorithm are shown in the following figures. Figure 5.17 shows the results of an automatically-synthesized tree-skiing path. The solid line is the spline path to be followed. The dotted path is calculated using N=10 stages. The dashed line shows the result for N=12 stages. In this case planning ahead an extra two stages considerably changes the motion by choosing a different route through the trees.

The example in Figure 5.18 shows how the path-following algorithm can be used to solve for a path through a slalom course. The poles are drawn as long obstacles in order to restrict passage to along one side only. Some additional obstacles are added to speed the calculation of the solution. These obstacles prevent the search tree from expanding into areas for which it can be predicted that they will not contain the optimal path. The solid line is given as the path to follow, and the dotted line is the calculated solution. Figure 5.19 shows an animation sequence of a skier mapped onto a path-following solution for a slalom course. Figure 5.20 shows a telemark skier passing through a portion of the same slalom course and making use of the same solution.

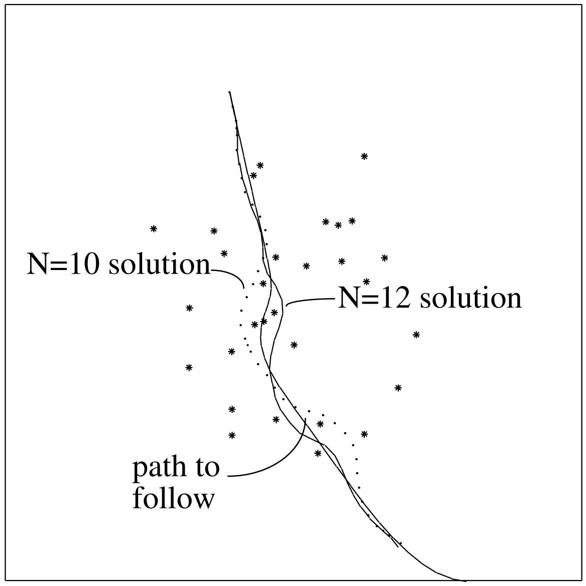


Figure 5.17 Tree skiing.
The solutions are dependent on the length of the chosen optimization interval.

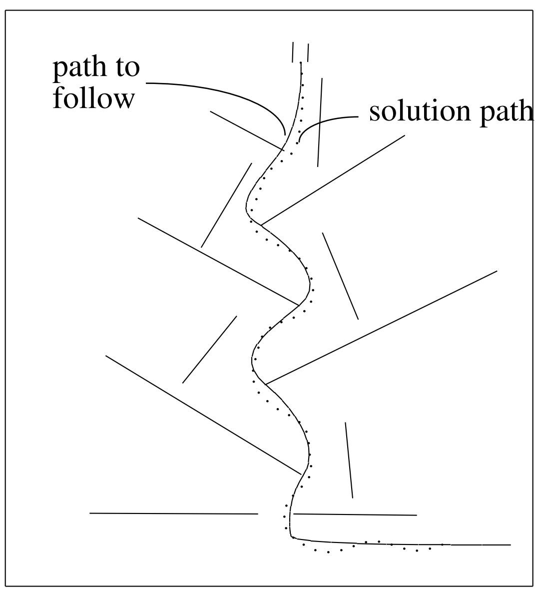


Figure 5.18 The slalom race.
The slalom poles are modelled as obstacles extending on the side where the skier should not pass. Further obstacles are added to reduce the amount of time spent searching areas which will likely not contain the final solution path.

Figure 5.19 The slalom race with an alpine skier display model.

Figure 5.20 The slalom race with a telemark skier display model.

An example of a bicyclist going around a corner is shown in Figure 5.21. The edges of the road are drawn as obstacles. Additional obstacles are drawn blocking the bottom and left exits from the intersection in order to restrict the size of the search tree. Figure 5.22 shows a bicyclist weaving through a series of potholes on a road. Figure 5.23 shows an animation sequence from the high-speed bicycle turn shown in Figure 5.21.

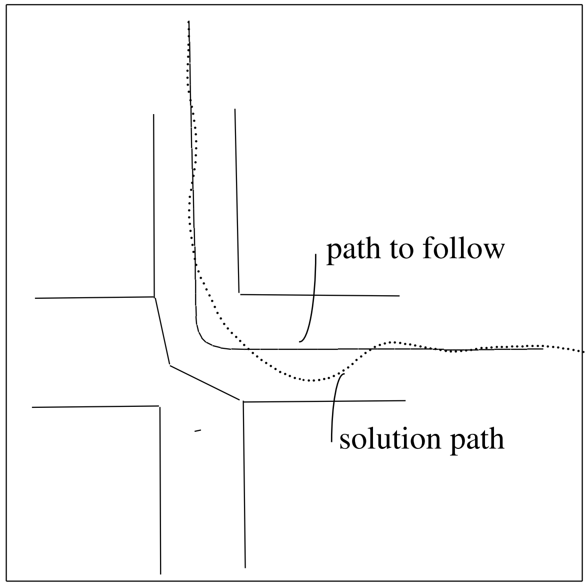


Figure 5.21 Bicycling around a corner.
The oscillations occuring before and after the turn are used to establish the lean into the turn and then recover from the turn.

The algorithm could also be used to find a path in the case where no desired path to foillow is given. In this case another optimization criterion would need to be specified. It is unclear whether other optimization metrics would exhibit the same properties for bounding solutions that the distance metric does.

Figure 5.22 Bicycling while avoiding potholes.

Figure 5.23 Display for bicycling around a corner.

5.7 Mapping a Display Model on to the Physical Model

Many body parts are not directly represented in the physical model. The realistic motion of many of these would be difficult to synthesize using a physical model because of the lack of obvious criteria for their synthesis. The motion of these parts is nevertheless coordinated with the main motion of the body. We shall make use of variables associated with the simplified physical model to drive these motions. This represents a type of *motion amplification* by defining the motion of all body parts in terms of the motion of the simplified physical model. The most useful variable in this regard is ϕ, the current phase of the turn. The force exerted on the leg of the physical model and several other variables are also used in the mapping.

Figure 5.24 shows at what point the mapping process takes place. The input to the mapping process consists of the variables shown in Table 5.2. The creation of a suitable mapping procedure typically involves some experimentation, especially if biomechanical data from real motions are not readily availableWe shall now briefly describe how the mapping was performed for several figures. .

Alpine skiers have some degree of up-and-down motion during turns. This is achieved in the display model by making the legs act as a spring. The force on the leg of our simplified model varies approximately ±30% from the static force during typical turns, as shown in Figure 5.6. A suitable spring constant k_s can be chosen based upon the amount of up-and-down motion that is desired. Given the current force in the leg and k_s, the leg length can be determined. This in turn defines the bend of the knees. The bend at the hips is directly correlated to the bend at

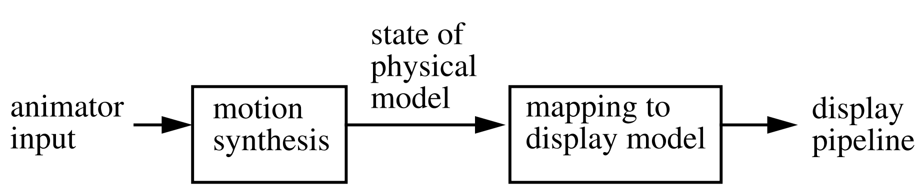


Figure 5.24 The animation pipeline.
The motions are first synthesized using the simplified physical model. The motion of the display model is then calculated by coordinating its motion with that of the physical model. The display mapping defines how this coordination takes place.

Table 5.2 Input parameters for the mapping process.

x_f, y_f	foot position
θ_{dir}	current direction
θ_v	lean-angle of body
C_f	foot curvature
F	force exerted on leg
ϕ	phase angle

the knees to portray a coordinated motion. A linear relationship is assumed between the curvature of the ski and the angle at which the ski is tilted. The constant to use is easily estimated from a plot like the one in Figure 5.6. The tilt of the outside, weight-bearing ski is made to be slightly larger than that of the inside ski. When the tilt angle of the body, θ_v, and that of the ski do not correspond, we need a way of tilting the ski on edge with respect to the body. This is accomplished by twisting the weight-bearing leg to the inside. The pole planting motions of the arms are coordinated with the phase so that the pole plant occurs before the actual turn, at $\phi = 30°$ and $210°$. A twisting motion of the upper body is made to be in phase with C_f.

The snowboarder is similar in many respects to the skier. The bends at the knees and waist are calculated based upon the leg force, as with the skier. The upper body is made to twist in phase with C_f. Tilting the snowboard with respect to the body is easily accomplished by bending at the ankles. The arms are moved to lead the twisting motion of the upper body slightly.

The bicyclist requires only a simple calculation to determine how the front wheel should be turned to achieve the given curvature C_f. Inverse kinematics is used to keep the hands on the handlebars. The pedals move as a function of the distance travelled. Inverse kinematics is used to keep the feet on the pedals.

Turning for a telemark skier is achieved by separating the skis and placing them at an angle with respect to each other. Inverse kinematics defines the position of the legs. The remaining motions are similar to that of the skier.

5.8 Discussion

The path-following algorithm provides a method of specifying the motion in terms of the desired path of the foot. This provides a complementary solution to the parameterized solution, in which motion was specified using parameters related to the body motion. These two methods of control, namely a parameterized controller and an optimization-based controller, each have their uses. The trade-offs between these two methods of control as seen here are probably typical of their use in general.

The motion amplification method of using a simple dynamic model to drive a more complex display model represents a type of hierarchical control. One could further use the final kinematic positions produced by the display mapping as the set-points for controllers in a more detailed and complete physical model. We have not yet experimented with such a scheme, but we believe it is feasible. It is not entirely clear what additional gains in the quality of the motion would be attained in return for the additional complexity of control and simulation. One of the general limitations of the kinematic display model mapping is that the motion can seem too perfect. This would be especially evident in situations involving interaction with other objects or phenomena in the environment.

Many subtle effects that occur in various turning sports are not included in our model. We have described the basic physics that these sports have in common and a common method of control. Further detailed analysis of turning motions is possible, but the results then become specific to the given sport. This is also another source of difficulty when considering the possibility of proceeding to a more detailed and complete physical model. Simulating the transmission of forces from the snow to a skier through a flexing, twisting ski is a difficult simulation problem. A full dynamic simulation of a bicyclist is probably easier to perform, but will not yield as many benefits over the kinematic display mapping. More complete dynamic simulation would yield more realistic motions for turning figures moving quickly over bumpy and variable terrain. Such terrain would require controllers with some additional complexity over the ones presented here, however.

Modelling and control of turning motions across non-planar terrain is a formidable challenge. Skiers and snowboarders can make good use of changes in terrain to facilitate turning. The problem here is one similar to the path-following problem in that any decision must be based not only upon the current state of the figure, but also the state of the environment. Experienced practitioners of sports can rapidly make generalizations about how terrain will affect a planned motion. It is difficult as of yet, however, to efficiently synthesize and store these generalizations.

The branch-and-bound method used to perform the optimization might be useful in general as a control-synthesis technique. It is especially useful for systems with a small number of controllable inputs and a large number of degrees of freedom. In these cases it is more convenient to explore possible motions in the 'input space' of the system rather than its state space.

In the future, it will be useful to develop ways of automating the generation of parameterized controllers. Turning is such a common and reasonably-simple motion that it is worthwhile to analytically determine a parameterized controller, as we have done here. In general, however, the synthesis of such a controller is still a laborious task that is preferably automated.

The approach of controlling a simplified physical model and then using kinematics to 'amplify' its motions is not universally applicable to motions. For many objects it is difficult to arrive at a suitable simplified model that is easy to control. Furthermore, arriving at a simplified physical model and determining a suitable controller should preferably be automated and invisible to the animator. This is a difficult task.

The control techniques as presented in this chapter are nevertheless a useful tool for animation. By specifying the appropriate turning parameters or drawing a desired path to follow, an animator can achieve realistic turning motions in a flexible way with relatively little effort. The process of mapping a display onto a simpler physical model is effective at capturing the coordination of movement exhibited in turning motions.

Chapter 6 Sensor-Actuator Networks

People and animals are capable of moving in many different ways. In this chapter we shall examine how we can automate the discovery of various modes of locomotion for figures we wish to animate, given only a physical and geometrical representation of the figure. We wish to be able to control more complex systems (creatures) than those of Chapter 3, without having to explicitly specify the control decomposition in advance, as with our work on walking and turning. The controllers we propose in this chapter are called *sensor-actuator networks* (SANs). A SAN is a network of weighted connections between a set of simple binary sensors and the actuators (i.e., the motors or muscles).

Using the method presented here, it is possible to find answers to the question “here is a creature; how can it move?”. The user supplies the configuration of a mechanical system that has been augmented with simple sensors and actuators. A two-step stochastic procedure for finding and then improving suitable SANs is given. Ten different creatures controlled by this method are presented.

There are two main novel features of SAN controllers. First, they operate entirely using very limited, but useful sensory data. Control solutions are often cast into a form where sensory information is processed to produce an estimate of the system state. The state estimate is then used as the basis for further calculations to determine the desired control. SANs have no notion of the state of the system as conventionally defined. Second, the stochastic synthesis procedures used to create SANs are unique and powerful tools.

A principal advantage of the method to be introduced in this chapter is that it requires less knowledge to use than other control methods for physically-based animation. The user provides the construction of a creature and can then ask the question "how would it move?" Our method can then yield several controllers providing feasible modes of locomotion, although there is no *guarantee* that a suitable controller will be found[1]. This capability encourages experimentation with many different creatures. We have been surprised with many of the modes of locomotion discovered. Even the simplest objects are capable of a large repertoire of interesting motions. Many of the controllers generated using our method can take advantage of phenomena such as sliding friction or collision impact, which are easy to simulate but hard to incorporate in many optimization methods. Figure 6.1 shows modes of locomotion discovered for 4 different creatures.

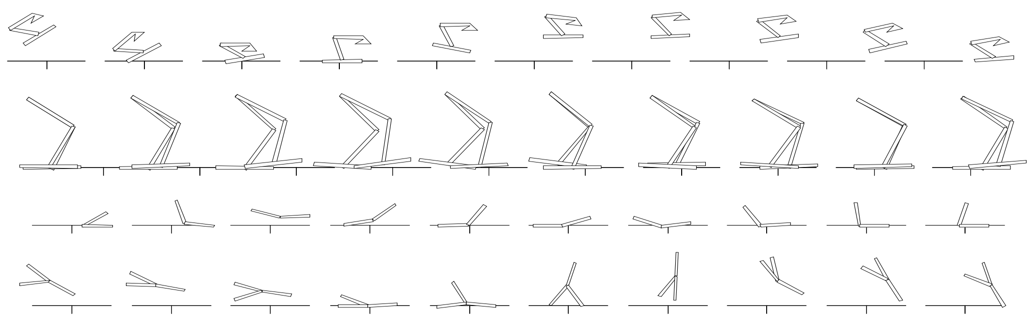


Figure 6.1 Modes of locomotion using sensor-actuator networks.
All the motions shown here continue indefinitely in a periodic fashion.

SANs are useful for control problems where the systems are non-linear, non-smooth, and not statically stable. A large number of crawling, jumping, hopping, flipping, and walking creatures fall in this category. Our sensor-actuator networks (SANs) prove to be particularly adept at controlling creatures in this class. It is difficult to perform optimizations for such systems that have discontinuities. However, it is not difficult to simulate many systems of this class. We shall use this to our advantage by performing repeated trials to determine a suitable controller. This means that a minimal knowledge of the physics of the mechanical system is required. We carry out our trials using physical simulations, although they could equally well be carried out directly on the real objects in principle.

1. This is mentioned as an observation and not as a point of comparison. No other technique can offer such a guarantee either.

A sensor-actuator network is a non-linear network of weighted connections between a small number of binary sensors and the actuators (the motors or muscles of our creatures). The network has internal delays, thereby giving it time-varying properties. In this chapter we provide a method for determining the parameters associated with SAN networks in order to obtain instances of a desired behaviour. We furthermore show that SANs can be used to control a large variety of interesting creatures.

The next section relates our work to influential previous work. Section 6.2 discusses how to construct a creature. Section 6.3 presents the architecture of SANs. Section 6.4 gives the algorithms for synthesizing and fine-tuning SANs. Section 6.5 presents a variety of results, and section 6.6 concludes.

6.1 Related Work

Developing control solutions for arbitrary dynamical systems is a difficult problem. Consequently, control solutions are usually specific to a constrained class of control problems. Examples of this are the walking and turning controllers we have introduced in Chapters 4 and 5. More general solutions such as the method of Chapter 3 often have problems with any discontinuities that can appear in the motion. For the creatures we wish to consider in this chapter, such discontinuities will be commonplace. Examples of this include feet hitting the ground and joints reaching the end of their swing.

The structure of the controller we propose is similar to those proposed in the work of Wilhelms and Skinner [106] and Braitenberg [13]. Both discuss controllers constructed using weighted connections between sensors and actuators (or *effectors*), possibly through intermediate nodes of some kind. [13] presents a series of thought experiments showing that such networks are capable of producing complex and seemingly intelligent behaviour. This kind of close coupling between sensing and reaction has also been postulated to exist in some biological systems [40]. The system in [106] allows the user to interactively construct the mapping between sensors and actuators for a creature consisting of a rigid body that can propel itself in three dimensions with a jet. Several different kinds of nodes are suggested for use in the connection network. Examples of attraction and avoidance behaviours can be constructed using their approach. Our work will show that SANs, which are similar networks, can be *automatically* synthesized and can be used to control a large variety of physically realizable creatures. Furthermore, we show it is only necessary to consider *one* kind of network node.

SANs are similar in topology to many artificial neural networks (ANNs), but are different in several respects. Non-recurrent ANNs have no internal delays and thus provide a static input-output mapping. Furthermore, the synthesis method we employ does not use the derivative-based learning methods.

SANs are loosely related to the work presented in [14] on subsumption architectures for the control of mobile robots. The subsumption architecture is used to implement control at various levels in a control hierarchy. It has also been shown that a simple architecture whose sensing and action are closely coupled can be seen to act in the pursuit of a goal [67]. A system model of an insect nervous system that produces interesting behaviour is presented in [10]. While these results show that interesting behaviour can be produced from a network of simple elements, they do not provide for a way to automatically synthesize locomotion.

Our synthesis technique embodies a generate-and-test philosophy that can also be found in some other recent work. Ngo and Marks [78] use it to achieve similar goals to ours using a genetic algorithm. In [68] a trial-and-error method using statistical measures is shown to learn how to coordinate leg motions for an insect robot. A user-controlled generate-and-test procedure to produce striking images is presented in [89]. A discussion of search-based control methodology is given in [77]. The work of [65] presents a genetic algorithm for 'evolving' circuits that oscillate, and then couples oscillating circuits to control a walk for a six-legged, statically-stable robot.

Work on passive walking [69] illustrates the phenomenon of mechanical systems reaching and maintaining stable limit cycles. Similar limit cycles have also been found in the motions of hopping robots [54]. Such *attractors* or limit cycles have also been studied in neural networks and have been conjectured as being the basis for behavioural action in nervous systems [57] [88]. In our work, the dynamical system consisting of the SAN and the creature has a propensity for such limit cycles, in which case it results in periodic motions, some of which are the useful gaits we are looking for.

6.2 Constructing a Creature

The details of our method are best described through the use of an example. We shall use the *bounder* creature, shown in Figure 6.2. The construction diagrams of the other creatures we consider can be found in Appendix B. The bounder consists of 5 links, 4 angular actuators, and 8 binary sensors. SANs take binary sensor values as inputs and produce a set of desired angles or lengths as outputs. These desired values are used by proportional-derivative (PD) controllers to determine the internal torques or forces to apply. A PD controller is functionally equivalent to a spring and damper between the desired position and current position of a link. We shall now discuss each of the components of a creature in more detail and then discuss our simulator.

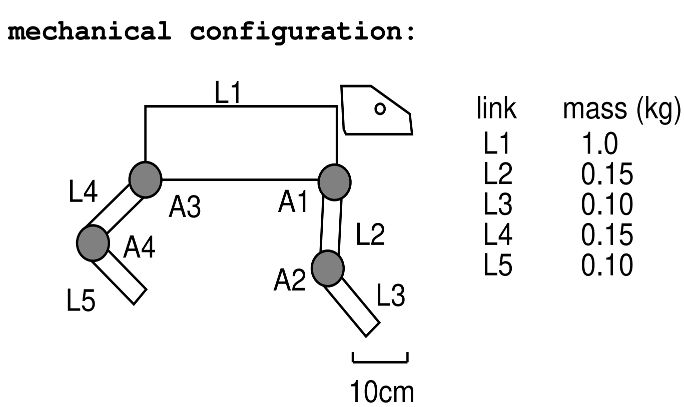


Figure 6.2 The bounder creature.
This figure shows the information that must be specified when designing a creature.

6.2.1 Mechanical Configuration

The user begins by specifying the desired mechanical configuration for the creature. All our creatures are built of rigid links and have planar dynamics, most of them operating in a vertical plane (i.e., under the influence of gravity). This is a limitation of our current implementation rather than the technique in general. As we shall soon see, planar creatures can have a very large repertoire of interesting behaviours. Specifying the physical structure of the creature requires specifying the mass, moment of inertia, and shape of each link, as well as how they are connected with joints.

6.2.2 Sensors

All sensors for our creatures are binary. If a sensor is 'on,' it produces a value of 1; otherwise it produces a 0. We currently use the four kinds of sensors shown in Figure 6.3. Touch sensors (e.g., S1 and S2 for the bounder) turn on when in contact with the ground and otherwise remain off. Angle sensors (e.g., S3) determine if the angle of a limb is within the fixed range determined by the minimum and maximum angle specified for the sensor. This angle is measured relative to the link it is attached to, and the zero position is indicated with a dashed line in the creature diagrams. Eye sensors turn on if the *follow point* is in their cone of view. The follow point is a point to which some of our creatures will be attracted. One can thus control these creatures by dragging the follow point in front of them along a desired path. The cone of view for an eye is defined by a minimum and maximum angle. The zero-degree reference for eyes is shown in the creature diagrams with a dashed line. Length sensors are similar to angle sensors, but measure linear distances.

Sensors are added to a creature after the mechanical configuration has been designed. It is necessary to use some intuition and creativity in deciding what sensors will be useful to a creature. Touch sensors are useful for points expected to be in contact with the ground. Angle sensors are useful for detecting when a limb has reached the end of its swing. Eye sensors are necessary to be able to track a follow point. Length sensors are useful in the control of linear

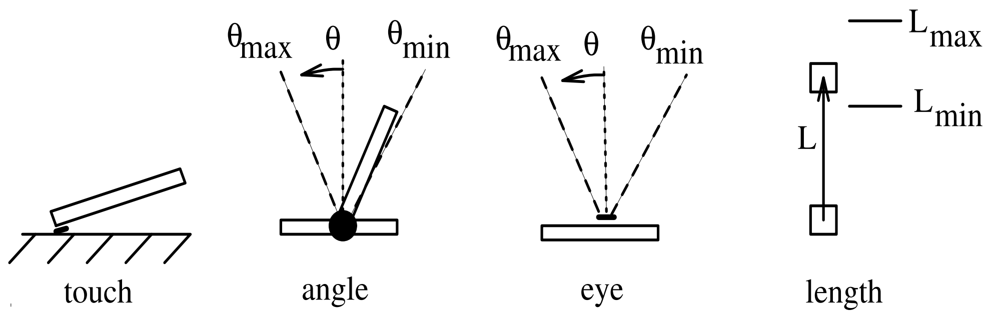


Figure 6.3 Sensor types.
All four sensor types return a binary value. See the text for a further description of their operation.

actuators. The total number of sensors should also be considered. Too many or too few sensors can introduce problems for the network synthesis technique to be outlined shortly.

6.2.3 Actuators

We shall use two kinds of actuators in our creatures: *linear* and *angular*. These are shown in Figure 6.4. Specifying an angular actuator requires specifying the joint concerned, the upper and lower limits that the desired angle can take (measured in degrees in our creature diagrams), and specifying the *strength* of the actuator.

A length actuator exerts linear forces between a pair of points. It is able to pull its endpoints together or also push them apart. Our crawler creature (see Appendix B) will make use of these). A linear actuator requires the specification of the points of attachment, the minimum and maximum allowable desired length, and the strength of the actuator.

The strength of an actuator is determined by two constants associated with its corresponding PD controller: k_s and k_d . k_s represents a spring constant and k_d represents a damping constant. An angular actuator generates an internal joint torque given by $T = k_s(\theta_d - \theta) - k_d\dot{\theta}$, while a linear actuator generates a force $F = k_s(l_d - l) - k_d\dot{l}$. For many motions, choosing $k_s = 10k_d$ provides for a suitably energetic system with some damping. The value of k_s can often be chosen by doing a simple calculation. For example, a 10 degree deflection of limb L5 of the bounder should lead to a large enough force to support half the body weight when the leg is bent. A simple calculation will produce k_s= 0.4 Nm/deg, which is used for all the actuators in the bounder. No hard joint limits are provided by the actuators, although they could of course be implemented in the simulation if desired.

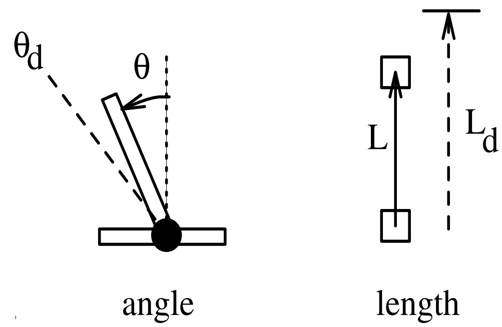


Figure 6.4 Actuator types.
Actuators are proportional-derivative controllers, exerting a torque or force equivalent to a spring and damper being placed between the current position of a part and the desired position of that part.

6.2.4 Dynamics Simulation

Generating and using SANs requires only the capability to simulate the dynamics of a system. It is beneficial to have a fast dynamics simulator, as our method for generating suitable SANs requires many simulation trials. We shall now briefly describe how our dynamics simulator works. It is the same simulator we use for our walking human figure of Chapter 4 and in some previous work [93]. The important characteristics of the dynamics simulator are highlighted again here for clarity. [30] is suggested as a good general reference on dynamics simulation.

We make use of a *dynamics compiler* that uses the mechanical configuration information to generate a 'C' procedure that solves the equations of motion for a single time step. The procedure generates and solves a set of linear equations Ax = b, where x is the set of unknown accelerations. The elements in A and b are functions of the system state, the physical parameters of the system, the internal torques, and external forces. A recursive Newton-Euler formulation is used. This is $O(n^3)$ in the number of links and is quite suitable for $n < 10$.

The creatures are treated as free bodies in space. The external forces applied by the ground are calculated using stiff spring and dampers. We favour this approach as being simpler and more flexible than the alternative choice of reformulating the equations of motion upon impact of a link with the ground. Physical phenomena such as wind and water forces are added as external forces. For example, for our fish creature, we calculate the net water force for each link as $F = k\int_A (v \bullet N)\, dA$, where $v \bullet N$ defines the component of the velocity in the direction of the surface normal, and A is the total surface area of the link.

The coefficient of friction of the ground can be set to a desired value. Friction modelling is important because most of the objects we shall deal with must make use of the static or dynamic frictional forces occurring between the object and the ground in order to move forward. Figure 6.5 shows how the friction model works. A spring and damper is attached between the point on the model contacting the ground, P_c, and the initial point of penetration, P_1. The force exerted by this spring and damper is decomposed in a tangential force F_t and a normal force F_n. If $F_t/F_n > \mu_s$ then slippage is deemed to occur, where μ_s is the static coefficient of friction. When slippage occurs, the point of initial penetration is reset to lie directly above P_c, shown as point P_2 in Figure 6.5. This is not a comprehensive model of friction, but it is sufficient for our goal of avoiding 'sticky' contact points which can only release by lifting off the ground. If the point P_c is moved to the left at a constant speed, the horizontal force exerted by our friction model will have the shape shown in Figure 6.6. From this figure it is evident that the we have $\mu_d \approx 0.5\mu_s$ for our friction model, where μ_d is the dynamic coefficient of friction.

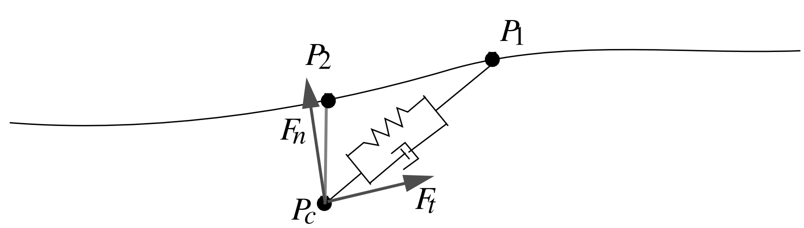


Figure 6.5 Friction model.
All ground contact is modelled using springs and dampers which are attached to specific points on the creature when these sink below the ground.

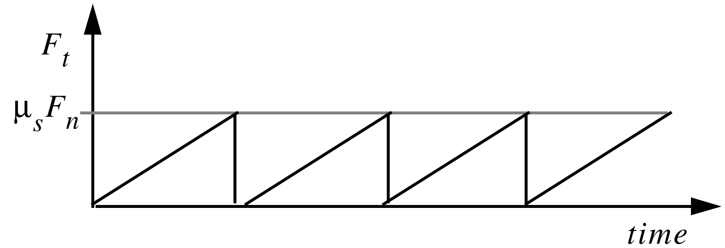


Figure 6.6 Tangential force for friction model.
The friction model simulates dynamic friction by allowing slippage at discrete points in time.

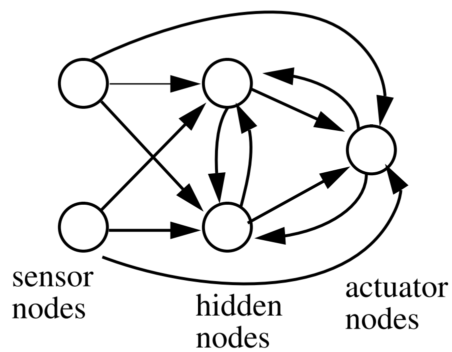


Figure 6.7 Topology of a sensor-actuator network.
All nodes are fully interconnected, except for the sensor nodes, which take on the values of the associated sensors.

6.3 Sensor-Actuator Networks

SANs provide control by connecting sensors to actuators through a network of weighted connections. We will discuss in the next section the important question of how to determine the weight values. This section describes the structure and operation of SANs. The particular structure that we have chosen is probably one of many such structures that could be equally effective. This is still an open topic for investigation.

A simple example of a SAN is shown in Figure 6.7. The network consists of nodes and unidirectional weighted connections. The weights of the connections can take on values in a fixed range. In our implementation we choose integers in the range [-2,2]. As shown, there are three kinds of nodes: sensor nodes, hidden nodes, and actuator nodes. The sensor nodes are fully connected to all hidden nodes and actuator nodes. All hidden nodes and actuator nodes are fully interconnected. As a heuristic, we choose the number of hidden nodes to be approximately equal

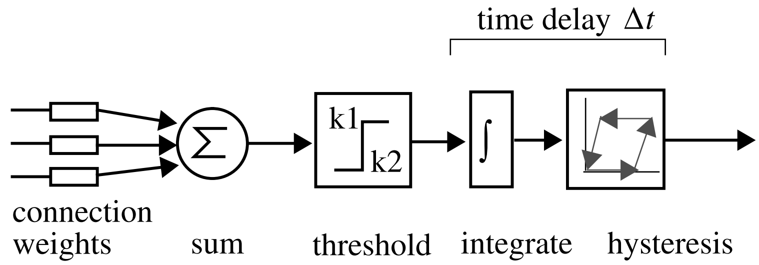


Figure 6.8 Function of a SAN node.
A node sums its weighted binary inputs and then uses a threshold to produce a binary output. The integrator and hysteresis function serve to implement a time delay.

to the number of sensor nodes. For example, the SANs for the bounder have 8 sensor nodes, 8 hidden nodes, and 4 actuator nodes. In our experience, too many or too few hidden nodes can make it difficult to establish oscillations of a suitable duration in the network.

The sensor nodes simply take on the values of their associated sensors. The hidden and actuator nodes function as shown in Figure 6.8. The basic function of a node is to sum the weighted inputs and to output a ‘1’ if the sum is positive. This is a function similar to those performed in neural networks. In order to obtain useful controllers, however, we have found it is useful for the controller to have its own time-varying properties. This is effected by having a time delay associated with the operation of each node. This delay is implemented with the integrator and the following hysteresis function. The constants k_1 and k_2 provide control over the time delay for a node to turn on and off, respectively. A hysteresis function is used instead of a simple threshold function to ensure that a time delay always occurs between successive changes in the output state.

Actuator nodes make direct use of the result of the summing operation at a node to determine the desired angle or length of the actuator. The sum is first bounded to lie in a fixed range and then linearly mapped to the length or angle range of the actuator. We bound the sum to the range [-2,2] in our implementation. This provides a choice of 5 possible outputs for each actuator.

The function of a node is both simple to program and to build directly in hardware. The code implementing the functionality of a node is shown in Figure 6.9. The variable 'istate' is the internal state of a node associated with the integrator. Its hardware implementation would require three resistors, a capacitor, and three opamps per node[1]. An example of a fully interconnected 3-node SAN is shown in Figure 6.10. It is also possible to construct a purely digital circuit implementation using an EPROM and a small amount of programmable logic, although this is not shown here.

```
sum = 0
for (each input i)
        sum = sum + input[i]*weight[i]
if (sum>0) then
        istate = istate + k1*dt
else
        istate = istate - k2*dt
if (istate>1.0) then istate = 1.0
if (istate<0.0) then istate = 0.0
if (output==1 and istate==0.0) then output=0
if (output==0 and istate==1.0) then output=1
```

Figure 6.9 Code corresponding to node function.

The resistors denoted Rb act as pull-up and pull-down resistors. The connection weights are implemented as pairs of resistors. The summation operation takes place as a sum of currents produced by the connection weights. The comparator performs the threshold function, followed by an RC integrator. The output of the RC circuit is buffered with a unity-gain amplifier and then used as the input to the last amplifier circuit, which implements the hysteresis function.

Because of the delays and the internal interconnections, the SAN has time-varying properties of its own that are important in generating useful control networks. This differs from neural networks, which conventionally perform a mapping from inputs to outputs without any delay or other time-varying properties. We view the motions that the SANs produce as being the result of two interacting dynamical systems, that of the SAN and that of the creature itself.

1. An opamp can be constructed using 10 transistors.

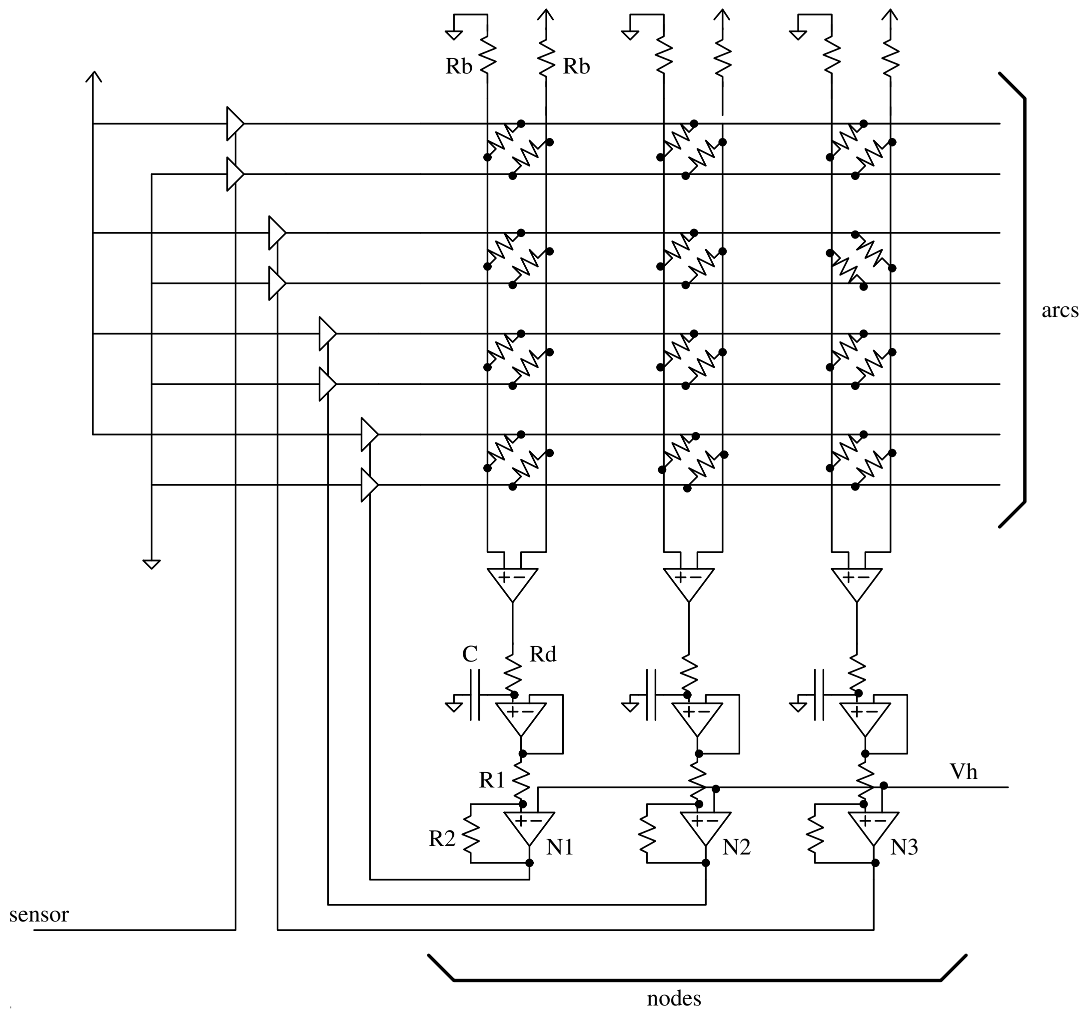


Figure 6.10 Hardware implementation of a 3-node sensor-actuator network. There are also many other possible implementations of the same functionality.

The constants k_1 and k_2 should be chosen in relation to the expected duration of a periodic locomotion cycle for the creature. Thus the node delays for an elephant should be much larger than those for a mouse. The delays are given by $T_{on} = 1/k_1$ and $T_{off} = 1/k_2$. Typically $T_{on} = T_{off} = 0.25 \times T_{cycle}$ yields good results, where T_{cycle} is the expected duration of one locomotion cycle. The delays used for all of our examples fall in the range of 0.07-0.15s.

6.4 Network Synthesis

We have now defined the construction of creatures and the structure of SANs, but we still need a method to obtain the weight values in the SAN. These will ultimately determine a creature's behaviour. Before outlining the approach we take to this problem, let us briefly consider another possible approach. Consider a top-down method whereby we assume we know the type of motion or desired gait and must now determine the details to generate it. Such an approach has several problems. First, it is often difficult to come up with the desired gait, especially if the creature has no counterpart in nature. Many of our creatures are capable of modes of locomotion that we never could have conceived of beforehand. Second, motion in the desired way might be physically impossible for the creature. We wish to restrict our search to strictly those solutions that are physically feasible. Lastly, the non-linear nature of the creature and its sensing devices makes any such top-down synthesis a very difficult task.

The top-down approach tries to determine a controller given a desired behaviour. The bottom-up approach that we shall take repeatedly generates and evaluates controllers (through simulations) until one or several are found that have desirable behaviours. We thereby take advantage of the fact that the simulation function is much less expensive than the top-down synthesis function. If we can choose controllers from an appropriate space of potential controllers, it becomes cheaper to find a suitable controller by repeatedly generating and evaluating controllers until a desired controller is found. We have found that the SAN architecture defines a controller space that is well populated with useful controllers.

Our network synthesis procedure consists of two phases. Phase 1 involves random SAN generation and evaluation. Phase 2 takes the best SAN controllers found in phase 1 and improves upon them. These two phases together define a coarse-to-fine strategy in searching for suitable controllers.

6.4.1 The Evaluation Metric

An evaluation metric is required for being able to determine the quality of a motion generated by any given SAN. For most of our creatures, this is simply the distance travelled in a fixed amount of time: $f_{eval} = |x(t_{final})|$. The creatures that travel the furthest are usually the ones that have an interesting mode of locomotion. We usually do not care whether our planar creatures move to the right or the left because both can yield interesting motions. The evaluation metric can be further qualified to be the distance travelled without falling over. To obtain a controller with energetic hops, the average height of the creature can be incorporated into the evaluation metric: $f_{eval} = \int_0^{t_{final}} (\dot{x} + ky^2)\, dt$. The $\dot{x}$ term measures the distance travelled while the ky^2 term measures the average height. For creatures that should make use of their eye sensors to track a follow-point, the evaluation metric is: $f_{eval} = \int_0^{t_{final}} (\dot{p} \bullet F)\, dt$, where p is the position of the object, $\dot{p}$ its velocity, and F the unit vector pointing towards the follow point. This metric is a simple measure of how well pursuit is maintained.

It is should be noted that the energy available to a creature is limited because the range of desired values for a joint is also limited. As a result, it is not necessary to include an energy expenditure term in the evaluation metric in order to prevent ever-increasing muscle activity.

6.4.2 Phase 1: Random Generation and Evaluation

A controller is generated at random by choosing all of its weights at random. This is roughly equivalent to randomly selecting the reactions to be associated with each sensor. A randomly generated SAN is then evaluated by simulating the behaviour of the creature with the SAN controller and calculating the evaluation metric.

The random search performed by phase 1 is used to discover different possible modes of locomotion for a creature. In order for the random generation to be an effective synthesis method, however, we must have some reasonable expectation of finding reasonable working controllers. At first glance, this does not seem a likely proposition. The space being randomly sampled is very large. For our bounder example it contains 5^{240} possible samples: each of the 240 weights can take any of 5 values. Furthermore, there is no guarantee that any point in this space corresponds to a useful controller.

It is our hypothesis that within the large search space there are many pockets containing useful weight combinations. While the SAN architecture clearly imposes a structure on the solutions, it is not obvious how to search the space of all possible solutions within this structure. We have found that our dart-throwing strategy is successful a small but significant proportion of the time. Typically 1-5% of the random SANs result in useful motions.

There are several features of the SANs that seem to be crucial to ending up with interesting and useful controllers. These features are the hidden nodes, the delay associated with the nodes, and the non-linearity of the nodes. When any of these features are removed, it most frequently results in a creature that rests in an immobile state after some initial motion. It is perhaps no coincidence that these are exactly the features that make the SANs capable of producing oscillating patterns.

Figure 6.11 shows a histogram of 1000 randomly generated controllers for the bounder. The evaluation metric is the distance travelled to the left or right in a six second period, given in metres. This run produced at least 6 visibly different modes of locomotion. It is unclear to us how to unambiguously categorize the resulting modes of locomotion because there are continuous variations possible between some of the gaits observed. All the controllers that resulted in the body of the bounder touching the ground at any point in time were given an evaluation metric of zero. Some modes of locomotion appeared several times. The histograms for all our other creatures were similar in shape: a large percentage of undesirable controllers and a long tail containing useful and interesting controllers.

6.4.3 Phase 2: Fine Tuning

A second fine-tuning phase may be applied to improve the controllers obtained as a result of the first phase. In many cases, it can be seen by looking at an animation of the creature that a controller is almost successful or could possibly be improved. The fine-tuning phase makes

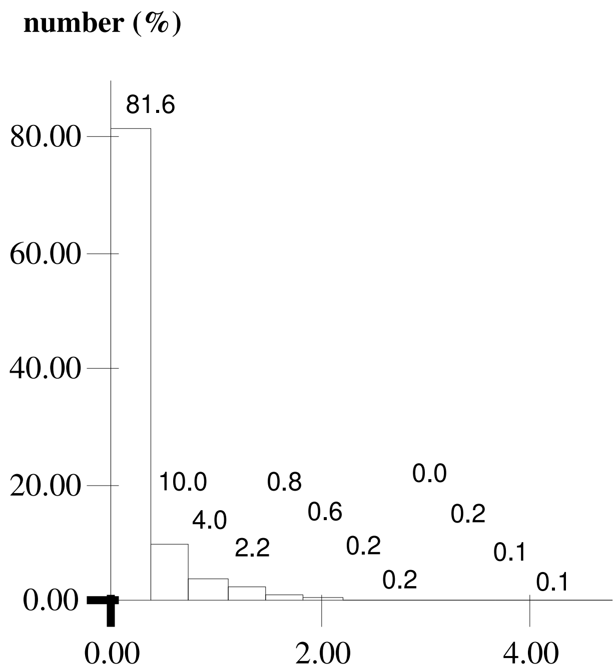


Figure 6.11 Performance histogram of random SANS for the bounder. Most generated SANs produce no useful motion, but a small percentage producing useful motion can consistently be found.

small adjustments to some of the parameters of the SAN and the creature to improve its performance. The weights chosen in the previous step are not among the parameters to be adjusted, however. Because of the non-linear operation of the nodes in the SAN, small changes in the weights result in either no change at all or a very large change in the dynamics of the system.

The parameters we shall adjust in the fine-tuning phase are those associated with the sensors and the actuators, as well as the delays for the hidden nodes. We are thus making some adjustments to both the creature as it was initially designed and the SAN being used to control its motion. The list of adjustable parameters is shown in Table 6.1. We wish to retain the same fundamental mode of locomotion but *evolve* the values of some of the parameters to obtain an improved gait, as defined by the optimization metric. There are thus 4 parameters to adjust for each actuator, 2 for each SAN node, and 2 for each sensor. For our bounder, this results in a total of 44 adjustable parameters.

The approach taken in phase 2 is to make small positive or negative perturbations to randomly chosen parameters. Evaluations are then performed to see if a given change improves the resulting motion. We consider two different approaches to performing and evaluating the parameter changes. These are stochastic gradient ascent (SGA) and simulated annealing (SA). These two approaches are distinct in that one searches for a local optimum, while the second searches for a global optimum. Our stochastic gradient ascent algorithm is often referred to as *stochastic gradient descent* in literature where a minimum value of a function is desired.

Table 6.1 Adjustable parameters for phase 2.

Parameter	description
k1	delay in SAN node for turning on
k2	delay in SAN node for turning off
Amin	minimum desired angle or length for actuator
Amax	maximum desired angle or length for actuator
kp	spring constant for actuator
kd	damper constant for actuator
Smin	lower bound of sensing range for sensor
Smax	upper bound of sensing range for sensor

The structure of the SGA and SA algorithms is similar and is shown in Figure 6.12. The SGA algorithm has probability $p = 1$ of rejecting a change resulting in worse performance, while the SA algorithm gradually increases the value of p over time. This creates some temporary setbacks in the performance of the controller, but these will hopefully be offset by the gains to be had from escaping locally optimal solutions. The SA algorithm used is equivalent to that described in [31] and [52]. The simulated annealing algorithm accepts some parameter changes that result in worse performance in order to be able to escape local minima (or maxima in our case). Both algorithms begin with the nominal parameter values assigned when the sensors and actuators were first designed.

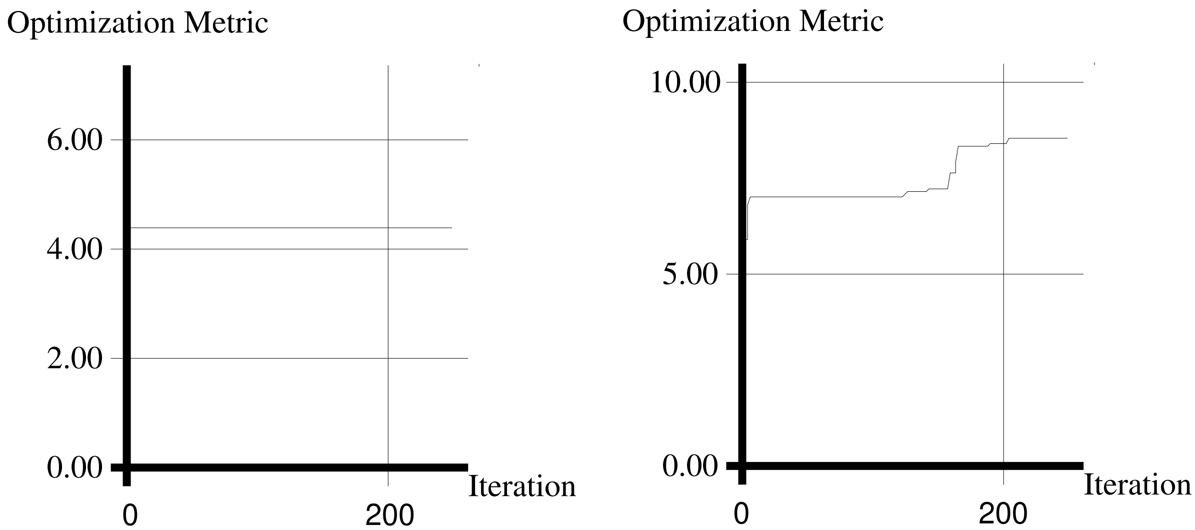
```
for (1000 trials)
        randomly choose a parameter to vary
        perturb the parameter value by +delta or -delta
        evaluate the new creature by simulation
        if (creature improved) then
                keep change
        else
                reject change with probability p
```

Figure 6.12 Pseudocode for stochastic gradient ascent and simulated annealing algorithms.

The results of applying the SGA algorithm to the bounder and walker creatures are shown in Figure 6.13. The SGA algorithm did not improve the performance of the bounder at all, indicating the initial solution was a local minimum. The walker does make some improvements in its gait. The results for simulated annealing are shown in Figure 6.14. The simulated annealing schedule consisted of 65 evaluations at each annealing temperature, with the next 'temperature' being 0.75 times the previous one. For the particular example of the walker, SGA produced better results than SA for the same number of evaluations. In this case the SA algorithm accepts several bad decisions from which it cannot completely recover. In general, we expect that simulated annealing finds better solutions given many evaluations and a suitable annealing schedule, while stochastic gradient descent quickly finds reasonable local maxima.

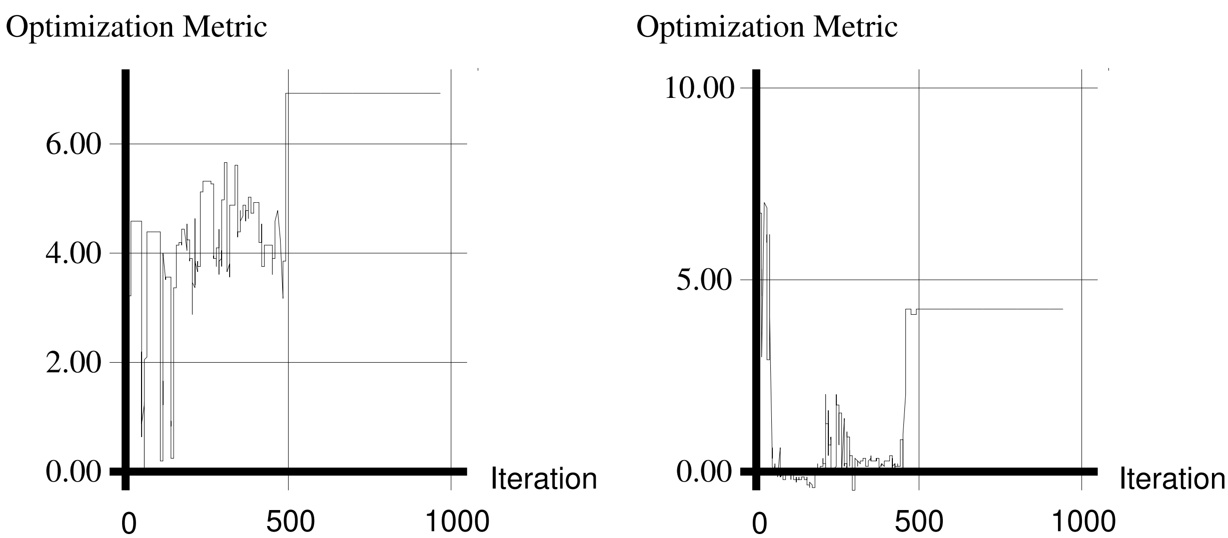
6.5 Results

SANs prove to be capable of serving as useful controllers for almost any relatively simple creature. We have successfully experimented with a total of 10 creatures. These creatures are listed in Table 6.2, and their mechanical configurations are given in Appendix B. Figures 6.15 and 6.16 show frames from an animation using SAN-controlled creatures. All of the creatures, with the exception of the fish and the crawler, are drawn in the vertical plane and make use of the ground to propel themselves forward. The fish makes use of the reaction forces of water to propel itself forward. The four masses of the crawler experience less friction sliding forward than backwards. The crawler uses this property to be able to move forward.



(a) the bounder (b) the walker

Figure 6.13 Stochastic gradient ascent results.
The bounder creature does not improve in the example on the left; it is already in a local minima. The example on the right using the walker creature makes several improvements in performance.



(a) the bounder (b) the walker

Figure 6.14 Simulated annealing results.
The sample run for the bounder accepts several serious setbacks, but finally results in a much-improved motion. The sample run for the walker shows a case where there was no recovery from some an early acceptance of a bad decisions.

Table 6.2 The experimental creatures.

creature	links	sensors	actuators	hidden nodes	top-speed (cm/s)
crawler	4	8	2	10	11
fish	4	6	2	5	19
bounder	5	8	4	8	115
luxo	3	6	2	6	79
cart	2	6	1	5	23
walker	6	11	5	6	101
threelink	3	8	2	7	12
twolink	2	6	1	5	33
star	3	8	2	7	55
fourlink	4	11	3	8	9

Most of the creature designs shown are the original and only attempts at designing the creature. There remains, however, a certain measure of intuition involved in determining what sensors and actuators might be useful to the creature. The angle ranges of the sensors and actuators for both the bounder and luxo were varied in several experiments before settling on the given choices, which seem to be capable of yielding a broad range of interesting gaits.

Figure 6.15 Bounders on the run.

Figure 6.16 Walkers on campus.

An evaluation of 200 random controllers (phase 1) finds useful controllers for each of the creatures, although a wider variety of gaits can be obtained by evaluating even more. Typically, we automatically retain the top 10 solutions for later examination. For most creatures we use the distance moved in 6 seconds as an evaluation metric. A 30 second evaluation is used for the crawler and the fish in order to be able to test their tracking ability. An evaluation of 200 controllers thus requires 1200 seconds of simulation for most creatures. This can take from 1 hour for the cart creature to 6 hours for the walker creature on a Sun SPARC IPC. Faster results could be expected on many other modern workstations. The process is also trivially parallelizable by distributing the trials.

The forces and torques produced by the SAN controllers are usually not smooth because of the presence of impacts and the use of binary sensors. Figure 6.17 shows the torques for the actuators of the top-performing luxo creature. It can be seen from the graph of torque A2 that this

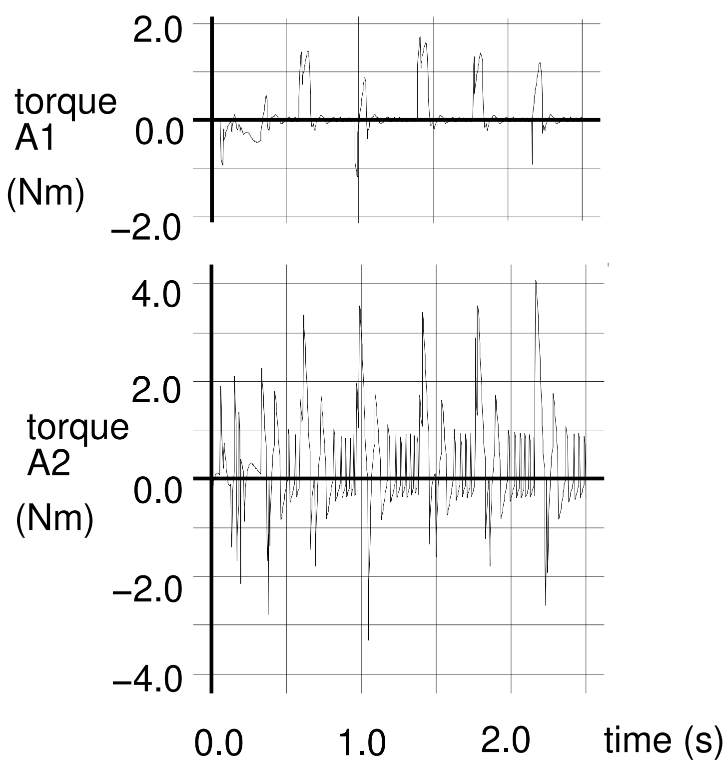


Figure 6.17 Torques for a speedy Luxo creature.
The high-frequency oscillations seen in the torques are a result of the underdamped spring-and-damper constants.

joint is very much underdamped. This solution has emerged because a stronger spring allows for a more energetic and hence faster motion.

The crawler and the fish were designed to perform tracking. Their purpose in life is to always swim or crawl towards the follow point. They perceive this follow point only through their binary eye sensors. The fish has a joint connecting link L4 to L3 that is controlled by a passive angular spring and damper. This joint and link L4 serve as the *caudal* fin of the fish, a necessary feature for efficient swimming. The crawler and the fish are creatures that form a dynamical system with their SANs that is capable of tracking an object successfully. They can do so with binary eyes and have never been given any information on how to move forward, turn, or associate eye information with turning. Figure 6.18 shows a spline path along which we have moved the follow point (the dotted line) and also the resulting path taken by the head of the fish in chasing it Figure 6.19 shows the path followed by the crawler in chasing a moving follow point.

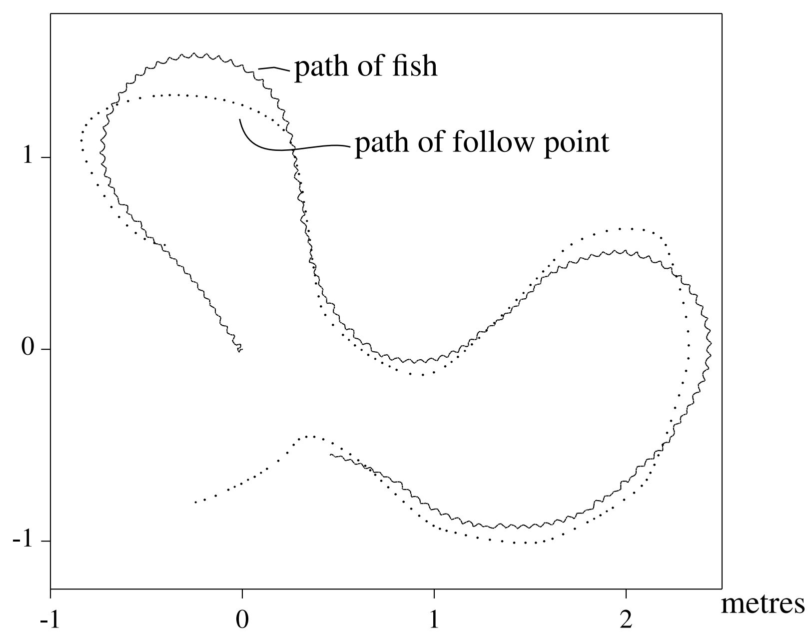


Figure 6.18 The fish creature chasing the follow point.
The fish can only ‘see’ the follow-point with its three binary eyes.
The evaluation metric used was a measure of how well pursuit is maintained.

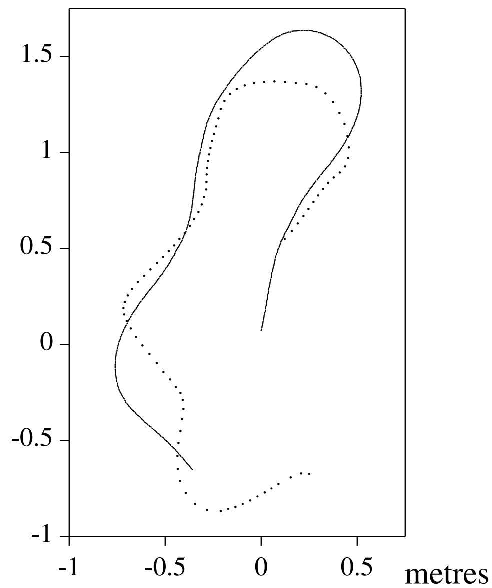
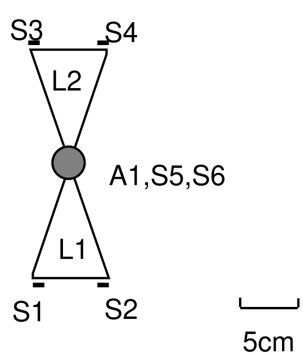


Figure 6.19 The crawler creature chasing the follow point.
The crawler has two binary eyes it uses to perform tracking. A suitable SAN for tracking behaviour is automatically synthesized by using a measure of how well pursuit is maintained as an evaluation metric.

`mechanical configuration:`



link	mass (kg)
L1	0.10
L2	0.10

`sensors:`

sensor	type	link	min	max
S1	touch	L1		
S2	touch	L1		
S3	touch	L2		
S4	touch	L2		
S5	angle	L2	–180	–120
S6	angle	L2	120	180

`actuators:`

act.	min	max	ks	kd
A1	–140	140	0.002	0.0001

Figure 6.20 The cart creature.
The two triangular halves can swing to touch each other, but cannot swing through each other.

6.5.1 Variety of Solutions

The most novel modes of locomotion were discovered for very simple creatures, such as those having only two or three links. The cart creature, shown in Figure 6.20, s perhaps the best

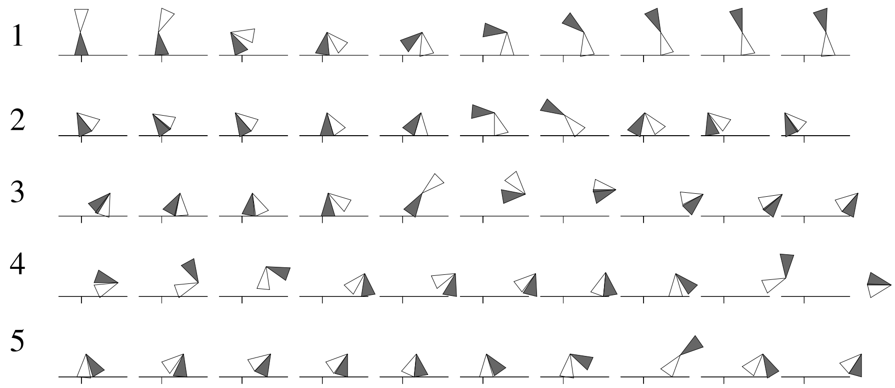


Figure 6.21 Five modes of locomotion for the cart creature.
The creature was originally constructed in the hope that the end-over-end flipping motion, shown as mode 1, could be found. This was but one of five unique modes of locomotion discovered.

example. The creature derives its name from 'cartwheel' because it was originally designed with the hope it could move by flipping end-over-end. Performing cartwheels is indeed one of the physically-feasible modes of locomotion discovered. The creature is also capable of four other unique and valid gaits, however. All 5 modes are shown in Figure 6.21. These modes were found in a run of 1000 controller evaluations. Several modes looked promising, but were unable to fully sustain periodic locomotion initially. These motions were easily fixed to yield proper periodic locomotion using the fine-tuning phase as described in the previous section.

A simple two-link chain is also capable of remarkable motions (see Appendix B for its construction). It is important to note that one of the links is heavier than the other, a property that was chosen to avoid situations where the chain is unable to locomote because of its symmetry. One SAN moves the creature forward using a flapping motion that lifts the joint vertex upwards and forwards in small hops. Another performs repeated 'jumps' onto its back. Yet another manages to get the links into a position such that it can do a big aerial jump. It 'falls' upon landing, but manages to get up to perform another jump, forming a repetitive motion.

More complex figures such as the bounder and the walker produce gaits more familiar to us. The bounder has gaits moving in small hops, big hops, shuffling, and others which are difficult to describe succinctly. The walker moves by performing shuffles, hops, and taking alternating steps. For many objects, the gaits produced are dependent upon the coefficient of friction. This is particularly the case for the three-link and four-link chain creatures.

Many of the solutions not ranking among the best in terms of the evaluation metric can also be interesting. Many creatures quickly become fixed in an immobile state. Others, however, move a bit and then perform a repeated twitching motion with a leg. This is also the case for creatures that fall over into a state from which they cannot get up. These often repetitively twitch their legs in a style reminiscent of an upside-down insect.

There are limits to the complexity of motion that can be expected to emerge by evaluating randomly-generated controllers. It would be futile, for example, to wait for a SAN to be found that controlled a reasonable model of a human body performing a high-jump. As we will discuss shortly, however, we believe that more complex controllers might be achievable through an evolutionary approach.

6.5.2 Robustness of Solutions

SANs are also an interesting control structure because they can produce robust motions, although this is determined experimentally rather than in a control-theoretic sense. Figure 6.22 shows a bounder moving over rough terrain using a SAN that was chosen for its performance over flat terrain. A measure of robustness could be included in the search procedure of phase 1 by performing all the evaluations over terrain of the desired roughness. This experimental notion of robustness is different from the classical notion in that no guarantees on performance can be given.

The robustness has limits because the creatures as presently constructed have no means of detecting the upcoming terrain. As such, they are functioning in a manner equivalent to a person walking blindly over rough terrain. When moving over rough terrain, the same mode of

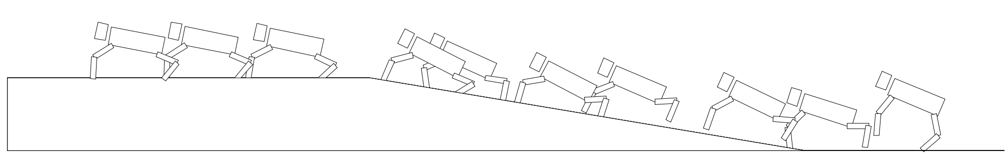


Figure 6.22 The bounder creature climbing a hill.

locomotion is usually maintained, with the variations in terrain causing the timing of events to be changed slightly. The gaits are not as robust once they have been fine-tuned. The situation is analogous to running blindly as opposed to walking blindly. Motions optimized for speed are usually less stable.

6.5.3 The SAN in Operation

An insight into how SANs function can be obtained by looking at the values of the nodes during operation. Figure 6.23 shows the binary output values of each node for two SAN controllers constructed for the 'bounder' creature. S1-S8 are the values of the corresponding sensor nodes, H1-H8 are the values of hidden nodes, and A1-A4 are the values of actuator nodes. It is worthwhile restating that the actuator value used for control of the creature consists of the sum in the actuator node, which is first bounded to the range [-2,2] and then linearly mapped to the range of the given actuator as a desired angle or position. The actuator nodes also retain their binary output, which are connected to other nodes in the network.

There are several observations that can be made from Figure 6.23. The actuator nodes do not have to oscillate for them to be producing motion. As mentioned earlier, the actuators make direct use of the result of the summing operation of a node, which is then mapped into the appropriate actuator range to create a setpoint for the PD control of the actuator.

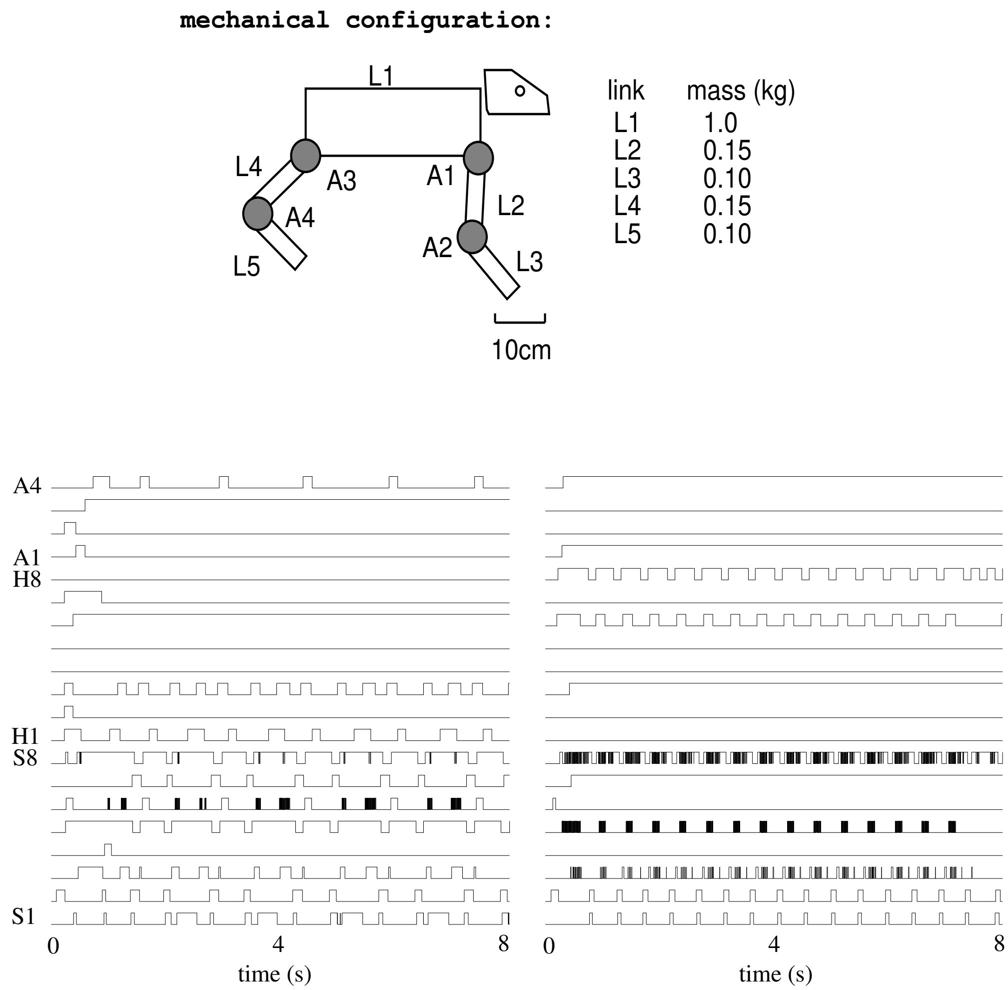


Figure 6.23 SAN operation: two controllers for the bounder creature. The rapid oscillation of the angle sensors occurs because of the underdamped spring and damper constants used for the actuators.

Many of the hidden nodes do not participate in the control of the motion because it can be seen that they do not change in value. These nodes could be removed from the network by placing appropriate constant values on all of their output connections. The hidden and actuator nodes that do participate in the oscillatory pattern eventually behave in a periodic way, corresponding to the periodic mode of locomotion that is observed. The period of these cycles is very different in the two examples shown. The results shown on the left have a period of 1.5 seconds, while those on the right has a period of 0.4 seconds. The initial motion is not periodic, corresponding to the point where the bounder was first dropped onto the ground. The bounder is dropped from a fixed starting configuration onto the ground from a height of 2 cm. The nodes in the network are also initialized to zero. Activity in the network begins when the back leg of the bounder touches the ground, activating sensor S2.

Many of the sensor nodes exhibit chattering behaviour, where they momentarily turn on and off. This is a result of the springs and dampers we use to simulate contact with the ground as well as the underdamped PD control used for the angular actuators. The network is able to function properly despite the erratic behaviour of the sensors because of the integrator and hysteresis loop in the hidden and actuator nodes.

6.5.4 Evolution

The second phase of the network synthesis involves making small changes to some parameters of the sensors, actuators, and delays in the SAN. One could also consider including small changes to the actual physical parameters of the system in order to improve its motion. These parameters could include the link lengths, masses, and points of attachment. Such changes are akin to an artificial kind of evolution. For adherence to the constraints of biological systems, the values of these parameters should be interrelated. For example, a stronger spring constant in the actuator, corresponding to a stronger 'muscle', should increase the mass of the appropriate link. Similarly, a larger link should also have an increased mass. One could also consider measuring the mechanical stress placed on the links during locomotion to ensure that the dimensions of the link are sufficient to support the given gait. It would be interesting to experiment with other physical properties of the links, such as making them deformable.

The evolutionary principle could also perhaps be applied to the synthesis of controllers for more complex systems. It is unlikely that random search will stumble upon the best mode of locomotion for a complex articulated figure, such as a good model of a horse. We feel that this is not a large shortcoming of our method because nature itself does not directly arrive at suitable controllers for such creatures. The skeleton, musculature, and control for a horse are the result of a long series of evolutionary changes. It might be possible to arrive at complex controllers by first learning how to control simpler figures that are similar, such as those that have fewer joints or that have more stable locomotion. The large feet placed on our walker is an example of this. One might be able to generate a controller for a walker with more reasonably sized feet by incrementally reducing the size of the feet while determining the controller adjustments necessary to compensate. The idea is to use the controller of a similar, more statically stable creature as a starting point for the controller of a more dynamic motion.

6.6 Conclusions

We have presented a new method that automatically searches for interesting and physically-feasible modes of locomotion for arbitrarily-designed creatures. The solutions are in the form of simple controllers that use binary sensors as input and produce actuator forces and torques. Sensor-actuator networks are a very compact representation of the complex and varied gaits that they produce. We have demonstrated their use for a series of ten different creatures. We are furthermore able to fine-tune the performance of a creature by automating the adjustment of many parameters of the SAN and the creature.

There are many advantages to using SANs. Using the method is simple. It requires no knowledge of the underlying equations of motion. The user can rapidly construct a new creature and have the method ‘discover’ the ways it can locomote. The method goes beyond using physically-based constraints to perform a kind of interpolation. We feel that this method encourages experimentation with the construction of creatures. Because no knowledge of the physics of motion beyond a simulation capability is required, it is easy to incorporate more complex physical phenomena and to generate controllers that take advantage of these phenomena. As an example, the coefficient of friction can have a profound impact on the choice of the best gait. SANs are amenable to the control of mechanical systems that do not always have smooth motion. This includes a large class of interesting creatures.

Our method always results in physically-realizable motions, something that can be an advantage or a restriction in animation. The user can influence the motion through the specification of the evaluation metric and evaluation terrain in order to achieve the desired speed, energy, height, and robustness of the control. The ability to fine-tune the sensors, actuators, and delays in the SAN means the creature designer need not choose the optimal design parameters to begin with. We believe this kind of artificial 'evolution' will be important in creating more complex controllers.

SANs have some disadvantages from the point of view of an animator. They do not at present provide the user with as much control over the resulting motion as keyframing or physics-as-constraints methods. Furthermore, there is no guarantee that any useful solution will be found. SANs in their current incarnation do not work very well for systems dominated by linear or smooth dynamics. We have tried in some simple experiments to produce a SAN to control our physical turning model of Chapter 5 to chase a follow point, but with no success. The synthesis method thus far fails to find the best modes of locomotion for complex creatures.

We believe there are many possible future directions for this work. Many experiments can be carried out using other types of architectures, sensors, and actuators. SANs could possibly be generated for deformable objects. Elastic elements can play important roles in many motions [3]. The fine-tuning phase could be used to determine how to insert passive elastic elements into systems in order to reduce the energy expended for a given mode of locomotion. The use of SANs as the bottom level of a control hierarchy needs to be investigated. The simplicity of the SAN controllers makes the real construction of some of the creatures a possibility. More understanding and analysis of the dynamical systems formed by the SANs and the creatures is needed, similar to the analysis performed in [54] and [92] for simple hopping robots. One should be able to construct a limit cycle corresponding to a desired gait and give it a wide domain of attraction.

Chapter 7 Conclusions and Future Work

While we are yet unable to automatically create a realistic animation of a squirrel leaping from branch to branch, we have presented a variety of methods that make inroads toward solving such a problem. In this chapter we shall first summarize the contributions of our work and then present some thoughts on future work in this area.

7.1 Discussion of Contributions

We have presented several approaches for controlling physically-based animations. We emphasize that for active creatures, one cannot properly create a physically-based animation without solving the associated control problem. Our control techniques provide animators with new tools to aid in minimizing the effort needed to create realistic animations. While such tools may be powerful, they are meant to augment other techniques such as keyframing. Proceeding from a physically-based animation sequence to one determined by keyframing is a simple matter. The position of the object in the last frame can be used as a starting keyframe for the keyframed sequence of motions. Proceeding in the other direction is also a relatively simple matter, because there is also position and velocity information present in keyframe sequences that can be used as an initial state of the figure for the beginning of a physically-based animation.

This thesis has primarily been concerned with the synthesis of control solutions for articulated figures. The space of possible solutions for this problem is very large and still not very well studied or understood at this point. For this reason we have chosen to develop and refine several new techniques instead of focusing our investigation on just one technique. Although the process of simulating any one of our creatures or characters is deterministic and relatively well defined, controlling it is nevertheless a difficult problem. Knowing all the details about the mechanics of a motion at every instant in time does not provide many clues as to how to best create a control structure that must operate on a larger time scale.

Our results indicate that choosing the right representation for a motion control problem greatly affects the complexity of finding solutions. Different representations implicitly place different constraints upon the solution and thus offer different search spaces which must be traversed. Approximating-graph dynamic programming (AGDP) offers state-space as a search space, effectively encompassing all possible motions. This search space rapidly becomes too large for anything beyond the simplest of objects. The walking controller is an example of how the search space can be reduced for the task of walking by imposing periodicity upon the solution, as well as making several other simplifying assumptions.

In the path-following control method for turning objects, the search space consisted of exploring all motions possible from the current state within a fixed-length time window. This particular search space can be explored efficiently with a branch-and-bound algorithm. Even so, this type of search space rapidly becomes too large for objects with multiple control inputs.

The last representation we present, that of sensor-actuator networks, has a search space consisting of a set of parameter values in a predefined control structure. While this search space is very large, there also appear to be many reasonable solutions possible, many of which can be found relativly easily through a generate-and-test procedure. The solutions can then be further refined in an efficient way with a second ‘fine-tuning’ phase, making for an effective coarse-to-fine search through the parameter space.

Much of the work in this thesis relies on numerical instead of analytical representations of the equations of motion. In the AGDP work of Chapter 3 we perform short simulations to generate the arcs in the approximating graph. Similarly, in Chapter 4, we suggest the use of a particular decomposition for a walking controller and experimentally verify the usefulness and robustness of the resulting controller. Chapter 5 presents a branch-and-bound optimization method of controlling our turning model. This optimization method again makes use of many successive simulations. Lastly, the SANs of Chapter 6 take the use of simulations to an extreme in generating suitable controllers. The one exception to this is the parameterized control of turning, where an explicit analysis of the equations of motion proves to be very useful in yielding a parameterized control method.

One common characteristic of our animation control methods is that they do not calculate control directly as a function of time. In keyframe animation, motion is viewed strictly as a function of time. In keeping with this tradition, some of the early influential work on control for physically-based animation views the control as a quantity to be calculated over time. As with keyframing, this leads to the production of a trajectory, not a controller. While this has certain advantages for the animator in terms of specifying the desired motion, it results in a solution that is specific to the situation it was calculated for.

There are many possible domains for control functions other than time and we have explored only some of them. The AGDP method determines the control as a function of the object's state. SANs calculate control as a function of a set of binary sensors. The walking controller calculates control based on the current state and phase. The parametric turning controller is also state-based. The path-following turn controller produces control as a function of time and its environment.

The work in this thesis has implications for several other fields. In terms of control, the AGDP technique is a new method of applying dynamic programming to one-point boundary problems. The walking work represents a useful example of how specific complex problems can be decomposed to find a workable solution. Lastly, the work on SANs demonstrates how suitable controllers can be synthesized using very general objective functions and experimental means.

There are also implications for building robots and autonomous systems contained in our work. The work on AGDP suggests that dynamic programming can be used for the purposes of control, in effect performing path planning through the state-space of the system. The success of SANs suggests that interesting, capable systems can be built that use a minimal amount of sensory information. Lastly, the SANs suggest that automatic synthesis of dynamic locomotion and tracking behaviours is feasible.

Lastly, we believe our work can be of use in the study of biomechanics of humans and a wide variety of animals. Using the experimental approach taken for the SANs, it might be possible to determine how the body shape and mass of a given creature should evolve to improve its locomotion. For example, a more efficient walk might be attainable if one of the limbs was slightly longer or had a different centre of mass. Similarly, we could perform experiments to determine how much energy could be saved for specific motions by allowing for the storage of energy in elastic elements of the system.

7.2 Future Work

There are many possible future directions to take in expanding upon the work presented in this thesis. Some specific directions have already been outlined in the concluding remarks of each chapter. The suggestions for work we shall present here are of a more general nature.

In order for physically-based animation to be a truly useful tool for animators, it is necessary to restore some of the animator's control over the motion. This means that controllers should be parameterized in terms of variables useful to the animator, such as speed, height, energy, etc. While some of the controllers we have presented are parameterized, they were not all the product of a completely automatic synthesis process. This will have to be the case if we wish to provide controllers that serve as general tools for animators, who will want to experiment with new and unique creatures. The SANs seem to offer the possibility of being easy to parameterize.

While the work in this thesis has dealt exclusively with the control problem, there is a need for improved 'skins' or 'clothes' with which to dress our mechanical models to produce interesting images. It needs to be determined how such skins should be attached to the mechanical skeleton and how it can be made to deform upon bending of the joints or contact with other objects. Previous work in this area exists ([20] for example), but further work could yield better solutions.

The motions produced by the SANs are periodic in nature. Several of the other types of controllers we have introduced allow for non-periodic motion. It would be interesting to examine how one could generalize SANs to allow for non-periodic behaviour, such as that of an arm reaching for a cup. One could perhaps conceive of a way to increase the number of nodes in a SAN to allow for an incremental approach to achieving refined motions.

Motions produced by physically-based animations are realistic, at least in so far as the laws of physics are maintained. Most characters in cartoon animations have exaggerated motions that do not obey the laws of physics. Nevertheless, these motions are presumably recognizable because they are based upon real motions that do obey the appropriate laws. There is thus a need for a method to create motion caricatures based upon physically-based motions.

If multiple control techniques are necessary for physically-based animation, it will be an interesting challenge to integrate the various techniques into a single system that is still easy to use by the animator. Even if a single control technique dominates, it will at least be necessary to provide a seamless integration with keyframing methods.

One can wonder if the perception of motions can be related to the creation of motions. This would be an interesting relationship to study, given that the work presented in this thesis can be used to create the motions of some creatures having real biological counterparts.

Lastly, it is possible to model humans and animals at much greater levels of detail than as a set of connected rigid links with torques acting upon them. The physiology of musculo-skeletal systems is quite complex and may hold the key to generating even more realistic motions. A simple first step in this direction would be the use of antagonistic muscle pairs at each joint instead of directly generating applied torques.

Appendix A: Equations of motion for turning

The following is a brief summary of the equations of motion of the turning model.

A free-body diagram showing the forces and moments acting on the body is shown in Figure 1. The equations of motion of the body are given below. F_x and F_y refer to the horizontal and vertical components of the force exerted on the foot by the ground. I_v is the moment of inertia of the body about its centre of mass. α_v is the angular acceleration of the body and leg about the z-axis. a_x and a_y are the horizontal and vertical components, respectively, of the linear acceleration of the body. These terms are related as follows.

$$T = F_x l cos\theta_v + F_y l sin\theta_v = I_v \alpha_v \tag{A.1}$$

$$F_x = ma_x \tag{A.2}$$

$$F_y = ma_y + mg \tag{A.3}$$

Equation A.1 equates the sum of the angular moments in the xy-plane to the change in angular momentum, and Equations A.2 and A.3 equate the sum of the forces to the change in linear momentum.

The acceleration of the centre of mass can be expressed relative to the acceleration of the foot, a_f, as in equation A.4 below. Here, $a_{cm/f}$ is the acceleration of the centre of mass with respect to the foot, and a_c is the coriolis force, which is zero in our case. The orthogonal decomposition of equation A.4 is given in equations A.5 and A.6. ω_v is the angular velocity in the xy-plane. V_f is the forward velocity of the foot. r_f is the radius of curvature of the foot in the

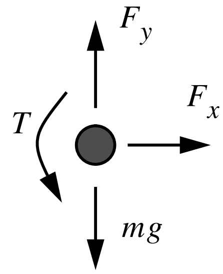


Figure A.1 Free body diagram.

xz-plane, and represents our control variable. The last term in equation A.5 is due to rotation of the body in the xz-plane.

$$a_{cm} = a_f + a_{cm/f} + a_c \tag{A.4}$$

$$a_x = a_{fx} - l\cos\theta_v\alpha_v + l\sin\theta_v\omega_v^2 + l\sin\theta_v\frac{V_f^2}{r_f^2} \tag{A.5}$$

$$a_y = a_{fy} - l\sin\theta_v\alpha_v - l\cos\theta_v\omega_v^2 \tag{A.6}$$

Substituting equations A.5 and A.6 into equations A.2 and A.3, and then using the results in equation A.1 gives the following:

$$I_v\alpha_v = mll_c\left(a_{fx} + l_s\frac{V_f^2}{r_f^2} - l_c\alpha_v + l_s\omega_v^2\right) + ml_s(a_{fy} - l_s\alpha_v - l_c\omega_v^2 + g) \tag{A.7}$$

where $l_c = l\cos\theta_v$ and $l_s = l\sin\theta_v$.

Assuming that the foot is travelling on a planar surface, we can write $a_{fx} = -V_f^2/r_f$ and a_{fy}=0. Substituting these into equation A.7 and combining terms gives the following result:

$$\frac{I_v\alpha_v}{ml} = -\frac{V_f^2}{r_f}\cos\theta_v + l\sin\theta_v\cos\theta_v\frac{V_f^2}{r_f^2} - l\alpha_v + g\sin\theta_v \tag{A.8}$$

Lastly, because we assume that the forward velocity, V_{cm} is known, we can relate the forward velocity of the foot to that of the centre of mass as follows:

$$V_f = V_{cm}\left(\frac{r_f}{r_f - l\,sin\theta_v}\right). \tag{A.9}$$

Using the result of equations A.8 and A.9, we can arrange the terms to form a quadratic equation in terms of the curvature of the foot, C_f, where $C_f = 1/r_f$. This result is equation A.10.

$$0 = aC_f^2 + bC_f + c, \tag{A.10}$$

where

$$a = (l\,sin\theta_v)^2 k + lV_{cm}^2 sin\theta_v cos\theta_v$$

$$b = -2l\,sin\theta_v k - V_{cm}^2 cos\theta_v$$

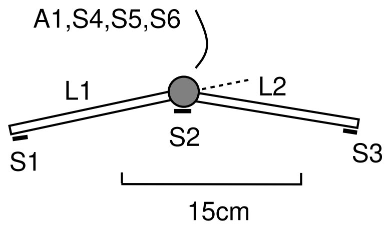
$$c = k$$

$$k = g\,sin\theta_v - \frac{\alpha_v}{ml}(I_v + l^2 m)$$

In the expressions for a, b, and c, k is a constant useful in simplifying the equation. The negative root of equation A.10 should be used, as it can be shown that this leads to the correct result of zero curvature necessary to obtain zero angular acceleration when the body is already vertical. The equation allows us to calculate C_f given α_v and vice-versa.

Appendix B: SAN creatures

mechanical configuration:



sensors:

sensor	type	link	min	max
S1	touch	L1		
S2	touch	L2		
S3	touch	L2		
S4	angle	L2	–180	45
S5	angle	L2	45	130
S6	angle	L2	130	200

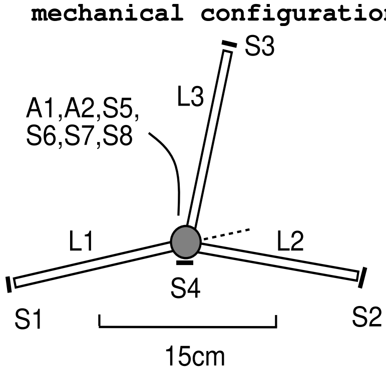
actuators:

act.	min	max	ks	kd
A1	30	150	0.006	0.0001

link	mass (kg)
L1	0.15
L2	0.10

Figure B.1 The twolink creature.

mechanical configuration:



sensors:

sensor	type	link	min	max
S1	touch	L1		
S2	touch	L2		
S3	touch	L3		
S4	touch	L1		
S5	angle	L2	–200	–160
S6	angle	L2	–40	200
S7	angle	L3	–200	40
S8	angle	L3	160	200

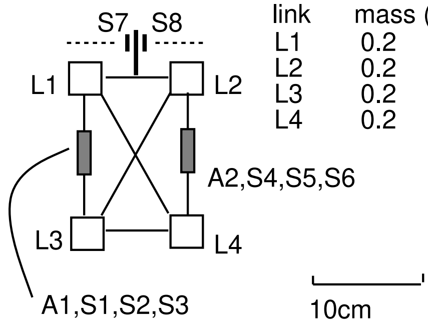
actuators:

act.	min	max	ks	kd
A1	–170	–30	0.008	0.0001
A2	30	170	0.008	0.0001

link	mass (kg)
L1	0.1
L2	0.1
L3	0.1

Figure B.2 The star creature.

mechanical configuration:



link	mass (kg)
L1	0.2
L2	0.2
L3	0.2
L4	0.2

Each line connecting two links is a spring and damper pair. The actuators A1 and A2 control the rest length of the side springs. The sensors S1–S6 measure the lengths of the sides.

sensors:

sensor	type	min	max
S1	length	0.0	12.5
S2	length	12.5	17.5
S3	length	17.5	25.0
S4	length	0.0	12.5
S5	length	12.5	17.5
S6	length	17.5	25.0
S7	eye	–80	80
S8	eye	–80	80

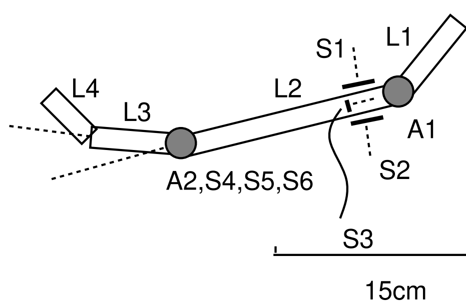
actuators:

act.	min	max	ks	kd
A1	0.10	0.20	10	1
A2	0.10	0.20	10	1

The masses have less friction sliding forwards than backwards.

Figure B.3 The crawler.

mechanical configuration:



link	mass (kg)
L1	0.2
L2	1.0
L3	0.3
L4	0.1

sensors:

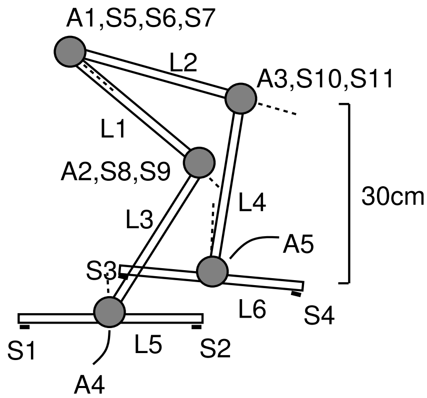
sensor	type	link	min	max
S1	eye	L2	10	180
S2	eye	L2	–10	10
S3	eye	L2	–180	–10
S4	angle	L3	–180	–25
S5	angle	L3	–25	25
S6	angle	L3	25	180

actuators:

act.	min	max	ks	kd
A1	–20	20	0.003	0.001
A2	–40	40	0.006	0.001

Figure B.4 The fish.

mechanical configuration:



link	mass (kg)
L1	0.3
L2	0.3
L3	0.2
L4	0.2
L5	0.1
L6	0.1

sensors:

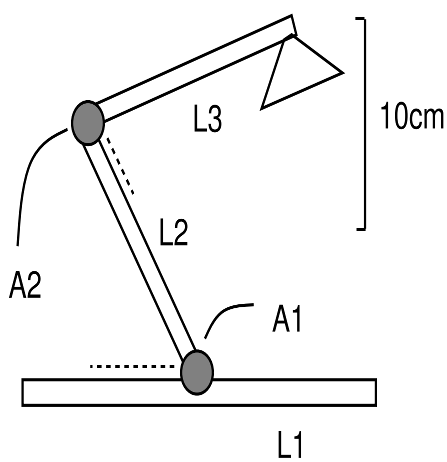
sensor	type	link	min	max
S1	touch	L5		
S2	touch	L5		
S3	touch	L6		
S4	touch	L6		
S5	angle	L2	–180	–15
S6	angle	L2	–15	–5
S7	angle	L2	15	180
S8	angle	L3	–180	–85
S9	angle	L3	–75	180
S10	angle	L4	–180	–85
S11	angle	L4	–75	180

actuators:

act.	min	max	ks	kd
A1	–30	30	0.15	0.01
A2	–105	–70	0.15	0.01
A3	–105	–70	0.15	0.01
A4	20	30	0.10	0.005
A5	20	30	0.10	0.005

Figure B.5 The walker.

mechanical configuration:



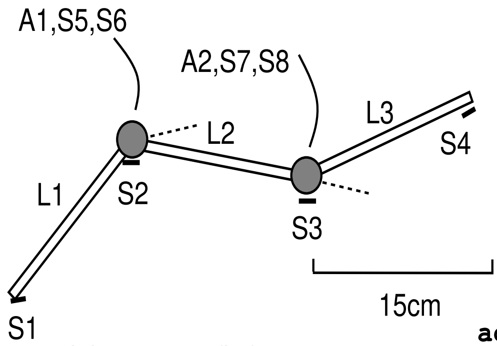
link	mass (kg)
L1	0.05
L2	0.10
L3	0.30

actuators:

act.	min	max	ks	kd
A1	–70	–50	0.05	0.001
A2	60	150	0.04	0.001

Figure B.6 Luxo.

mechanical configuration:



link	mass (kg)
L1	0.1
L2	0.1
L3	0.1

sensors:

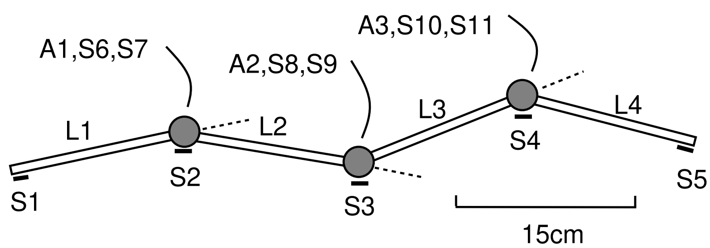
sensor	type	link	min	max
S1	touch	L1		
S2	touch	L2		
S3	touch	L3		
S4	touch	L3		
S5	angle	L2	–180	–90
S6	angle	L2	90	180
S7	angle	L3	–180	–90
S8	angle	L3	90	180

actuators:

act.	min	max	ks	kd
A1	–110	110	0.002	0.0001
A2	–110	110	0.002	0.0001

Figure B.7 The threelink creature.

mechanical configuration:



link	mass (kg)
L1	0.1
L2	0.1
L3	0.1
L4	0.1

sensors:

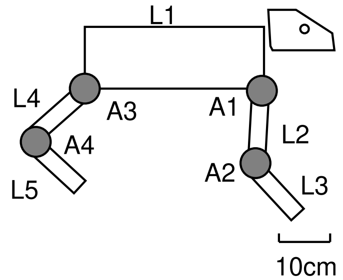
sensor	type	link	min	max
S1	touch	L1		
S2	touch	L2		
S3	touch	L3		
S4	touch	L4		
S5	touch	L4		
S6	angle	L2	–180	–50
S7	angle	L2	50	180
S8	angle	L3	–180	–50
S9	angle	L3	50	180
S10	angle	L4	–180	–50
S11	angle	L4	50	180

actuators:

act.	min	max	ks	kd
A1	–90	90	0.003	0.00015
A2	–90	90	0.003	0.00015
A3	–90	90	0.003	0.00015

Figure B.8 The fourlink creature.

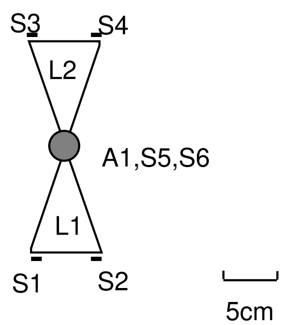
mechanical configuration:



link	mass (kg)
L1	1.0
L2	0.15
L3	0.10
L4	0.15
L5	0.10

Figure B.9 The bounder.

mechanical configuration:



link	mass (kg)
L1	0.10
L2	0.10

sensors:

sensor	type	link	min	max
S1	touch	L1		
S2	touch	L1		
S3	touch	L2		
S4	touch	L2		
S5	angle	L2	–180	–120
S6	angle	L2	120	180

actuators:

act.	min	max	ks	kd
A1	–140	140	0.002	0.0001

Figure B.10 The cart creature.

References

[1] R. M. Alexander. The gaits of bipedal and quadrupedal animals. *International Journal of Robotics Research*, Summer, 1984.

[2] R. M. Alexander. *Elastic Mechanisms in Animal Movement*. Cambridge University Press, 1988.

[3] R. M. Alexander. Three Uses for Springs in Legged Locomotion. *The International Journal of Robotics Research*, 9, 2, 1990, 53-61.

[4] W. W. Armstrong and M. Green. The Dynamics of Articulated Rigid Bodies for Purposes of Animation. *Proceedings of Graphics Interface '85*, 1985, 407-415.

[5] W. W. Armstrong, M. Green, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6), 52-61, June 1987.

[6] H. Asada and J. J. Slotine. *Robot Analysis and Control*. John Wiley and Sons, 1986.

[7] N. I. Badler, B. Barsky, and D. Zeltzer. *Making Them Move*. Morgan Kaufmann Publishers Inc., 1991.

[8] N. I. Badler, K. H. Manoocherhri, and G. Walters. Articulated figure positioning by multiple constraints. IEEE Computer Graphics and Applications, 7(6), 28-38, June 1987.

[9] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron-like adaptive elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5), 1985.

[10] R. Beer. *Intelligence as Adpative Behavior.* Academic Press, 1990.

[11] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[12] R. Boulic, N. M. Thalmann, D. Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6, 1990, 344-358.

[13] V. Braitenberg. *Vehicles: experiments in synthetic psychology.* MIT Press, 1984.

[14] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2, 1 (March 1986), 14-23.

[15] L. S. Brotman and A. N. Netravali. Motion Interpolation by Optimal Control. *Proceedings of SIGGRAPH '88*. In *ACM Computer Graphics*, 22, 4 (August 1988), 309-315.

[16] A. Bruderlin and T. W. Calvert. Goal-Directed Animation of Human Walking. *Proceedings of SIGGRAPH '89*. In *ACM Computer Graphics,* 23(4), 233-242, 1989.

[17] N. Burtnyk and M. Wein. Computer generated keyframe animation. *Journal of the Society of Motion Picture and Television Engineers*, 80(3), 149-153, March 1971.

[18] D. G. Cachola and G. F. Schrack. Modeling and animating three dimensional articulated figures. *Proceedings of Graphics Interface '86*, 152-157, 1986.

[19] T. Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. IEEE Computer Graphics and Applications, 41-50, November 1982.

[20] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered Construction for Deformable Animated Characters. *Proceedings of SIGGRAPH '89. In ACM Computer Graphics*, 23, 3 (July 1989), 243-252.

[21] R. H. Cannon. *Dynamics of Physical Systems*. McGraw-Hill, 1967.

[22] P. H. Channon, S. H. Hopkins, and D. T. Pham. Derivation of optimal walking motions for a bipedal walking robot. *Robotica*, 10, 1992, 165-172.

[23] Chi-Tsong Chen. *Linear System Theory and Design*. Holt, Rinehart, and Winston, 1984.

[24] C. Chow and D. Jacobson. Studies of human locomotion via optimal programming. *Mathematical Biosciences*, v. 10, 239-306, 1971.

[25] M. F. Cohen. Interactive Spacetime Control for Animation. *Proceedings of SIGGRAPH '92*. In *ACM Computer Graphics*, 26, 2 (July 1992), 293-302.

[26] M. Cotsaftis and C. Vibet. Decoupled Control for 2D N-link Biped Walking System. *Robtics and Autonomous Systems*, 5, 1989, 97-107.

[27] C. Csuri. Real time film animation. *IEEE Convention Digest*, p. 42-3, March 1971.

[28] L. Dorst, I. Mandhyan, and K. Trovato. The Geometrical Structure of Path Planning Problems. *Proceedings of the Intelligent Autonomous Systems International Conference*, 1989, 155-167.

[29] K. Drewery and J. Tsotsos. Goal directed animation using english motion commands. Proceedings of Graphics Interface '86, 131-5, 1986.

[30] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer, 1987.

[31] E. Fiume and M. Ouelette. On distributed, probabilistic algorithms for computer graphics. *Proceedings of Graphics Interface '89*, 211-218, 1989.

[32] D. R. Forsey and J. Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. *Proceedings of Graphics Interface '88*, 8-15, 1988.

[33] P. S. Freeman and D. E. Orin. Efficient Dynamic Simulation of a Quadruped Using a Decoupled Tree-Structure Approach. *The International Journal of Robotics Research*, 10, 6, 1991, 619-627.

[34] J. Furusho and M. Masubuchi. A Theoretically Motivated Reduced Order Model for the Control of Dynamic Biped Locomotion. *Journal of Dynamic Systems, Measurement, and Control*, 109, 1987, 155-163.

[35] J. Furusho and A. Sano. Sensor-Based Control of a Nine-Link Biped. *The International Journal of Robotics Research*, 9, 2, 1990, 83-98.

[36] M. Girard. Interactive design of computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6), 39-51, June 1987.

[37] C. L. Golliday and H. Hemami. An Approach to Analyzing Biped Locomotion Dynamics and Designing Robot Locomotion Controls. *IEEE Transactions on Automatic Control,* AC-22, 6, 1977, 963-972.

[38] M. Green. Using Dynamics in Computer Animation: Control and Solution Issues. In *Making Them Move*, N. Badler, B. Barsky, and D. Zeltzer (eds), Morgan Kaufmann, 1991.

[39] M. Green and H. Sun. MML: A Language and and System for Procedural Modelling and Motion. *Proceedings of Graphics Interface '88*, 1988, 16-25.

[40] S. Grillner. Locomotion in vertebrates: Central mechanisms and reflex interaction. *Physiological Reviews*, v. 55, 247-304, 1975.

[41] D. E. Hardt. Determining Muscle Forces in the Leg During Normal Human Walking: An Application and Evaluation of Optimization Methods. *Transactions of the ASME*, 100, 1978, 72-78.

[42] H. Hemami and R. Farnsworth. Postural and gait stability of a planar five link biped by simulation. *IEEE Transactions on Automatic Control*, 452-458, June 1977.

[43] J. K. Hodgins and M. H. Raibert. Biped Gymnastics. *The International Journal of Robotics Research*, 9, 2, 1990, 115-132.

[44] J. K. Hodgins, P. K. Sweeney, and D. G. Lawrence. Generating Natural-looking Motion for Computer Animation. *Proceedings of Graphics Interface '92*, 265-272, May 1992.

[45] C. S. Hsu. A discrete method of optimal control based upon the cell state space concept. *Journal of Optimization Theory and Applications*, 46(4), 1985.

[46] V. T. Inman, H. J. Ralston, and F. Todd. *Human Walking*. Williams & Wilkins, 1981.

[47] P. M. Isaacs and M. F. Cohen. Controlling dynamic simulation with kinematic constraints, behaviour functions, and inverse dynamics. *Proceedings of SIGGRAPH '87*. In *ACM Computer Graphics*, 21, 4, 215-224, 1987.

[48] P. M. Isaacs and M. F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. The Visual Computer, 4, 296-305, 1988.

[49] R. Johansson and M. Magnusson. Optimal Coordination and Control of Posture and Locomotion. *Mathematical Biosciences*, 103, 1991, 203-244.

[50] S. Kajita and K. Tani, Study of Dynamic Biped Locomotion on Rugged Terrain. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, 1991, 1405-1411.

[51] D. E. Kirk. *Optimal Control Theory: An Introduction.* Prentice-Hall Inc., 1970.

[52] S. Kirkpatric, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220,13 (May 1983), 671-680.

[53] S. Kitamura, Y. Kurematsu, and M. Iwata. Motion generation of a biped locomotive robot using an inverted pendulum model and neural networks. *Proceedings of the IEEE Conference on Decision and Control*, 3308-3312, 1990.

[54] D. E. Koditschek and M. Buhler. Analysis of a Simplified Hopping Robot. *The International Journal of Robotics Research*, 10, 6, 1991, 587-605.

[55] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed motion of articulated structures. *IEEE Computer Graphics and Applications*, 71-81, November 1982.

[56] B. C. Kuo. *Automatic Control Systems*. Prentice-Hall, Inc., 1987.

[57] H. C. Kwan. Network Relaxation as Behavioral Action: Some Conjectures on the Control of Movement by the Nervous System. RBCV-TR-88-26, Dept. of Computer Science, University of Toronto, October 1988.

[58] R. E. Larson. A Survey of Dynamic Programming Computational Procedures. *IEEE Transactions on Automatic Control*, 767-774, December 1967.

[59] R. E. Larson. *State-Increment Dynamic Programming*. American Elsevier Publishing Company Inc., 1968.

[60] R. E. Larson and John L. Casti. *Principles of Dynamic Programming.* Marcel Dekker Inc., 1978.

[61] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[62] J. Lasseter. Principles of traditional animation applied to 3-d computer animation. *Proceedings of SIGGRAPH '87*. In *ACM Computer Graphics*, 21, 4, 35-44, 1987.

[63] M. W. Lee and T. L. Kunii. Animation design: A database-oriented animation design method with a video image analysis capability. *Proceedings of Computer Animation '89*, 97-112, 1989.

[64] S. Levy. *Artificial Life: the quest for a new creation*. Pantheon Books, New York, 1992.

[65] M. A. Lewis, A. H. Fagg, and A. Solidum. Genetic Programming Approach to the Construction of a Neural Network for Control of a Walking Robot. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 1992, 2618-2623.

[66] R. Luus. Optimal control by dynamic programming using systematic reduction in grid size. *International Journal of Control,* 51(5), 995-1013, 1990.

[67] P. Maes. Situated Agents Can Have Goals. *Robotics and Autonomous Systems*, 6, 1990, 49-70.

[68] P. Maes and R. Brooks. Learning to Coordinate Behaviors. *Proc. of AAAI '90,* 1990, 796-802.

[69] T. McGeer. Passive Walking with Knees. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1640-1645, 1990.

[70] T. McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9, 2, 1990, 62-82.

[71] M. McKenna and D. Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. *Proceedings of SIGGRAPH '90.* In *ACM Computer Graphics*, 24, 4 (August 1990), 29-38.

[72] T. A. McMahon. Mechanics of Locomotion. *The International Journal of Robotics Research*, 3, 2, 1984, 4-28.

[73] G. S. P. Miller. The Motion Dynamics of Snakes and Worms. *Proceedings of SIGGRAPH '88*. In *ACM Computer Graphics*, 22, 4 (August 1988), 169-178.

[74] H. Miura and I. Shimoyama. Dynamic walk of a biped. *International Journal of Robotics Research*, 60-74, Summer 1984.

[75] M. P. Murray, A. B. Drought, and R. C. Kory. Walking Patterns of Normal Men. *Journal of Bone and Joint Surgery*, 46, 1964, p. 335.

[76] E. Muybridge. *The Human Figure in Motion*. Dover Publications, 1955.

[77] W.-Y. Ng. Perspectives on Search-Based Computer-Aided Control System Design. IEEE Control Systems Magazine, 13, 2 (April 1993), 65-72.

[78] J. T. Ngo and J. Marks. Spacetime Constraints Revisitied. Proceedings of SIGGRAPH '93. In *ACM Computer Graphics*, 27 (August 1993).

[79] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning. *Proceedings of the International Joint Conference on Neural Networks*, 1990.

[80] K. Ogo, A. Ganse, and I. Kato. Quasi-Dynamic Walking of Biped Walking Machine Aiming at Completion of Steady Walking. *Proceedings of the 3rd Symposium on the Theory and Practice of Robots and Manipulators*, 1978, 340-357.

[81] M. G. Pandy, F. E. Zajac, E Sim, and W. S. Levine. An Optimal Control Model for Maximum-Height Human Jumping. *J. Biomechanics*, 23, 12, 1185-1198, 1990.

[82] M. H. Raibert. *Legged robots that balance*. MIT Press, 1986.

[83] M. H. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. *Proceedings of SIGGRAPH '91*, In *ACM Computer Graphics*, 25, 4 (July 1991), 349-358.

[84] C. Reynolds. Computer animation with scripts and actors. *Proceedings of SIGGRAPH '81*. In *ACM Computer Graphics*, 1981.

[85] A. W. Salatian and Y. F. Zheng. Gait Synthesis for a Biped Robot Climbing Sloping Surfaces Using Neural Networks Part I: Static Learning. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 1992, 2601-2606.

[86] A. W. Salatian and Y. F. Zheng. Gait Synthesis for a Biped Robot Climbing Sloping Surfaces Using Neural Networks Part II: Dynamic Learning. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 1992, 2607-2611.

[87] SD/Fast User's Manual, Symbolic Dynamics Inc., Mountain View, California, USA, 1990.

[88] M. L. Shik and G. N. Orlovskii. Neurophysiology of a locomotor automatism. Physiological Reviews, v. 56, 465-501, 1976.

[89] K. Sims. Aritificial Evolution for Computer Graphics. Proceedings of SIGGRAPH '91. In ACM Computer Graphics, 25, 4 (July 1991), 319-328.

[90] H. Sodeyama et al. Study of the displacement of a skier's center of gravity during a ski turn. In *International Series on Biomechanics: Proceedings of the 5th International Conference of Biomechanics*. University Park Press, 1976.

[91] A. J. Stewart and J. F. Cremer. Beyond Keyframing: An Algorithmic Approach to Animation. *Proceedings of Graphics Interface '92*, 273-281, 1992.

[92] A. F. Vakakis, J. W. Burdick, and T. K. Caughey. An Interesting Strange Attractor in the Dynamics of a Hopping Robot. *The International Journal of Robotics Research*, 10, 6, 1991, 606-618.

[93] M. van de Panne. *Motion Synthesis for Simulation-Based Animation*. M.A.Sc. Thesis, University of Toronto, Department of Electrical Engineering, Sept. 1989.

[94] M. van de Panne, E. Fiume, and Z. G. Vranesic. Reusable Motion Synthesis Using State-Space Controller. *Proceedings of SIGGRAPH '90.* In *ACM Computer Graphics*, 24, 4 (August 1990), 225-234.

[95] M. van de Panne, E. Fiume, and Z. G. Vranesic. Control Techniques for Physically-Based Animation. *Proceedings of the Third Eurographics Workshop on Animation and Simulation*, Cambridge, England, 1992.

[96] M. van de Panne. Physically-Based Turning Animations (video). Available from Dynamic Graphics Project, Dept. of Computer Science, University of Toronto, Toronto, Ontario, CANADA, M5S 1A4. 1992.

[97] M. van de Panne, E. Fiume, and Z. G. Vranesic. A Controller for the Dynamic Walk of a Biped Across Variable Terrain. *Proceedings of the 31st IEEE Conference on Decision and Control*, 2668-2673, 1992.

[98] M. van de Panne, E. Fiume, and Z. G. Vranesic. Optimal Controller Synthesis Using Approximating-Graph Dynamic Programming. *Proceedings of the American Control Conference*, San Francisco, June 2-4, 1993.

[99] M. van de Panne and E. Fiume. Sensor-Actuator Networks. To appear in *Proceedings of SIGGRAPH '93*, In *ACM Computer Graphics*, 27 (August 1993).

[100] M. van de Panne, E. Fiume, and Z. Vranesic. Physically-Based Modeling and Control of Turning. to appear in *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*

[101] M. Vukobratovic et al, *Biped Locomotion: Dynamics, Stability, Control, and Application,,* Springer Verlag, 1990.

[102] N. Wagner, M. Mulder, and M. Hsu. A Knowledge Based Control Strategy for a Biped, *IEEE Robotics and Automation International Conference*, 1988, 1520-1524.

[103] J. Wilhelms and B. Barsky. Using Dynamic Analysis for the Animation of Articulated Bodies such as Humans and Robots. *Proceedings of Graphics Interface*, May 1985, 97-104.

[104] J. Wilhelms. *Graphical Simulation of the Motion of Articulated Bodies such as Humans and Robots with Particular Emphasis on the use of Dynamic Analysis*. Doctoral dissertation, Computer Science Division, University of California, Berkeley, California, U.S.A., July 1985.

[105] J. Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. IEEE Computer Graphics and Applications, 7(6), 12-27, June 1987.

[106] J. Wilhelms and R. Skinner. An Interactive Approach to Behavioral Control. *Proceedings of Graphics Interface '89*, 1-8, 1989.

[107] D. A. Winter. *Biomechanics and Motor Control of Human Movement*, second edition, John Wiley & Son, Inc., 1990.

[108] A. Witkin and M. Kass. Spacetime Constraints. *Proceedings of SIGGRAPH '88.* In *ACM Computer Graphics*, 22, 4 (August 1988), 159-168.

[109] M. R. Yeadon. The Simulation of Aerial Movement; Parts I, II, III, and IV. *Journal of Biomechanics*, 23(1), 1990, 59-89.

[110] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 53-60, November 1982.

[111] Y. Zheng and J. Shen. Gait Synthesis for the SD-2 Biped Robot to Climb Sloping Surface. *IEEE Transactions on Robotics and Automation*, 6, 1, 1990, 86-96.