PLANNING FOR DYNAMIC MOTIONS USING A SEARCH TREE

by

Pedro S. Huang

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

PLANNING FOR DYNAMIC MOTIONS USING A SEARCH TREE

for the degree of Master of Science, 1996

by

Pedro S. Huang

Department of Computer Science

University of Toronto

Abstract

The generation of physics-based motion for articulated figures can be studied as a path-planning problem through state-space. This thesis presents an algorithm that searches for control sequences that perform a desired action. The search is accelerated by making use of evaluation functions, pruning conditions, and a memory containing successful state-action pairs. Results for various low-level control problems are demonstrated, including balance for an acrobot (a two-link robot); simultaneous balance of two inverted pendulums; hopping and flipping for the acrobot; and swing-up solutions for a double pendulum on a cart. The application of this algorithm for high-level path-planning is demonstrated using the example of a hopping lamp that traverses rugged terrains.

Acknowledgements

A well-deserved thanks goes to my supervisor, Michiel van de Panne, who has provided a wealth of stimulating ideas and energetic enthusiasm. Michiel provided the starting points for this work; his subsequent guidance and direction have been invaluable. Eugene Fiume has always been available for helpful wisdom and supportive encouragement in navigating through academia and industry. James Stewart also provided much needed help. Finally, thanks to the rest of the gang at DGP for technical help, volleyball, lunches, parties, and other good times.

My parents worked hard and sacrificed much to ensure a better life for their children in North America. Thanks, Mom and Dad!

Finally, through patient sacrifice, tireless encouragement, undying perseverance and loving companionship, my wife, Jennifer, has been instrumental in giving me the confidence to pursue my dreams.

Contents

List of Figures

Chapter 1

Introduction

The quality and demand for computer animation has experienced recent, unprecedented growth. Consider, for example, the use of computer animation techniques in the entertainment industry, such as in the 1993 release of *Jurassic Park* [SCK93], which had audiences guessing as to what was real and what was not. Disney's recently released *Toy Story* [Las95] has earned the distinction of being the first completely computer-generated, full-length feature film. Where the use of computer animation techniques was virtually non-existent a decade ago, today, it is commonplace.

The word *animate* is defined in Merriam Webster's Dictionary as: *to make or design in such a way as to create apparently spontaneous lifelike movement* [Web93]. The goal of computer animation is to use the computer as a tool to bring life to a sequence of images.

Animation techniques can be placed within a spectrum that ranges from *cartoon* at one extreme to *physically–realistic* at the other. Cartoon animation exaggerates motions in unrealistic ways with the goal of producing farcical situations. For example, characters in a cartoon world find themselves unaffected by gravity until the moment they actually become aware of being beyond the edge of a cliff. In the movie, *The Mask* [RFV94], computer animation is used to enhance the lead character's already elastic and comedic performance.

In contrast, physically–realistic animation seeks to achieve results that are virtually indistinguishable from real life. As computer graphics produces increasingly photo-realistic images, animation flaws from unnatural motions can be more readily perceived. For example, while stick figures are easily forgiven for a partially flawed motion, a photo-realistic dinosaur is expected to appear to balance and move smoothly at all times, while interact-

ing naturally with its environment. Another example is the animation of hopping animal models, in which audiences reasonably expect anticipation and follow-through for jumps. A variety of techniques has been developed to deal with the various challenges that the creation of physically–realistic animations presents.

This thesis contributes to the state of the art of *physics-based animation*. Physics-based animation addresses the challenge of realism by making use of computer simulation to enforce physical constraints and to handle object interactions. In particular, this thesis explores an algorithmic technique for the automatic synthesis of dynamic motions for various objects, especially those that involve balance or precise timing and execution of the motion in order to succeed.

1.1 Motivation

How can we animate a human model to walk or perform realistic gymnastic manoeuvres? Given an arbitrary robot configuration, how would it move? Would it be able to balance, hop, or even flip?

Our work addresses these kinds of questions. We wish to develop a novel, but general, technique that produces motion control for problems whose motions would otherwise be tedious to generate. It is assumed that feasible motions may not be known ahead of time. The design of our algorithm is motivated by the desire to incorporate several desirable properties: (1) to have a simple representation; (2) to use previous experience to help improve future performance; (3) to find novel solutions; and (4) to be efficient.

We specifically avoid the use of domain-specific knowledge or any special-case knowledge of the environment. While embedding knowledge of these features into the control scheme would undoubtedly produce better solutions for specific problems, the solutions would not necessarily generalize with parameter changes. This is particularly true for the systems that we have experimented with whose behaviour is very sensitive to small changes in the parameters and initial conditions.

Our work focuses mainly on motion control synthesis for animated characters. However, we keep in mind a broader scope that includes the verification of physical robotic design through dynamic correctness in simulation. While dynamic laws in animation can be altered to achieve a desired effect, we cannot tamper with physical laws in the context of real-world

robotics. To this end, we design our models with realistic mass and inertial parameters.

1.2 Physics-Based Animation

Given an appropriate model of the environment and a simulator, physics-based animation enforces realistic motion. The cost of this realism is the loss of fine-grained control over objects in the scene. An animator can no longer simply specify an object's trajectory or velocities over time, since those values may not be physically achievable. For example, the centre of mass of an object in flight, in the absence of any external forces except gravity, is constrained to follow a parabolic path.

While fine-grained control is necessary to achieve certain effects, an animator might prefer to let specific objects behave autonomously. This is particularly useful in a scene where many objects are interacting in a complex manner. Rather than tediously specifying the trajectories of all the objects, the animator would often prefer to focus on control of a few foreground characters, allowing the background characters to evolve in a natural manner.

Control of animated objects in a physics-based environment ultimately involves the specification over time of actuations consisting of forces and torques. Given these, the equations of motion determine resulting accelerations, which the simulator then integrates to produce velocities and positions to update the state vector.

1.3 The Problem of Control

Synthesizing the control required to produce a desired motion is a difficult problem. Consider the control that humans use to produce a walking motion. While much progress has been made in a general understanding of the principles of biped locomotion, there has been little progress made toward a controller for physics-based human locomotion that can be applied to general gaits and terrains. Yet, walking is second nature to us; conscious control is rarely required for navigating smooth terrains. In fact, the human body senses and interprets many cues from the environment, using these to orchestrate muscle activity in a manner that puts most sophisticated robotic control systems to shame.

A general solution of the control problem produces a mapping from the state of a system to actions that will produce an appropriate path to a desired goal state. In physics-based

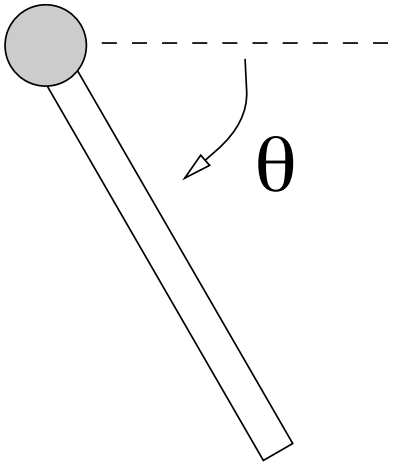

Figure 1.1: The pendulum model.

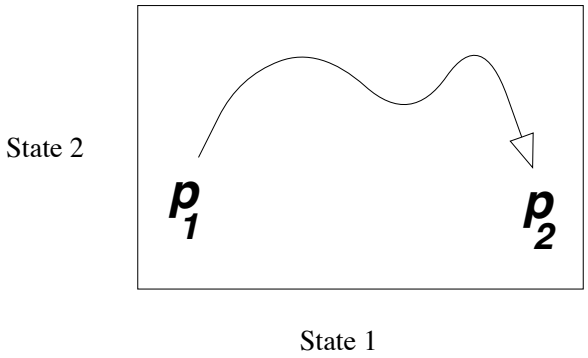

Figure 1.2: A path through state-space; p_1 and p_2 are initial and goal states.

animation, the state is described in terms of the degrees of freedom (DOFs) of the system and their derivatives. The space spanned by the DOFs and their derivatives is called the *state-space*. For example, the state of a pendulum (see Figure 1.1) could be described using the pair $< \theta, \theta' >$, where θ and θ' are respectively the hinge angle and hinge angular velocity.

The control problem can be thought of as a path-planning problem grthrough state-space as seen in Figure 1.2, which shows a two-dimensional space. A major obstacle to solving this problem is dealing with the size of the space, which grows exponentially with the number of dimensions. This is often referred to as the *curse of dimensionality*. For high-dimensional spaces, it can be difficult or altogether impossible to compute a path between points in the state-space, even when they are close together. These difficulties arise because of the inherent constraints of a physical environment such as inertia or gravity. A path between two points in state-space does not exist when no actuations exist that can control the system to go from one state to the other.

We are interested in solving a class of control problems where motions are very sensitive to the state of the system, such as balancing and hopping. The problems that we experiment with are limited to having a single control input.

1.4 Thesis Contributions

This thesis presents an algorithm which searches for a path through state-space that represents the performance of a desired motion. The planning is performed in much the same way as chess is played, building a search tree by evaluating the possible actions at each decision point. Planning is done for several stages before a specific control decision is accepted.

Experimental results from applying our search algorithm have demonstrated success in producing low-level control for various *underactuated* systems. An underactuated system has fewer actuators than it has DOFs, and such systems have been receiving growing attention in the control community [BF94]. In particular, we experiment with the *acrobot, cart-and-double-pendulum,* and *spinner* models, each of which has only a single controllable actuator. These models will be described in more detail later. Various motions are demonstrated, including balance, hop, flip, and swing-up motions.

High-level control is demonstrated by synthesizing motions which use complete jumps as control primitives. In this case, we use the algorithm to build a search tree representing concatenations of different hop-control primitives for *Luxo* the hopping lamp. We demonstrate successful path-planning for Luxo across a variety of terrains.

The search for successful motions can be accelerated by reusing prior experience. For the systems that we have studied, motions are highly sensitive to their initial conditions so that control sequences are difficult to reuse directly. This is because the exact repetition of a previous state is unlikely to ever occur. However, we demonstrate that prior experience can serve as a guide to generating control inputs that are more likely to succeed. Important accelerations of the search procedure are obtained for several of the example motions.

1.5 Thesis Organization

An outline of this thesis is presented in Figure 1.3. We begin in Chapter 2 with animation techniques and related work providing background to our work. In addition to previous work in Computer Animation, we have drawn from fields including Artificial Intelligence

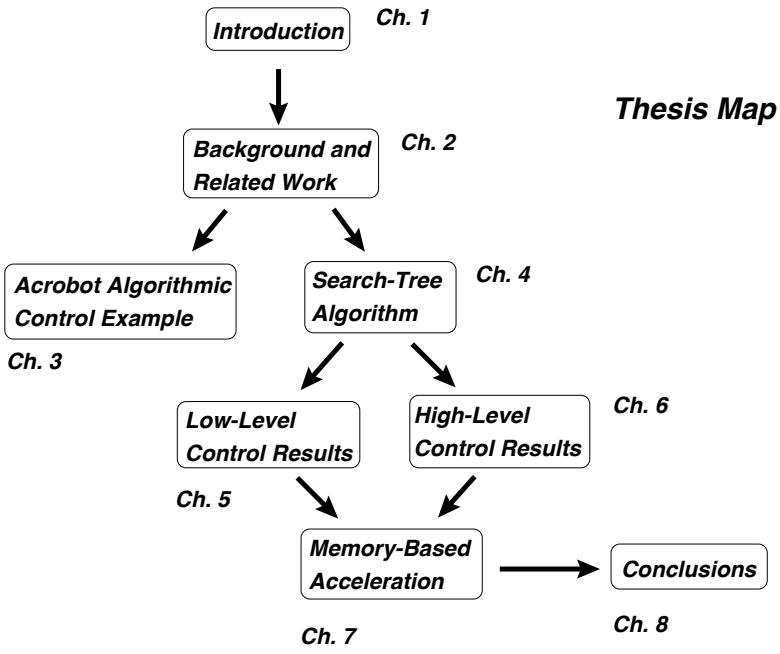

Figure 1.3: Map outlining the thesis flow.

and Control Theory. All earlier work on the acrobot example, which inspired much of the work in this thesis, originated from Control Theory.

In Chapter 3, we demonstrate a hand-tuned algorithmic controller for the acrobot. The acrobot example is chosen because it is difficult to control; it is not *a priori* clear how (or even if!) the acrobot can successfully balance or locomote using a single actuator. This example demonstrates the hand-crafting of a controller, a process which we later show can be largely automated using the search tree. Another purpose for this example is to familiarize the reader with the acrobot and its possible motions. We use lessons learned from this experience to guide the development of our general control algorithm.

The search tree algorithm is introduced and described in Chapter 4. This algorithm is applied to several models to demonstrate successful motion resulting from searching for low-level control. The experimental results are presented in Chapter 5, including low-level control of gymnastic manoeuvres, swing-ups, and balance. The algorithm is then applied to high-level control using *Luxo*, the hopping lamp. Successful locomotion planning experiments over rugged terrains are documented in Chapter 6. We return to the notion of using previous experience in Chapter 7, which demonstrates the search acceleration. Finally, we conclude in Chapter 8 with a summary of the work and of future directions that can be explored.

Chapter 2

Background and Related Work

Computer animation techniques can be broadly described as belonging to a spectrum that goes from complete animator control at one extreme to complete computer control at the other (see Figure 2.1). Associated with this continuum of control is a transition from full trajectory specification at one end to behavioural specification at the other. Different techniques are better for different applications. In practice, animation systems attempt to provide several techniques along this continuum to produce a flexible production system.

We first describe some approaches to computer animation, using the techniques of *motion capture*, *keyframe* animation, and *algorithmic* animation to illustrate examples at different points along the continuum (see Figure 2.2). Following this, physics-based animation techniques are presented. Contributions from Artificial Intelligence and Control Theory are also discussed, since they play a strong role in the design of the search algorithm described in this thesis.

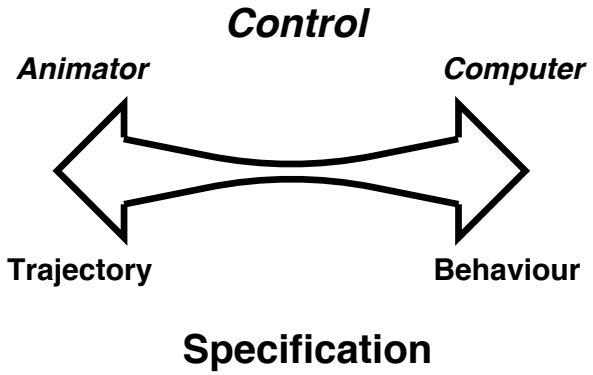

Figure 2.1: Continuum of tradeoff between animator and computer control.

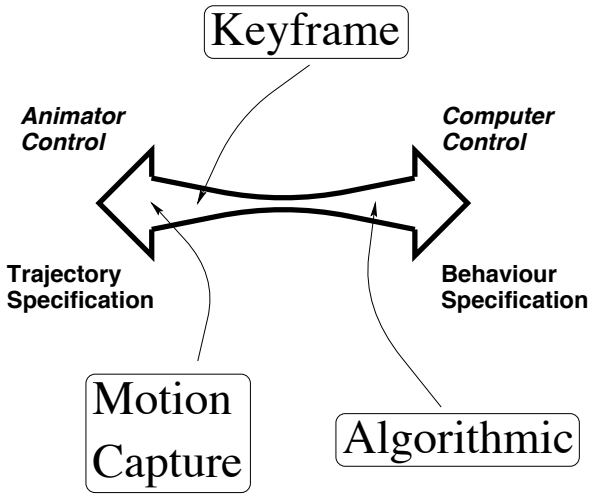

Figure 2.2: Some approaches within the control continuum.

2.1 Approaches to Animation

2.1.1 Motion Capture

Motion capture solves the problem of motion specification by capturing motion data from a physical model. For example, sensors that can detect position and orientation such as the *Flock-of-Birds*[1] or *Polhemus*[2] sensors may be attached to key positions on a human. Data for a motion is recorded as a manoeuvre is performed. This data can then be mapped onto a computer model of a human or even a completely different creature. Some manipulation of the data may be required when the features of the model do not correspond directly to the features of the original source. Editing techniques for motion capture data have recently begun to appear in the literature [WP95] [BW95].

Motion capture can be viewed as a type of puppetry, where the captured data specifies the movement of key points on the puppet. The successful use of motion capture requires that the data can be appropriately mapped to the model being animated; this constraint can limit the usefulness of this technique. Another problem is that collision and interpenetration are not handled at all, since the motion specification is directly mapped to a character in a potentially different environment.

The main advantage of motion capture is that fine details of a motion are preserved. This is important because it is easy to spot a motion that does not look right when particular

[1]Trademark of Ascension Technology
[2]Trademark of Polhemus Magnetic

visual cues that we are accustomed to seeing are absent [WP95] [BW95]. These cues may have low-frequency components corresponding to gross movements or they may have high-frequency components corresponding to subtle qualities of the motion.

2.1.2 Keyframe Animation

Keyframing is the traditional and favoured method of generating animations [FvDFH90]. This technique uses the computer as an assistant in creating the animation. The animation is specified by defining a set of key frames that describe the positions of the characters at certain *key* instants in time. The responsibility for creating these keyframes is usually given to skillful and experienced artists who must visualize the desired motion and make decisions about overall appearance. The keyframes are accompanied by directions for the creation of intermediate frames. The process of interpolating in between these keyframes is appropriately called *inbetweening*. Using this approach, all character interactions with the environment are controlled by the animator. Thus, the quality of the results depends entirely upon the skill of the animators. Ensuring realistic motion is both difficult and tedious.

Computer-based keyframe tools have eliminated some of the tedium in creating animations. The use of techniques such as spline interpolation has assisted in improving the task of inbetweening both in terms of speed and quality. Splines are used because of their well-defined continuity properties and of their intuitive manipulation through the positioning of a relatively small number of control points.

2.1.3 Algorithmic Animation

Algorithmic animation uses a model and a set of rules to directly generate motion from a set of initial conditions [HSL92] [SC92]. The motion behaviour is preprogrammed, usually using some sort of procedural or state-machine representation. While this approach saves the animator from having to specify a motion in detail, the cost is the loss of direct animator control over the motion.

Taken to its extreme, objects may be programmed with completely autonomous behaviours. Different animation sequences may be generated by simply choosing different initial conditions for the characters and allowing them play out their parts. Characters may have sensors and behavioural hierarchies that allow them to interact with and respond

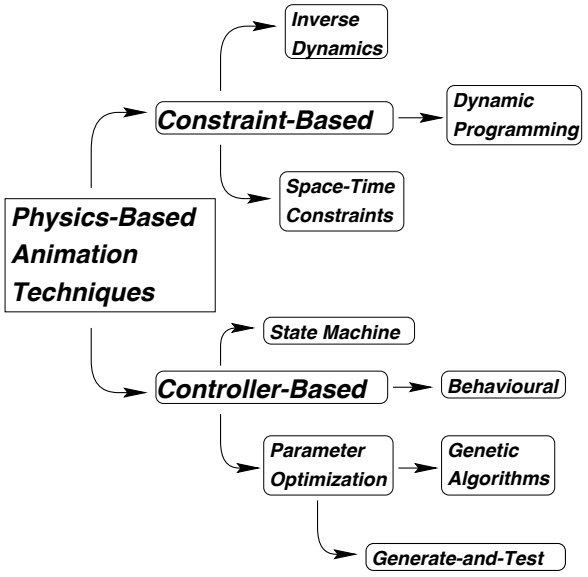

Figure 2.3: Physics-based animation techniques.

to their environment.

Physics-based animation is an instance of algorithmic animation. The objects begin with a chosen set of initial conditions. Actuators are used to move the objects, and the laws of physics act as constraints on the behaviour of the objects.

2.2 Physics-Based Animation Techniques

We broadly categorize physics-based animation approaches as either *constraint-based* or *controller-based* (see Figure 2.3). Constraint techniques solve for the actuations required to move an object from an initial state to a goal state, while including physics as a constraint to be satisfied. Controller-based techniques use a direct simulation to implement the laws of physics, but require the synthesis of a controller to apply the internal actuating forces. These controllers are often optimized for a particular goal, such as speed of locomotion or height of jump.

2.2.1 Constraint-Based Physical Animation

Space-Time Constraints

Witkin and Kass [WK88] decompose a desired motion path into a sequence of boundary conditions analogous to keyframes. Constraints in both space and time are used to solve a

sequence of two-point boundary problems for the required actuations. This technique, called *space-time constraints*, is used to animate a jumping sequence for a 2D, 3-link articulated model of an animate lamp (henceforth referred to simply as *Luxo*).

Inverse Dynamics

Inverse dynamics takes a set of control variables and maps them onto the forces and torques needed to control an object. Usually, this mapping is done by solving a system of equations numerically. Thus, constraints on the control variables for the motion can be programmed algorithmically and inverse dynamics can be used to compute an appropriate set of necessary actuations. Stewart and Cremer [SC92] use this technique to "program" a human figure to walk on level ground and to climb staircases in a realistic manner.

Dynamic Programming

Van de Panne, Fiume, and Vranesic [vFV90] solve the motion control problem in a reusable manner and treat the goal state as a single boundary condition. First, a cell-based discretization of state-space is performed. This is followed by a procedure which propagates control solutions from the goal state to all other state-space cells using dynamic programming. Finding the path to the goal state produces the required sequence of control actuations.

2.2.2 Controller-Based Physical Animation

Rather than describing a motion through a direct specification of the motion trajectory, a *motion controller* operates in a manner more akin to the way things move in the real world, namely by generating appropriate internal forces for the "muscles" of our animated character.

State-Machine Controllers

State-Machine controllers are designed to take advantage of features of a system that permit the decomposition of motion control into a set of separate control laws. Control decisions are based not only on the current state of the system, but also on the active control law. Rules to transition from one control law to another are usually based on the state of the system and on the amount of time that the control law has been active. The parameters of the controller are often manually tuned.

Raibert tackled the challenge of controlling running motions for creatures with light-weight legs in robotics [Rai85]. Their strategy for control decomposes the controller into three distinct and separable components: (1) height control, (2) attitude control, and (3) speed control. Hodgins and Raibert [HR90] extend this work on legged controllers to produce a control system that performs biped gymnastics in simulation. In this case the takeoff, flight, and landing are treated as distinct control phases. In [RH91], Raibert and Hodgins demonstrate a variety of gaits for monopeds, bipeds, and quadrupeds.

Hodgins, Wooten, et al [WH94, HWBO95] recently produced realistic simulations of human diving and running using a detailed human model and a tuned state-based controller. However, the controllers are specific to the parameters of the model being animated and require much tuning.

Parameter Optimization

An alternative to the hand-design of a controller is the automated optimization of control histories or of a set of controller parameters. The control histories can be optimized using methods such as nonlinear programming, gradient-descent, and simulated annealing.

From the biomechanical literature, Pandy [PAH92] parameterizes the control history of a nonlinear dynamical system using a set of nodal points. This control history is optimized using nonlinear programming methods. Pandy demonstrates his results by optimizing the height for human jumping using a musculoskeletal model.

Van de Panne and Fiume [vF93] search the space of possible controllers by first generating and testing for viable motions, then optimizing the controller parameters. They introduced sensor-actuator networks (SANs) as a controller representation with the goal of achieving *closed-loop* control. Closed-loop control, in contrast to *open-loop* control, uses feedback from the environment in making a control decision. A SAN is a non-linear network of weighted connections from sensors to actuators. Random weights are initially assigned and the ones which produce desirable motions are kept and optimized.

Van de Panne, Kim, and Fiume [vKF94] study the limits of open-loop, cyclic motion controllers for producing locomotion in their work with "virtual wind-up toys." Using the classic spring-driven wind-up toy as a model, the motion controller loops through a repeated sequence of actuations. This approach is used to synthesize bounding, hopping, and crawling motions for creatures whose locomotion modes were not known in advance.

Grzeszczuk and Terzopoulos [GT95] synthesize realistic locomotion for the animation of deformable physics-based snakes, dolphins, and sharks. For each actuator in the creature, the actuation sequence is transformed into the Fourier domain in order to provide a more compact representation. To derive the control required for a specified motion sequence, a hierarchical control scheme is used that searches for optimal concatenations of the learned motion sequences.

Ngo and Marks [NM93] use a massively parallel genetic algorithm (GA) to search for a motion controller in which the controller is represented by a bank of stimulus-response rules. Natural selection is simulated by a GA using the principles of crossover and mutation on the components of the controller. Those controllers that are judged "fit" according to a selection criteria are added to the "gene pool," while those judged less "fit" are selected out. Experiments were carried out on several different configurations of multi-link creatures.

Sims [Sim94] uses *genetic algorithms* to evolve virtual creatures that were selected based on an optimization goal. With his creatures, the configuration of the creature and the controller were candidate parameters for optimization. The resulting behaviours demonstrated interesting ways of solving for the specific tasks presented.

Although parameter optimization methods mentioned have produced excellent animation results, their usefulness is limited to systems where useful regions of the controller space can be readily found and search progress is easily defined. This is partly due to the type of algorithms that are used for optimization. When a controller fails to successfully complete a motion, the cause of the failure is usually not identified, but rather the entire set of parameters is penalized. In many cases, partial progress toward a solution should be rewarded and the parameters that led to failure should be identified so that a more directed search can be performed.

Van de Panne, Fiume, and Vranesic [vFV93] use a search algorithm to plan turning motions for bicyclists and skiers. Inverse kinematics are used to dress up the motion appropriately. The search algorithm we present in this thesis is loosely related to this work.

Behavioural Control

Behavioural control endows virtual creatures with built-in stimulus-response systems. On the extreme automated end of the animator/computer control continuum, these systems produce animation sequences in which the creatures are only controlled by changes to the

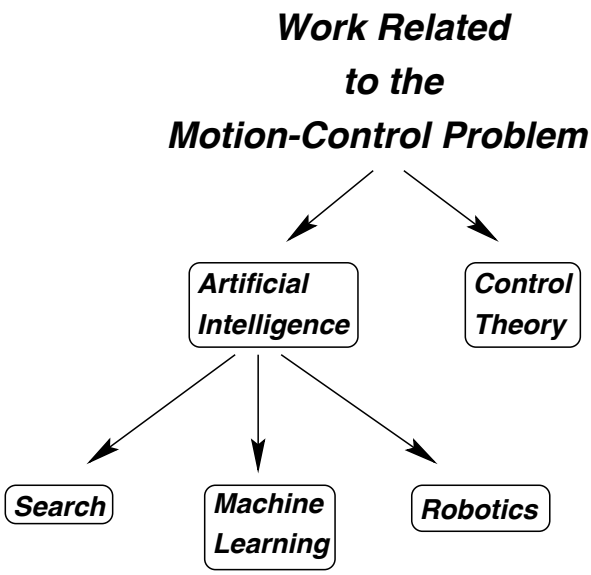

Figure 2.4: Related work from other fields.

environment that the creature can sense.

Wilhelms and Skinner [WS89] enumerate a set of stimulus-response behaviours that control the behaviours of objects that must navigate a virtual environment with obstacles that produce different stimuli. The strengths of the stimuli can obey arbitrary distance laws, such as the typical inverse-squared proportionality.

The artificial life of Tu, Terzopoulos, and Grzeszczuk [TT94] [TTG94] models an undersea ecosystem where both prey and predator face the daily realities and dangers of marine life. The behavioural model encapsulates a hierarchy of needs, which is used to process the sensory data to resolve conflicting goals. Modeled behaviours include target-following, schooling, mating, and obstacle-avoidance.

2.3 Related Work from Other Fields

Besides computer animation, other fields (see Figure 2.4) have influenced the design of the algorithm that we present in this thesis. In particular, Artificial Intelligence contributes search, machine learning, and robotics techniques. Earlier work on controlling the acrobot model is from Control Theory. As the bodies of literature in these fields is quite extensive, we cite only several examples immediately relevant to the work of this thesis.

2.3.1 Artificial Intelligence

Search

Tree-based search techniques [Win84], also known as *decision-trees*, are popular for solving problems where a series of decisions must be made. For example, the options available to a chess player can be represented by a tree whose root node represents the current state of the game. The problem is to search for the correct branch to take that will ultimately optimize the player's game. In our algorithm, a tree node represents the state of the dynamical system to be controlled. The edges of the tree represent choices of control inputs.

Machine Learning

Machine learning encodes experience for reuse, forming a behavioural model. This experience can later be queried when choosing an action. For example, a chess program may, over the course of several games, store board patterns that eventually lead to a win. When a familiar pattern is recognized, the program may attempt to make similar decisions. Alternately, the program may avoid making similar decisions for patterns that lead to a loss.

Atkeson [Atk91] describes a local regression method for robot learning which extrapolates actions that can be represented by a continuous-valued variable. This method optimizes the similarity metric of nearest neighbours in order to refine robotic memory-based control. Lowe [Low93] describes a variable-kernel similarity metric method which optimizes the local model for a binary output.

Reinforcement learning attempts to assign credit for actions leading to success or failure. This is intended to reinforce or discourage certain actions. Sutton and Barto [BSA83] [Sut91] demonstrate the effectiveness of applying reinforcement learning to cart-pole balance and other problems. Sutton explores issues such as the credit-assignment problem (assigning credit or blame to a series of actions which lead to a success or failure) and the addition of adaptive critic elements. These latter elements use outcomes to reward earlier predictions of success or failure.

Robotics

Khatib [Lat91] proposes the use of potential-field methods to guide robots during on-line operation. Objects that are to be avoided produce a positive field while the goal projects

a negative field. The robot can use this field to perform navigation by traveling in the direction of most-decreasing gradient.

A problem with pure potential-field methods is that complex environments will often be riddled with potential wells that are only locally minimal. This is the same local optimization problem encountered when using any greedy algorithm. Randomized path planning [Lat91] is a technique used to attempt to avoid being trapped in local minima.

2.3.2 Control Theory

One of the motivations for the work in this thesis is to tackle a problem which is receiving attention within the control community, namely that of controlling *underactuated* mechanisms. An underactuated mechanism is one which has fewer actuators than degrees of freedom. These systems are particularly interesting because they are difficult to control. They are non-linear and, like many animated objects, rely on accumulating appropriate momenta to perform a desired action.

As an extreme example, we consider the *acrobot*, the jointed robot shown in Figure 3.1. Figures such as the acrobot, which are composed of links that are connected by joints, are known as *articulated*. In particular, the acrobot has a single actuator at the joint connecting its two links.

We work with the acrobot for two reasons. First, articulated figures such as humans are effectively underactuated because ankles only provide limited control over the global orientation of the body. Second, the acrobot serves as an example of an articulated figure that is difficult to control despite having very few DOFs. We propose that the number of DOFs is not always a good measure of the difficulty of a control problem. Other factors, such as the size of acceptable regions of state-space and the connectivity of these regions through control inputs, play a major role in determining the overall difficulty of an action.

Earlier Work on the Acrobot

The work of Hauser and Murray [HM90] first proposes a controller which makes an acrobot balance in place. In this case, the acrobot's foot is fixed to the ground using a frictionless, unactuated hinge. Bortoff and Spong [Bor92] [BS92] use approximate linearization to produce slow movements among stable configurations. Their method is also shown to work on a real acrobot system.

Berkemeier and Fearing [BF92] attempt further types of motion with the acrobot and arrive at a controller capable of balancing, sliding, and hopping, subject to some constraints on the acrobot's physical design. The control for both balancing and sliding is successfully tested [BF94] using an inclined mechanism to reduce the effects of gravity. The hop is not implemented due to technical difficulties.

Chapter 3

A Control Example: The Acrobot

Before we present our proposed general solution for certain control problems in animation, we motivate the need for such solutions using a typical control problem that we first solve using a state-machine controller. In particular, we work with finding a hopping motion for the acrobot, which is shown in Figure 3.1. The acrobot is equipped with only a single actuator, making it an *underactuated* mechanism, which means that there are fewer actuators than there are degrees of freedom. This single actuator is located at P_2 and effects a torque between the two links. The end of the bottom link, called the *foot*, is completely free to translate and rotate.

This case study demonstrates several points which are critical to the development of our search algorithm. First, hand-crafted controllers can require large amounts of effort to tune and they often rely on cyclical behaviour. We use both manual and automated tuning to arrive at experimentally robust control. Second, tuned controllers can fail with small changes in the parameters or in the environment. Since control laws can have a limited range of applicability, once the state of the system falls outside the controllable domain, the controller is likely to fail. Finally, even when goals for the system such as balance may be intuitive, the encoding of these goals into control laws may require expertise that animators may not possess or even care to learn.

3.1 The Acrobot Control Problem

The main difference between the acrobot and Raibert's well-known monoped hopper [Rai85] is in the number of actuated degrees of freedom. The acrobot controls for both attitude and

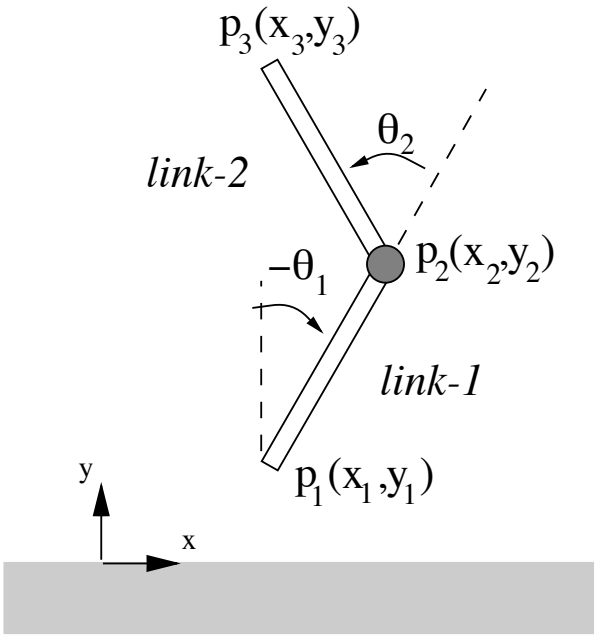

Figure 3.1: The two-link robot, displaying positional state components.

upward acceleration using a single actuator. Successful control of this mechanism must rely on building up appropriate momenta in earlier control stages. In contrast, Raibert's hopper uses two independent actuators; one controls for the orientation of the hip joint while the other controls for the thrust of the leg.

Since it is not immediately evident how a hopping motion for the acrobot can be performed, it is likely to be difficult for an animator to visualize and draw a set of keyframes that will realize a physical motion. An open-loop controller is unlikely to be effective since small perturbations in control actions have a major impact on the state of the system over time. It is unclear and unlikely that open-loop control can attract the motion into a stable cycle.

This problem can be solved by resorting to a state-machine solution, which is a fairly common control structure used to solve legged locomotion problems [HR90] [RH91]. The two phases of the state machine encode control laws for (1) balance and (2) upward acceleration. Testing the height of the acrobot head within each control phase will provide the means of switching between these two control laws.

The controller presented here cannot be proven robust in any rigorous sense. However, the results produced through experimental simulations are found to work well and they produce visually convincing motions. For the purpose of animation, this visual realism is

k_{p_1}	$8000N/m$
k_{d_1}	$800Ns/m$
k_{p_2}	$400N/m$
k_{d_2}	$40Ns/m$
h_1	$0.90m$
h_2	$0.95m$
x_{offset}	$-0.02m$
m_1, m_2	$10kg$
I_1, I_2	$0.8333kg\ s/m^2$
t_{sim}	$0.001s$

Figure 3.2: Simulation Parameters

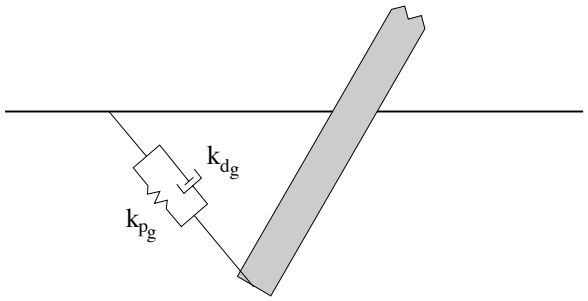

Figure 3.3: Simulation of ground contact.

what really counts.

3.2 Acrobot Simulation Parameters

The physical model parameters of the acrobot are given Figure 3.2. The acrobot has two links with a single actuator placed at the connecting joint P_2, effecting a torque between the two links as shown in Figure 3.1. Point P_1 will be referred to as the *foot* and point P_3 as the *head*.

The dynamics simulation uses a penalty method to model the ground contact; a stiff spring-and-damper system applies a force to pull the foot back towards the initial point of ground contact, as shown in Figure 3.3, using $k_{p_g} = 14000N/m$ and $k_{d_g} = 800Ns/m$. The compliance modelled by the spring-and-damper system can be equivalently considered to be a property of the floor or of the foot of the model.

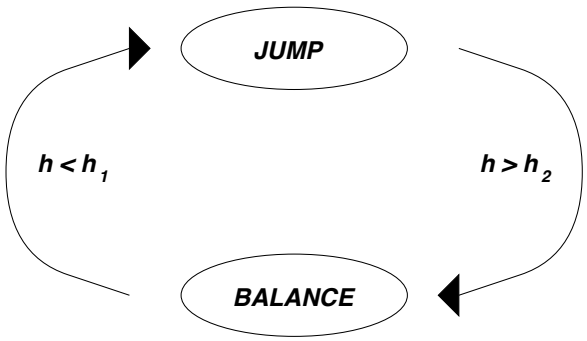

Figure 3.4: FSM for acrobot hopping, where h is the height of the 'head'.

3.3 The Hopping Problem

The *hopping* problem is to generate stable control for the acrobot to move in a desired direction across a terrain. The only allowable contact point with the ground is the foot. The difficulty in generating control results from the inherent instability of the acrobot.

In order to be convinced of the difficulty of the resulting control problem, we encourage the reader to try the following experiment. While standing upright, place all the weight on the heels and attempt to maintain balance by bending only at the waist. Performing flips while maintaining balance before and afterwards is predictably even more challenging, although we exempt the reader from attempting this manoeuvre!

3.4 The Hybrid Controller

A hybrid controller is one in which the high-level control is discrete and the low-level control is continuous. We derive a hybrid controller of two states and a means of switching between them, with the result of producing a stable hopping motion. Each state has a continuous control law that produces a control action for any given configuration of the acrobot. The basic two-state structure is shown in Figure 3.4. We first describe the control laws in effect for each state and then describe the strategy for switching between them that provides a successful and stable hopping motion as sketched in Figure 3.5.

3.4.1 Balance Control

When the acrobot is not in flight, applying a torque to the actuator will cause a reaction force to act on the foot. We would like to control for the horizontal component of this force,

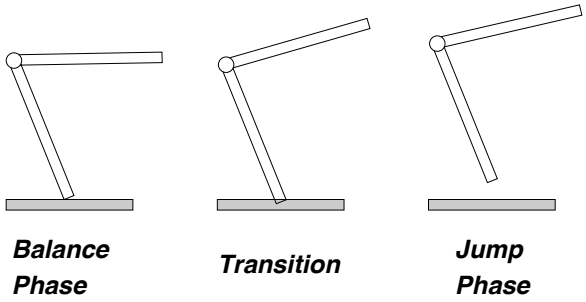

Figure 3.5: Sketch of acrobot hop showing the transition from balance to flight.

since we can use this to keep the acrobot's centre of mass above the supporting foot. We ignore, for now, the effects on the angular momentum of the acrobot.

Thus, to drive the x-coordinate of the centre of mass to zero (relative to the foot), the balance control law must calculate an appropriate actuator torque τ. This can be done by making the horizontal component of the net external reaction force, f_x, drive the centre of mass appropriately. A simple PD controller suffices for this:

$$f_x \quad = \quad -k_{p_1}\bar{x} - k_{d_1}\dot{\bar{x}},$$

where k_{p_1} and k_{d_1} are the PD constants, $\bar{x}$ is the location of the centre of mass with respect to the foot, and $\dot{\bar{x}}$ is its time derivative. The desired reaction force is proportional to the applied torque and can be expressed with the relation $f_x \; \alpha \; \tau cos(\theta_1)$, as shown in Figure 3.6. The balance control law thus becomes

$$\tau \quad = \quad (-k_{p_1}\bar{x} - k_{d_1}\dot{\bar{x}})/cos(\theta_1).$$

The balance control law is not capable of bringing the acrobot to a motionless balanced state. This is because the expression ignores the angular momentum of the changing configuration and so does not attempt to dampen it. In fact, a few attempts were made at dampening the angular momentum; they failed because they conflicted with the more important goal of balance.

The control law does, however, attract the acrobot from an initial unstable state onto a stable limit cycle. Figure 3.7 illustrates this using a phase portrait of θ_1 and $\dot{\theta_1}$. The magnitude of the limit oscillation is proportional to the initial disturbance of the centre of mass. This control law works best with a zero damping constant, k_{d_1}.

This control law can be employed to yield a regulated, stable limit cycle for the acrobot. To employ it as a part of the hopping strategy, however, the parameters of the controller

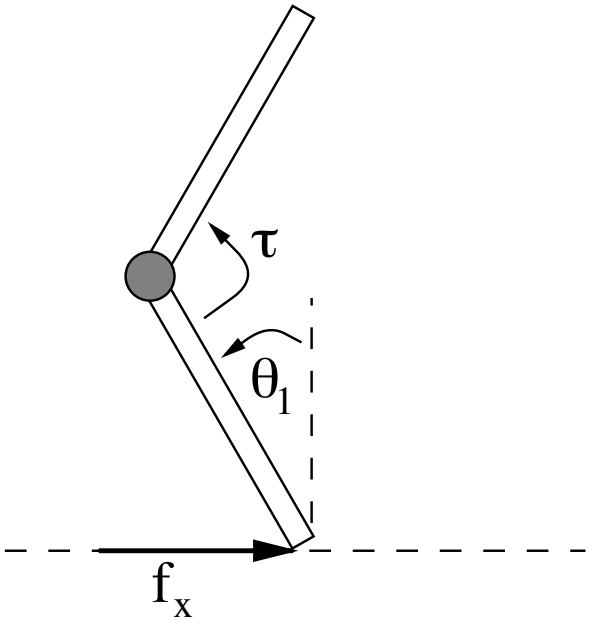

Figure 3.6: Operation of balance control.

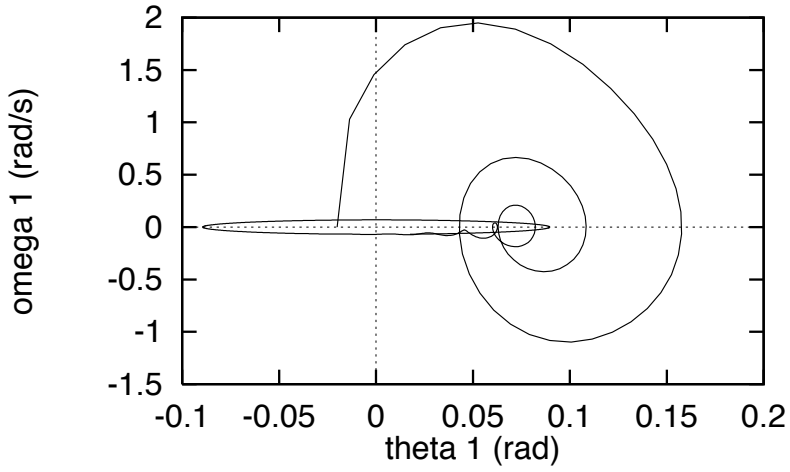

Figure 3.7: Convergence to a balancing limit cycle.

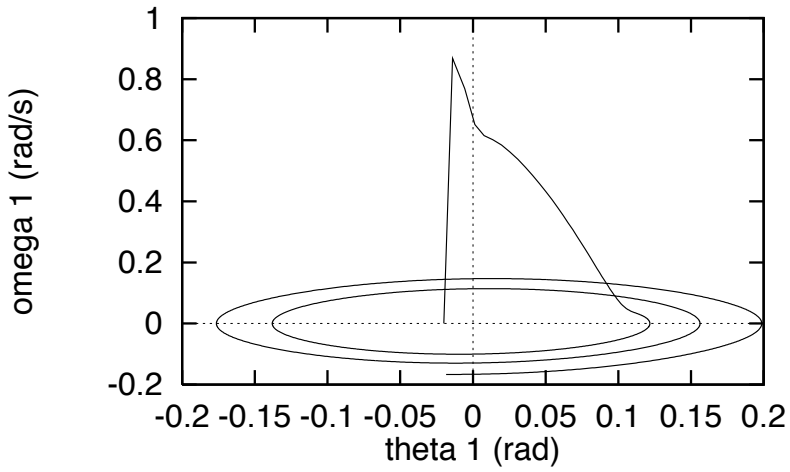

Figure 3.8: An unstable limit cycle used to induce the transition to the jumping state.

are chosen so as to lead to a slow destabilizing oscillation in order to eventually trigger the transition to the jump state. This will be described in greater detail shortly.

In order to trigger hopping, we shall choose to obtain oscillations of increasing amplitude over time as shown in Figure 3.8. Increasing the value of k_{d_1} has the effect of increasing the limit cycle amplitude over time, such as in Figure 3.8. The initial conditions and controller parameters are identical for Figures 3.7 and 3.8, except for k_{d_1} ($k_{d_1} = 0$ for Figure 3.7, $k_{d_1} = 800$ in Figure 3.8). However, these oscillations by themselves will not directly perform the hop. After attaining a critical amplitude, the jump state is entered. In the next section, we describe the control law used for this jump state. This is followed with a more precise description of the rules used for changing states in the hybrid controller.

3.4.2 Jump Control

The job of the jump controller is to cause a rapid extension motion of short duration in order to 'kick' the acrobot into the air. The control law for the jump state is a PD controller which attempts to straighten the acrobot. The desired expression is

$$\tau = -k_{p_2}\theta_2 - k_{d_2}\dot{\theta}_2, \tag{3.1}$$

where k_{p_2} and k_{d_2} are PD constants and θ_2 is measured as shown in Figure 3.1.

3.4.3 Switching Behaviour

The last element of the hybrid controller description is the set of rules which governs transitions between the balance and jump states. The transition to the jump state is made whenever the 'head' of the acrobot (p_3 in Figure 3.1) dips below a fixed height h_1. Similarly, the transition back to the balance state occurs when the 'head' rises above a fixed height h_2. The net effect of the jump-state control is to initiate a hop by forcing an extension of the acrobot, while the hop is completed using the balance state. Since the acrobot concludes the flight phase of its motion in the balance-control state, the foot is controlled to centre itself approximately underneath the centre of mass. This is necessary to achieve a stable landing.

To control the speed and direction of the hopping motion, one can use a simple trick to 'fool' the balance control into thinking its centre of mass is slightly offset from its true position. This will cause the acrobot to 'lean' forward or backward, depending on the direction that we wish it to jump. The magnitude of this offset controls the magnitude of the lean and thus the size of the jump. The following modification is made to the balance control:

$$f_x = (-k_{p_1}(\bar{x} - x_{offset}) - k_{d_1}\dot{\bar{x}})/cos(\theta_1), \tag{3.2}$$

where x_{offset} is used to alter the effective centre of mass displacement. A hopping motion to the right is obtained using $x_{offset} < 0$, thus causing the balance control law to keep the acrobot off-balance to the right. The magnitude of x_{offset} controls the speed of the locomotion in a roughly linear manner as shown in Figure 3.12.

Figure 3.9 illustrates the cyclic motion of the head of the acrobot during hopping, using a phase portrait. The trigger points used in this case, $h_1 = 0.90$ and $h_2 = 0.95$ are evident. The discrete nature of the simulation means that the transition is effected with a slight delay. The path traced by a single period of the motion in its stable limit is known as a *limit cycle*. The existence of this limit is significant for control in that configurations of the acrobot which are close to the limit cycle can be attracted to approach this path over time. Periodicity of motion also guarantees the perpetuation of the motion, which is important if the animated figure is to exist autonomously.

As with many other control algorithms, there are several parameters which have an important effect on the performance of the hybrid controller. Many of these 'magic' pa-

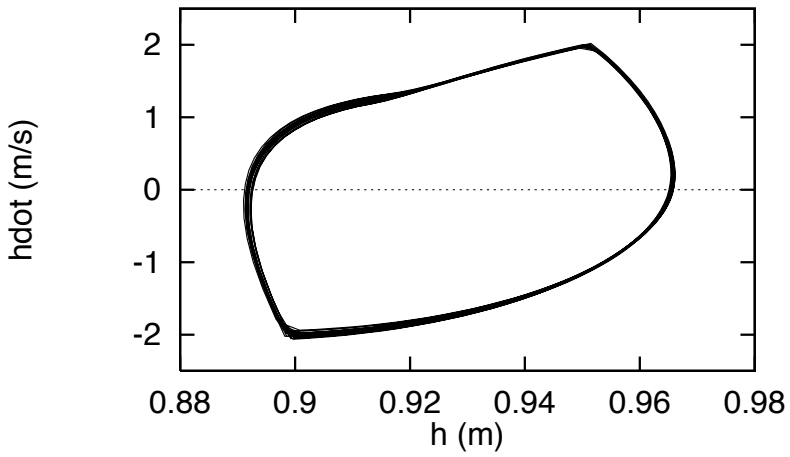

Figure 3.9: Limit cycle for the acrobot 'head', with state switches at y=0.90 and y=0.95.

rameters have been hand-tuned, while others are useful in that they provide motions with different features. These effects are discussed in the next section, along with a presentation of other results.

3.5 Results

The parameters used in the simulations (unless otherwise indicated) are given earlier in Figure 3.2. Although the k_{p_1} and k_{d_1} values appear high at first glance, they are applied with respect to the offset in the centre of mass, which is usually quite small. Experiments were performed in which the torque was clamped to $\pm 300 Nm$ with little difference in the resulting performance.

The parameters were determined by first selecting some arbitrary but reasonable parameters for the model, then tuning the other parameters as necessary. In particular, the mass and inertial parameters of the acrobot were first defined. The spring-damper system that models ground-contact required some coarse tuning to provide a reasonable 'bounce' that was neither overly 'springy' nor fully absorbed. Finally, the other parameters were hand-tuned through trial-and-error. The ratio between the k_p and k_d constants was fixed to simplify the tuning process.

Using the nominal parameters of Figure 3.2, the hopping behaviour shown in Figure 3.10 is obtained for motion across a flat terrain. From top to bottom (reading from the extreme right of the plot), the variables represent x, θ_2, y, θ_1. The units of x and y are metres, while the units of θ_1 and θ_2 are radians. For damping constants in the range $105 \leq k_{d_1} \leq 185$,

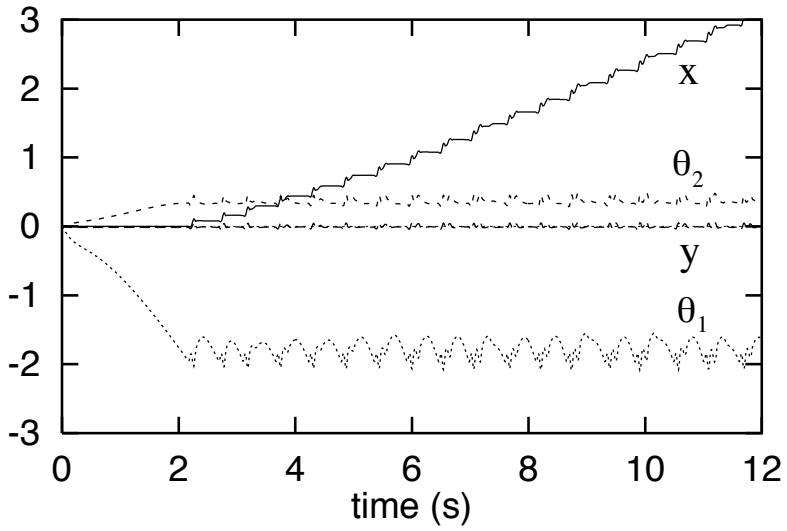

Figure 3.10: Selected acrobot state variables.

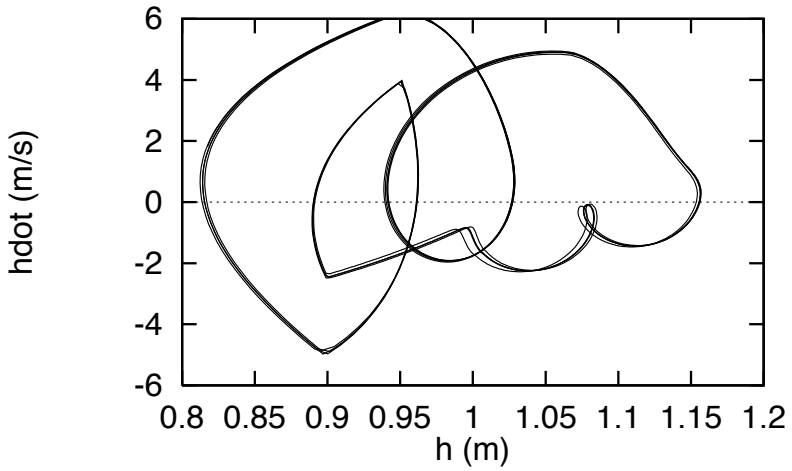

Figure 3.11: A double-period motion resulting after a bifurcation.

a steady-state limit cycle is obtained. Higher values lead to bifurcations in the limit cycle, as shown in Figure 3.11. A stable but non-repetitive motion occurs for $0 \leq k_{d_1} < 105$. The hopping motion fails (i.e. the acrobot falls over) for $k_{d_1} \geq 185$. This behaviour is a typical feature of controllers. That is, controllers tend to operate well within a certain range of parameters for the model and for the external environmental parameters.

The speed of the gait is a smooth function of x_{offset}, as shown in Figure 3.12. Note that the sign of x_{offset} determines the direction of the hopping motion. The value of x_{offset} can be changed in a continuous fashion to provide velocity control. The hybrid controller is robust enough to allow for locomotion up and down modest inclines and across variable terrain, as illustrated in Figures 3.13 and 3.14. These figures show the trajectory of the

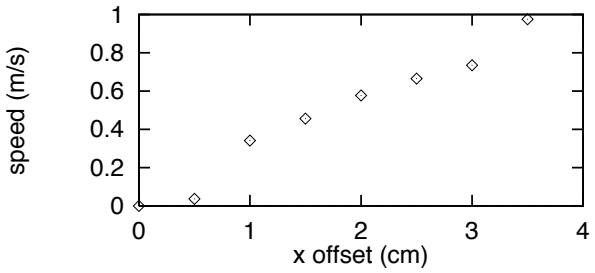

Figure 3.12: Speed control.

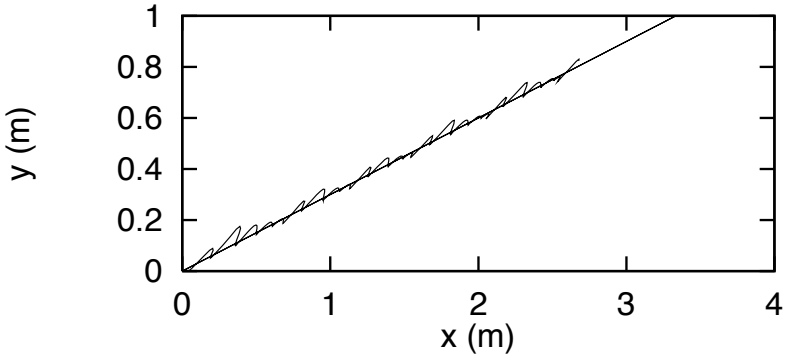

Figure 3.13: Climbing an incline (slope=0.3).

foot of the acrobot as it hops over the terrain.

The hybrid controller works well for a wide range of choices for h_1 and h_2. If we relate h_2 and h_1 by $h_2 - h_1 = 0.10m$, the hybrid controller works for $0.55m \leq h_1 \leq 1.85m$. Stable hopping locomotion can thus be obtained for both a very extended acrobot (small θ_2) or a very compact, folded acrobot (large θ_2).

Experiments also show that the hybrid controller is effective in controlling acrobots with different masses and moments of inertia. This demonstrates the robustness of the controller, which is a desirable control characteristic. The constants k_{p_1}, k_{d_1}, k_{p_2}, k_{d_2}, k_{p_s}, and k_{d_s} need to be scaled by the same factor as the mass scale factor, but no other changes are

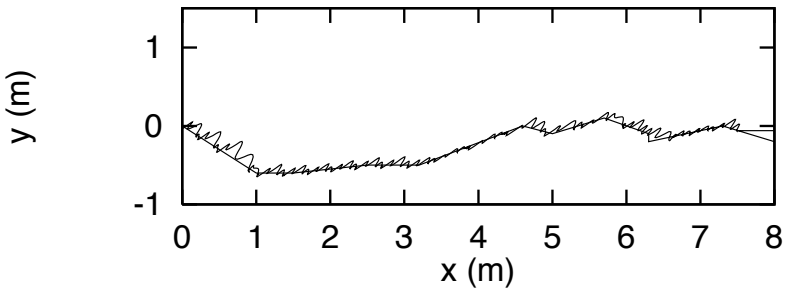

Figure 3.14: Locomotion across variable terrain.

required.

3.6 Conclusions

A new hybrid control strategy for controlling acrobot locomotion has been developed and tested in simulation. The successful state-machine controller encodes the intuitively desirable goals of balance and upward acceleration into two separate control laws. The transition between these two laws is triggered by testing the height of the head. Once the parameters of the model and controller are tuned, the resulting behaviour is experimentally robust. These parameters have a limited range of stability and are specific to the configuration of the acrobot. In simulation experiments, this controller is speed-controllable and is capable of ascending and descending various slopes and navigating various terrains. The results show that hybrid controllers can in this case provide a simple solution to an otherwise difficult control problem.

This case study using the acrobot model demonstrates that control can be difficult, even for figures with a small number of degrees of freedom [HM90] [Bor92] [BF92]. The hybrid controller that is developed for the hopping motion effectively accomplishes the specific task for which it is designed. Time, expertise and even some luck were required to properly craft such a controller. However, an animator may not necessarily have this time, expertise, or even the inclination for designing, let alone tuning such a controller.

Learning from this example, we would like to develop a general control strategy that does not require fine tuning of the controller parameters. This strategy should operate across a large domain of configurations. The encoding of motions goals should be intuitive and should not require carefully crafted control laws. Such a strategy is proposed and evaluated in the chapters following. We will use an automated search for dynamic motions that satisfy specified goals and constraints. Since many different motions are possible, the search will discover different motions that are both valid and interesting.

Chapter 4

The Search Algorithm

The approach we propose for control problems in animation is reminiscent of early search algorithms such as those developed for playing chess. At any given point in this game, a choice must be made from among a set of candidate moves. Moves require careful consideration because of their potential impact on the development of the remaining game. Most computer chess-playing algorithms perform a 'look-ahead' search through candidate sequences of decisions. Because of the complexity of chess, algorithms must rely on human-supplied evaluation metrics or on experience gained using machine-learning techniques. Our proposed algorithm is also similar in this respect.

4.1 Decision Trees

The sequence of decisions made by a chess player can be represented in the form of a tree, which is a directed graph rooted at a single node corresponding to the initial state. In this tree, each node represents a board configuration. The edges connecting a parent to a child node define a move from one state to another.

Because of the exponential growth of candidate moves in relation to the depth of the look-ahead, an exhaustive enumeration of the tree is impractical for even small depths. As a result, search *heuristics* are used to guide the process. A heuristic is a technique used to evaluate several alternative solutions. An *evaluation function* aids in this decision process. This function is applied to the nodes of the tree, providing a metric which can be used to compare different board positions. Nodes which clearly lead to failure, such as imminent check-mate against the player, can be *pruned* or eliminated from consideration. A node

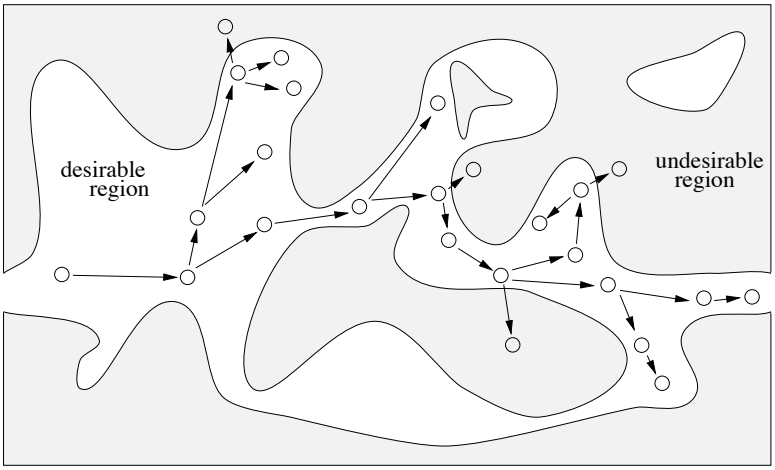

Figure 4.1: Building a state-space trajectory using a tree search.

which is promising can be *extended* by enumerating some or all of its children. When a path that is being explored appears to lead to a dead-end, *back-tracking* occurs, which means tracing backward to a different part of the tree and searching from that point.

A good introduction to search trees and heuristics can be found in [Win84]. We propose to use a modified form of the best-first search tree. This strategy chooses always to extend the node which has the current highest value of the evaluation function.

4.2 The Search Algorithm

4.2.1 General Description

We treat the search space of our problem as a set of sequential control decisions that are made at discrete instants in time. Since each sequence begins from the same initial state conditions, the set of control decisions can be represented by a tree structure such as shown in Figure 4.1. The nodes of the tree are generated through a forward dynamic simulation resulting from the application of a control action, represented by the directed edge from the parent to the child. The control problem is then reduced to producing an algorithm that can efficiently generate and search through this tree. Equivalently, we seek to find the control required to drive a system through a traversal of state-space, guided using an evaluation function, and made efficient by the pruning of actions leading to undesirable configurations.

Just as a new search tree is built for each move in a chess game, we generate a search tree

for each action. Also, as in chess, we are able to reuse relevant branches of the computed search tree from an earlier phase. To make a decision about any given action, our algorithm uses a modified version of the best-first search tree.

Two choices must be made whenever carrying out a simulation to further expand the search tree. First, a starting node must be selected. This should presumably be one which looks like it is part of a promising motion, so we shall define an evaluation function v to quantify how promising any given node is. To prevent the same node from being selected continually, v normally has a term which penalizes for the number of node children. Thus, a node will only be extended a certain number of times before backtracking will occur. Second, we need a method of choosing the control input u to apply. The simplest method is one of stochastically selecting the control input from a fixed uniform distribution function, although the form of the control input generator is left to the user to define.

4.2.2 Simplifications

In our representation of the search space, we have made a series of simplifications to the problem. First, we have restricted control to a sequence of actions that are applied for a fixed time duration. Since we place no restriction on the form of the control action, this representation does not limit the generality of the solution. However, in practice, this representation can make the search for timing-critical actions more difficult.

Another simplification is that we do not attempt to choose among all possible control actions at each node, since the number of possible actions is large and often even continuous. Instead, we choose to take a sampling of the possible actions and evaluate among those.

A final simplification is that control actions are single-valued. A multiple-value action can either encode control inputs for multiple actuators or encode a sequence of different actions for a single actuator. Although we could encode multiple actions as a vector within a single input, we choose not to do this in order to simplify the problem. Also, it is not clear how the encoding of multiple actions would affect the complexity of the problem since the size of both the search space and the solution space would grow.

```
1. best_node = choose_node(tree);
2. rand_input = generate_input();
3. new_node = simulate(state(best_node),rand_input);
4. new_node.score = evaluate(state(new_node),
                             depth(new_node));
5. if (test_prune(new_node)) {
     delete(new_node);
   } else insert(new_node,tree);
6. best_node.score = best_node.score + v_retry;
7. if (depth(tree) > depth_max) {
     trajectory = add_input(trajectory,tree,new_node);
     tree = child_along_path(tree,new_node);
   }
8. loop;
```

Figure 4.2: Pseudocode for the search controller.

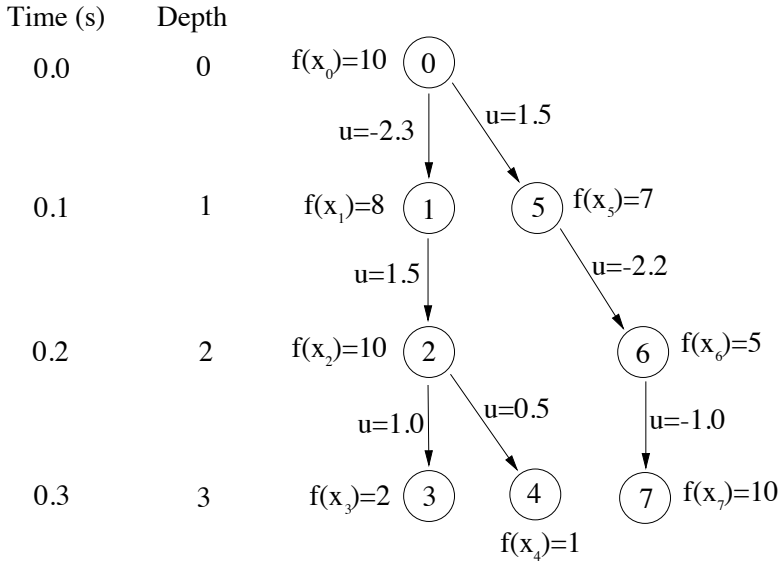

Figure 4.3: An example search tree.

4.3 Algorithm Detail and Example

The algorithm, which controls the development of the search tree, is described with the pseudocode in Figure 4.2. An example search tree is shown in Figure 4.3 for a single decision. The development of the tree is documented in the table of Figure 4.4. At each step, the actual scores for the nodes are given. The highlighted score indicates which node will be extended in the next step. For sorting efficiency, we store the tree nodes in a heap. We will refer to this example as we describe the algorithm in more detail.

In the example tree of Figure 4.3, the current candidate input-histories of three consecutive actions are obtained by tracing the possible directed paths from the root to each leaf.

	Node Scores: v_{eg}							
Step	0	1	2	3	4	5	6	7
1	**10**							
2	6	**9**						
3	6	5	**12**					
4	6	5	**8**	5				
5	**6**	5	4	5	4			
6	2	5	4	5	4	**8**		
7	2	5	4	5	4	4	**7**	
8	2	5	4	5	4	4	3	**13**

Figure 4.4: Detail of progress for the example search tree.

The resulting candidate set of input sequences U_{cand} are given by:

$$U_{cand} = \{(1.5, -2.2, -1.0), (-2.3, 1.5, 1.0), (-2.3, 1.5, 0.5)\}$$

In this tree, node n_7 will be next selected for extension. Using a look-ahead depth $n_{depth} = 4$, this will result in the acceptance of the action represented by $u = 1.5$. For control inputs applied for $0.1s$, this would correspond to $0.4s$ of look-ahead. The selected control input represents the edge from the root node along the path that includes node n_7. The simulation of the root node n_0 with the input $u = 1.5$ results in the configuration represented by node n_5.

After the action is accepted, another tree will be created to determine the next action. Node n_5 will be the root node of this new tree. We are able to reuse the portion of the tree rooted at n_5 since the existing results for this branch remain valid.

4.3.1 Selecting the Node to Extend

An evaluation function is used to guide the exploration of state-space by selecting for system configurations which are more promising. For example, if the goal is to move in the positive z direction, a component of the evaluation function should reasonably include z. The general form of this function is given by:

$$v_{eval} = f(x) + g(n_{depth}) + v_{retry} \cdot n_{children}$$

where x is the state of the system or creature at that node; n_{depth} is the depth of the node in the tree; $n_{children}$ are the number of node children; v_{retry} is the penalty that is added for

each node child; and f and g are user-specified functions. In the example, we use

$$\begin{aligned} f_{eg}(x) &= \{(0,10),(1,8),(2,10),(3,2),(4,1),(5,7),(6,5),(7,10)\} \\ g_{eg}(n_{depth}) &= n_{depth} \\ v_{retry\,eg} &= -4 \end{aligned}$$

to produce the evaluation function

$$v_{eg} = f_{eg}(x) + n_{depth} - 4 \cdot n_{children}$$

Use of the term v_{retry} ensures that the search distributes its efforts over all currently promising motions. In effect, the term causes backtracking to occur if the same node is being continually retried. Thus, even if a search spends a long time in a certain part of a tree, this does not prevent it from going to a completely different branch and subsequently returning if a solution is still not found.

4.3.2 Generating a Control Input

The form of the function that generates a potential control input is left for the user to define. In practice, we have found for several problems that a stochastic sampling from a uniform distribution function is more effective than a fixed discrete choice of samples. With fixed sampling, the algorithm would often get "stuck" backtracking through certain branches of the tree.

In cases where the choice of inputs are finite, it is useful to randomize the order in which the possible inputs are enumerated or to enumerate all the inputs in a single pass (before selecting another node to extend). A branching factor parameter n_{branch} controls the number of simulations computed for each node in a single pass. In our example, we use $n_{branch} = 1$ and the control inputs are randomly selected. Once a control input is generated, a forward dynamic simulation is performed to compute the state representing a new node.

As the search tree grows in depth, the number of nodes, and hence the forward simulations required, grows exponentially. Besides directing the growth of the tree using an evaluation function, it is useful to prune unwanted branches of the tree by specifying constraints. These are described in the next section.

4.3.3 Pruning the Search Tree

For our representation of control, which uses a sequence of control inputs, the following factors affect the size s of the search space: (1) the dimensions of state-space d_s, (2) the dimensions of the control inputs d_u, and (3) the length of the discrete time interval t_i. We can summarize this relation with

$$s = O((t_m/t_i)^{(d_s \cdot d_u)})$$

when searching for a motion sequence of length t_m. For the problems that we worked with, the nodes along the accepted decision path have anywhere from 2-10 branches. This tree can grow quickly for even a short motion sequence. If we can prune even half of the branches at each time interval, the size of the tree would decrease by $(t_m/t_i)^2$. For a sequence of 10 actions, this is a reduction by a factor of 1024! In practice, since the evaluation function already directs the growth of the tree, we will not see such a large acceleration in the search. Nevertheless, examples in Chapter 5 demonstrate that the speed-up can be substantial.

To reduce the number of branches in the tree, a constraint function is used to prune those nodes which represent configurations that are undesirable. For example, if the joint or head of the acrobot should ever touch the ground during a flip, the node representing this motion should be discarded. This particular example (see step 5 in the pseudocode) would require that the simulator provide a way of detecting the collision.

In general, using constraints makes the solution easier to find. This is because pruning prevents the algorithm from wasting time searching through branches that are known to lead to failure. In some cases, however, a poorly defined pruning function can actually divert the search from a good solution by disallowing valid configurations.

Pruning criteria in animation can take several forms. Animated characters typically have constraints on their joint angles. If these are not enforced in the simulation, they should be enforced using constraints. Locomotion problems can constrain the type of allowable motion by preventing certain configurations from occurring, such as landing on the head during a flip. Balancing problems typically include a constraint on the position of the centre of mass relative to the points of support, such as the feet. In general, constraint functions can be defined by the user using intuitive features of the system being modeled. These functions can then be iteratively refined as motion samples are produced and analyzed.

4.4 The Simulation

To support this algorithm, the state of the simulation is constantly being reset to that stored in the current node being extended. For a real-time simulation, this can be a problem since, although the average time to search for each action may be short, the time for any given single decision has the potential for being long if the node is in a particularly ill-conditioned state. For controlling real robots, this is even more problematic since the real state of the robot can quickly deviate from the state in the simulation. In controlling real robots, reusing branches of precomputed simulation may only be practical when the state of the robot is sufficiently close to that in the simulation. Otherwise, the entire tree may need to be regenerated.

We have made the assumption thus far that the tree can only be generated through simulation. Once the search algorithm has produced a series of successful motions, it may be possible to generalize these results and reuse the control that was learned for earlier successful motion. This can be used to accelerate subsequent searches. We explore this machine-learning direction in Chapter 7.

Chapter 5

Low-Level Control Results

The search algorithm of the previous section has been applied to several control problems at different levels of abstraction. We refer to *low-level* control as sequences of basic control inputs that are provided to actuators. On the other hand, *high-level* control refers to sequences of control inputs for abstracted controllers. For example, low-level control for Luxo could be the specification of angles for *poses* used by a *pose-controller* [vKF94]. A pose is a specification of desired angles for each controllable joint. In a pose-controller, PD-control is used to drive the joints to these desired angles for the duration of the pose. A hop can be generated by concatenating a set of appropriate poses. Thus, a hop is a high-level control abstraction that encapsulates a sequence of low-level poses. High-level control for Luxo could be the specification of *hops* used by a hop-controller.

The low-level control problems that we solve for use models for the acrobot (Figure 3.1), the *cart and double-pendulum*[1] (Figure 5.1), and the *spinner* (see Figure 5.2). For the acrobot, we will solve for balance, flipping, and hopping. For the cart and double-pendulum, we solve for various swing-up motions. Finally, for the spinner, we solve for simultaneous inverted balance for the pendulums.

While many of these problems that we choose may not be particularly interesting in animation (although some undoubtedly are), we choose to solve them since they are recognized in the control literature as difficult problems [HM90] [BS92] [BF92] [KOH94]. Control techniques used to approach these problems have largely used analytical models for the computation of solutions. We see these problems as being barely within the grasp of analytical

[1]The cart/pendulum and spinner models were simulated using code produced by SD/Fast from Symbolic Dynamics

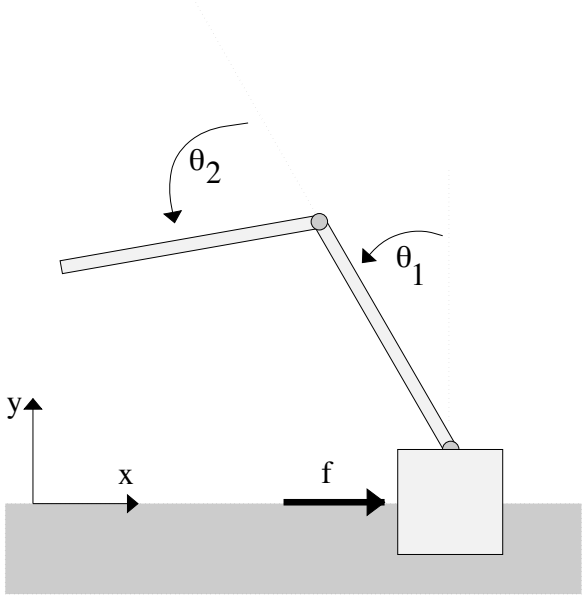

Figure 5.1: The double pendulum.

solutions, but well within the bounds of the algorithmic technique that we propose. The evaluation and pruning functions that we use are surprisingly simple and we obtain motions that are both new and entertaining.

5.1 Acrobot Balance

General parameters for the acrobot simulation are provided in Figure 5.4. The same parameters, unless otherwise specified, are also used for all our acrobot motions.

For acrobot balance, we attach the foot to the ground with an unactuated hinge joint. The goal is then to actuate the joint connecting the links in such a way as to provide a reaction force at the foot which keeps the acrobot balanced and which prevents the head from folding over and touching the ground. Initially, we define the evaluation function using the intuitive notion that the centre of mass should be kept high and centred above the foot. The pruning function tests that the joint and head remain well above the ground. The following functions are used:

$$\begin{aligned} v_{ab} &= n_{depth} + 5 \cdot |cm_x| + |cm_y| - 10 \cdot n_{children} \\ p_{ab} &= (P_{2_y} < 0.5) \vee (P_{3_y} < 1.0) \end{aligned}$$

where cm_x and cm_y are respectively horizontal and vertical offsets of the centre of mass from

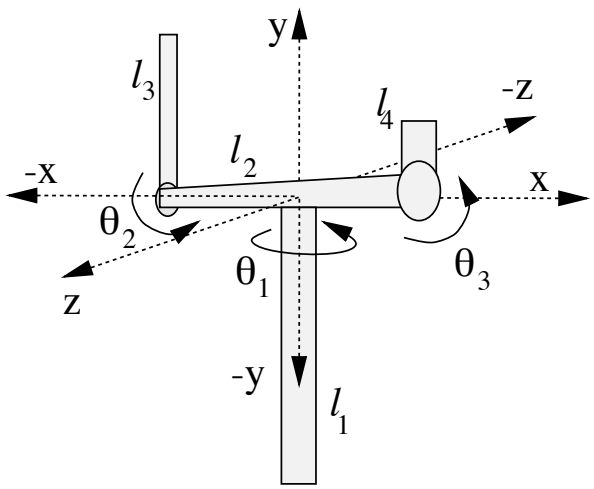

Figure 5.2: The Spinner.

the foot. The points P_1 and P_2 are as defined in Figure 3.1. A PD-controller is attached to the hinge joint which takes as input a desired angle for a fixed time duration $t_{interval}$.

An initial implementation of the search algorithm, conducted with $t_{interval} = 0.1s$, requires approximately 10 000 trials to accept 0.8 s of motion. Since we use 20 stages of look-ahead, this means that an additional 2.0s of motion was generated. However, the latter stages of this motion likely lead to imminent failure and are not included. For generating the control inputs, we sample from a random distribution from the range $-2.6 < \theta < 2.6$, where θ is measured in radians.

Fortunately, we take advantage of these trials to help in the construction of an improved search, using better evaluation and pruning functions. In the resulting successful motion, we observe that θ_1 and θ_2 are approximately linearly related as shown in Figure 5.3, thus defining a near-stable manifold in the state-space. We can use this relationship to refine the evaluation functions and add constraints. In particular, we reformulate them as the following:

$$\begin{aligned} v_{ab} &= n_{depth} - 10 \cdot |3.75 * \theta_1 + \theta_2| - 10 \cdot n_{children} \\ p_{ab} &= (P_{2_y} < 0.5) \vee (P_{3_y} < 1.0) \vee (|3.75 * \theta_1 + \theta_2| > 1.0) \end{aligned}$$

The term $|3.75 * \theta_1 + \theta_2|$ has been derived from the graph of Figure 5.3 and rewards configurations that maintain the linear relationship in the evaluation function. For the pruning function, configurations that stray sufficiently from this linear relationship are eliminated. As a result, 16.4 s of successful input-history is accepted using 10 000 trials.

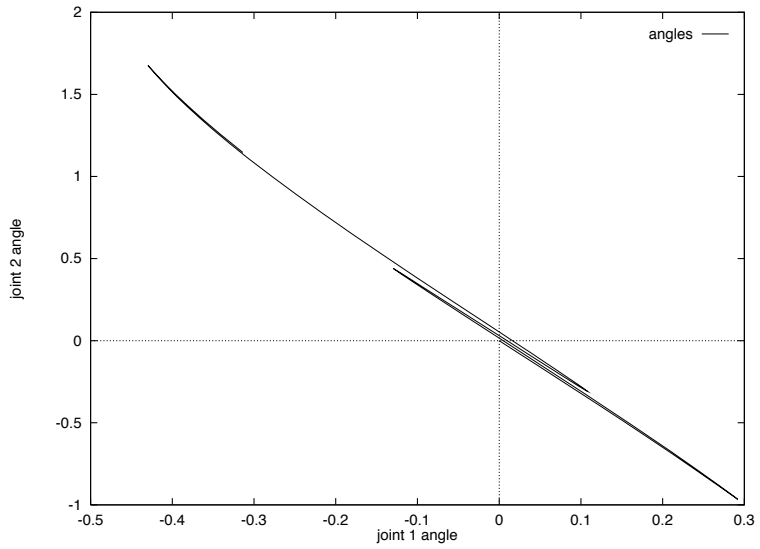

Figure 5.3: Foot and joint angle correlation after 1 second.

m_1, m_2	10 kg
l_1, l_2	1 m
I_1, I_2	0.8333 $kg \cdot s/m^2$
t_{sim}	0.001 s
$t_{interval}$	0.1 s
n_{depth}	20
k_p	200 N/m
k_d	20 $N \cdot s/m$

Figure 5.4: Model parameters.

This corresponds to an average of 1 accepted trial for every 6.1 attempts, which is a clear improvement.

5.2 Acrobot Gymnastics

When the foot hinge constraint is removed, the acrobot can be controlled to produce cartwheels, flips, and hops. These are both more interesting and more difficult motions from the points of view of animation and control. We again define the quantity $cm_x = x_{cm} - x_1$, where x_{cm} is the horizontal position of the centre of mass; x_1 is as shown in Figure 3.1. We also define l as the angular momentum measured in a counter-clockwise direction. The simulation time interval $t_{interval}$ has been increased to 0.18 s to increase the size of the simulation look-ahead window.

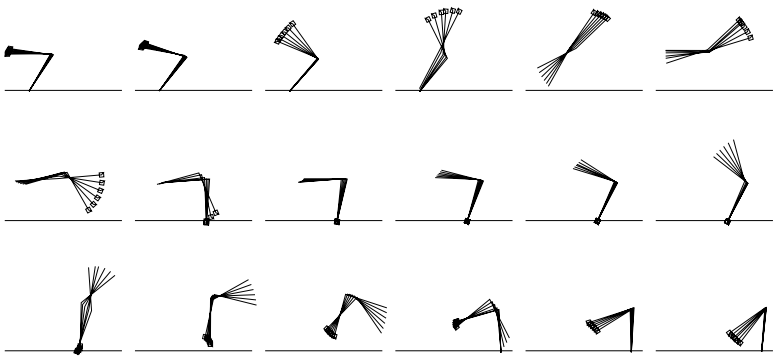

Figure 5.5: A clip from the cartwheel motion.

5.2.1 Acrobot Cartwheel

For a cartwheeling motion, the evaluation and pruning functions are:

$$\begin{aligned} v_{ac} &= -l - 10 \cdot |cm_x| + n_{depth}/10 - 5 \cdot n_{children} \\ p_{ac} &= (y_2 < 0.2) \vee [(y_1 < 0.0) \wedge (y_3 < 0.0)]. \end{aligned}$$

The pruning function constrains the joint to be at least 0.2 m above the ground and ensures that only the end of one link may contact the ground at any instant in time. The evaluation function rewards negative angular momentum l, penalizes centre of mass offsets $|cm_x|$, and rewards simulation progress n_{depth}. The result is a clockwise leg-over-arm motion as shown in Figure 5.5. This cartwheel motion will continue indefinitely. We use $k_p = 50\ N/m$ and $k_d = 5\ N \cdot s/m$ for the joint proportional-derivative (PD) constants.

The joint angles θ_2 and control inputs $\theta_2{}'$ for this motion are shown in Figure 5.6. Note that the control inputs define a step function, and the actual θ_2 is continually pulled toward the desired $\theta_2{}'$ by the PD-controller. From Figure 5.9, it takes on average approximately 90 simulations for every single accepted control action.

5.2.2 Acrobot Flips

For somersault motions, the evaluation and pruning functions are:

$$\begin{aligned} v_{flip} &= -10 \cdot |cm_x| + n_{depth} - 5 \cdot n_{children} \\ v_{front\ flip} &= -10 \cdot |cm_x| + n_{depth} - 5 \cdot n_{children} \\ p_{flip} &= (y_2 < 0.2) \vee (y_3 < 0.4) \\ p_{front\ flip} &= (y_2 < 0.2) \vee (y_3 < 0.4) \vee [(y_1 > 0.5) \wedge (l > 0.0)]. \end{aligned}$$

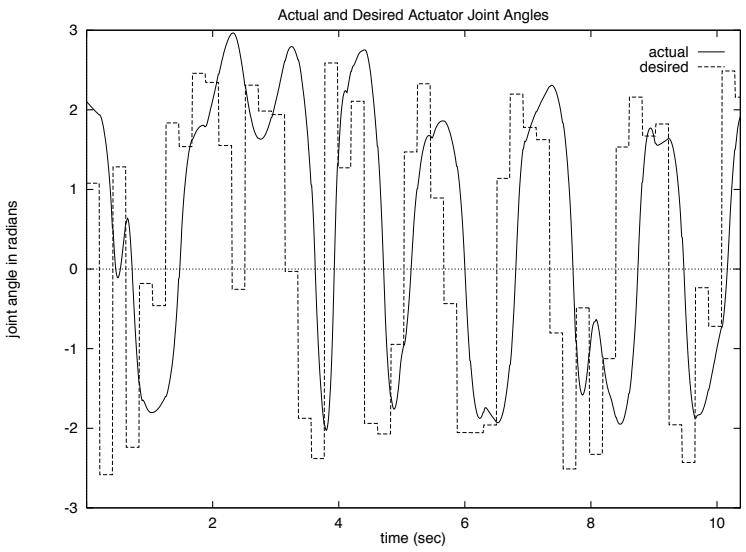

Figure 5.6: Control inputs to the joint angle.

The pruning function constrains the joint and head to avoid contact with the ground. The evaluation function penalizes centre of mass offsets and rewards simulation progress. Given these constraints, the flips naturally occur in the motions produced by the search algorithm. The flipping motions can be sustained indefinitely. Successful landing is attributed to the penalty for a large centre of mass offset. This encourages actions that drive the foot to be underneath the centre of mass. Since there is no specification of the direction of the flip, both front and back flips are possible by default.

The frames in Figure 5.7 demonstrate a back-flip followed immediately by a front-flip. To achieve the torque required for the flip, PD constants of $k_p = 200\ N/m$ and $k_d = 20\ N \cdot s/m$ are used. Higher-order flips (double, triple, etc.) have been achieved by simply increasing the k_p of the actuator. From Figure 5.9, it takes on average approximately 40 simulations for every single accepted control action.

A motion with only front flips uses the same pruning and evaluation functions as the normal flip with an additional constraint that whenever the foot is above 0.5 m, the angular momentum l must be positive.

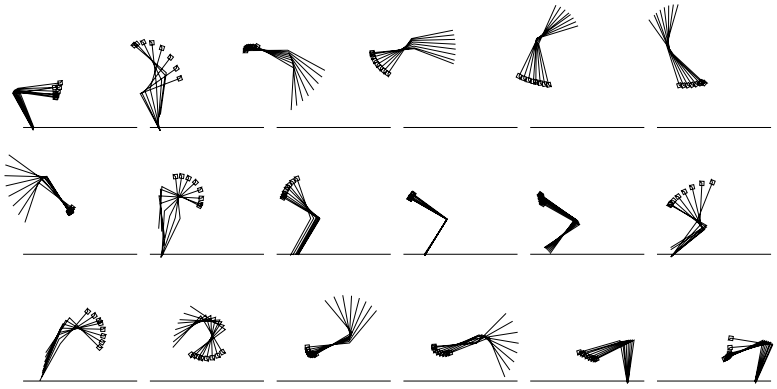

Figure 5.7: A sequence of flips.

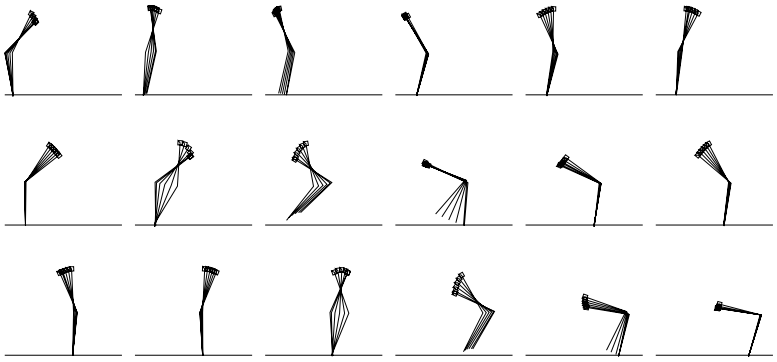

Figure 5.8: A hopping sequence.

5.2.3 Acrobot Hop

The hop requires the largest number of pruning constraints. The evaluation and pruning functions are:

$$\begin{aligned} v_{hop} &= n_{depth} - 5 \cdot n_{children} \\ p_{hop} &= (y_2 < 0.2) \vee (y_3 < 0.4) \vee [(y_1 > 0.0) \wedge (\dot{x}_1 < -0.1)] \vee (y_1 > 0.4) \end{aligned}$$

The pruning function keeps the joint and head above the ground, permits only small backward velocities when the foot is in the air, and keeps the foot below 0.4 m in the air. The evaluation function rewards temporal progress. Figure 5.8 demonstrates the resulting motion. To the best of our knowledge, this motion can be indefinitely sustained. The PD-constants used here are the same as those for the flips.

Figure 5.9 shows a history of how simulation trials are allocated to different levels of the tree (times of the motion) as the search algorithm proceeds. A negative slope in these graphs indicates a back-tracking behaviour in the search. The general rising slope of the

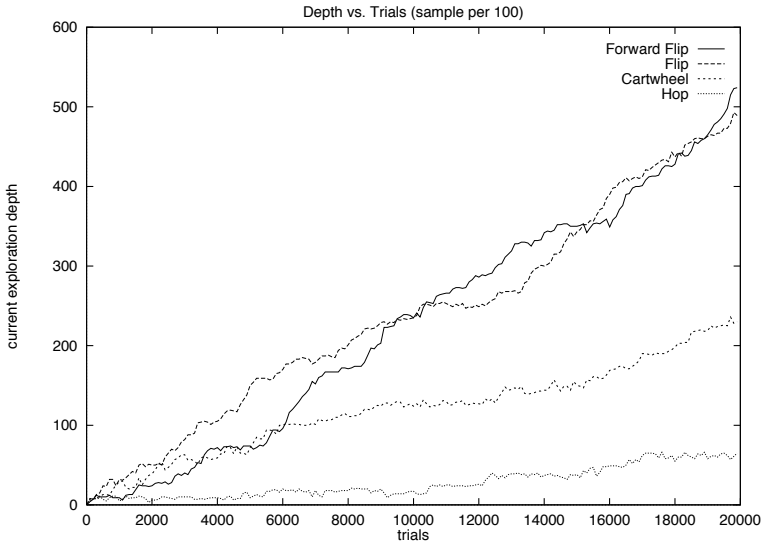

Figure 5.9: Progress rate for different motions.

curve indicates how easy a given type of motion is to plan. A steep slope indicates a relatively easy motion to plan, while a shallow slope indicates that progress is difficult. Surprisingly, flip motions are the easiest to generate, followed by the cartwheel in difficulty, and then the hop.

5.3 Double Pendulum

The double pendulum (see Figure 5.1) model is controlled using a sequence of fixed horizontal forces applied to the cart for a fixed time interval $t_{interval} = 0.1\ s$. Both of the joint hinges are unactuated. The algorithm produces control for two types of swing-up and for a continuous giant swing, while maintaining the cart within the range $|x| < 5$.

The difficulty in performing a swing-up is that energy must be built up appropriately through swinging back and forth [KOH94]. The forces of the cart must be directed to produce increases in the total potential and kinetic energy of the system. This problem can be compared to that of a person on a swing, where the total energy of the system must be raised incrementally with each swing through changing the length of the swing.

For the first swing-up problem, we restrict inputs to be taken from discrete range $u = \{-f, 0, f\}$, with $f = 40\ N$. This is known as *bang, bang, zero-input* control. The evaluation function rewards the configuration for potential and kinetic energies for the system (E_{system}) that approach the energy required for an inverted static position (E_{static}, which is unstable). The function also rewards the configuration that has a small θ_2 and $\theta_2{}'$, which corresponds

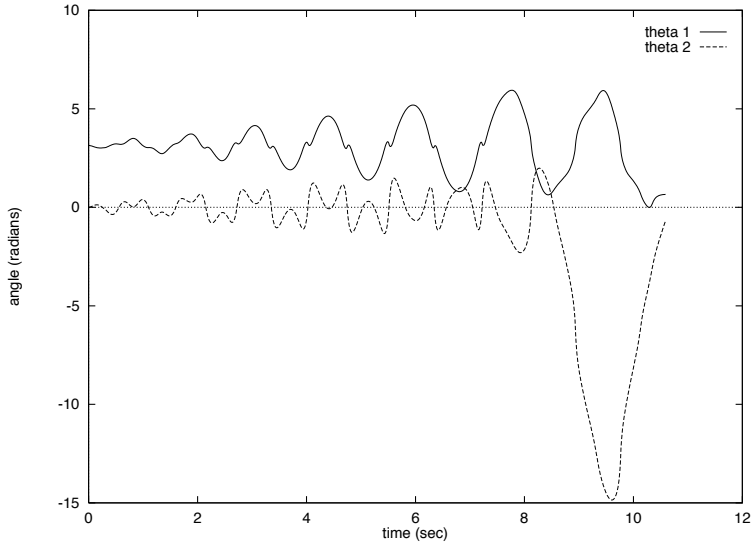

Figure 5.10: Bang, bang, zero-input swing-up.

to a fully extended double pendulum. The state-space trajectory for a successful swing-up is shown in Figure 5.10. This motion results from the following evaluation and pruning functions:

$$\begin{aligned} v_{su} &= -|E_{system} - E_{static}| - |\theta_2| - |\theta_2'| + y_{cm} - 10\,000 \times (n_{children} > 3) \\ p_{su} &= |x| > 5 \end{aligned}$$

where y_{cm} is the centre of mass of the two links.

For the second swing-up, we drop the restriction on taking discrete input samples and allow the control to take values from the continuous range $u = [-f, f]$. As expected, removing the constraint on the inputs results in a much better swing-up. Figure 5.11 shows the angles of the two pendulums in a particularly good example of the final swing-up motion. Note that not only do the angles approach zero radians, but their derivatives (angular speed) also drops.

Finally, the giant swing uses an evaluation function that ignores the energy term but rewards depth. The configuration continues to be rewarded for having a small θ_2 and θ_2'. The result is a cyclic swing as shown in Figure 5.12. Note that this swing was initiated from an inverted balance position to provide the energy required for the first swing. The angle θ_1 is wrapped around for clarity. The evaluation and pruning functions are modified to be as follows:

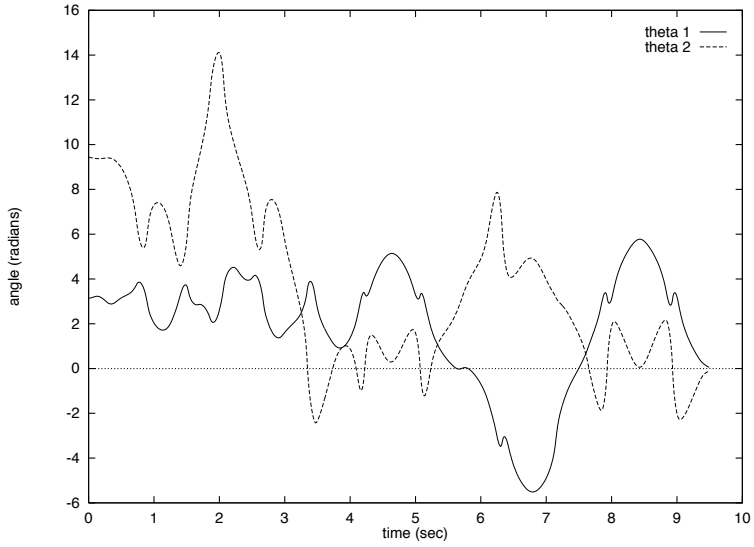

Figure 5.11: Continuous input swing-up.

$$\begin{aligned} v_{giant} &= -|\theta_2| - |\theta_2{}'| + y_{cm} + n_{depth} - 5 \times n_{children} \\ p_{giant} &= |x| > 5 \end{aligned}$$

5.4 Spinner

The final example of low-level results that we demonstrate uses the *spinner* model (see Figure 5.2) to solve the balance problem for a double inverted pendulum. In this problem, the only input is torsional control to the centre hinge rotating about the y-axis as measured by θ_1. The other two hinges, one for each pendulum, are unactuated. Beginning with both pendulums in an inverted position, the problem is to control the system to prevent either pendulum from falling over completely. The evaluation and pruning functions maintain an empirically-determined ratio of θ_2 to θ_3 that was determined by plotting the state vector for an initial sequence of successful control.

For an initial attempt at solving the problem, we use the following evaluation and pruning functions:

$$\begin{aligned} v_{sp1} &= n_{depth} - n_{children} \\ p_{sp1} &= (|\theta_2 > 0.3|) \vee (|\theta_3| > 1.2) \end{aligned}$$

Using these choices, only 21 trials have been accepted after 100 000 attempts. Examination of the state variables for the accepted trials yields an inverse correlation between θ_2 and θ_3 as shown in Figure 5.13.

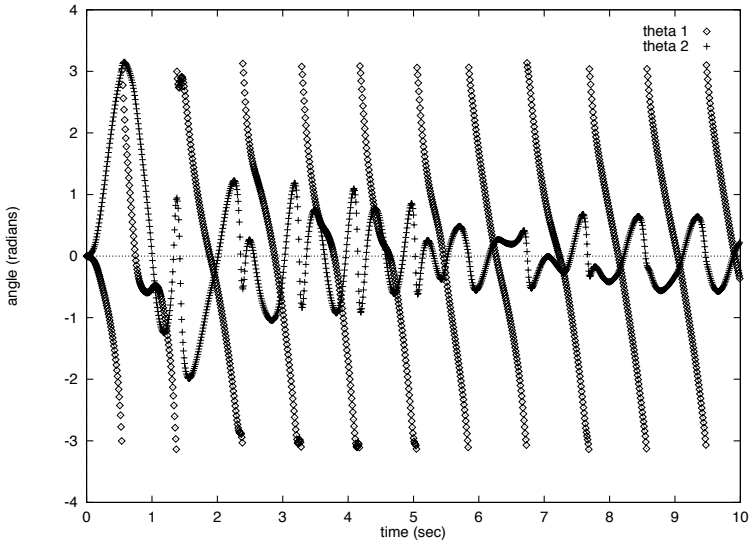

Figure 5.12: Giant swing.

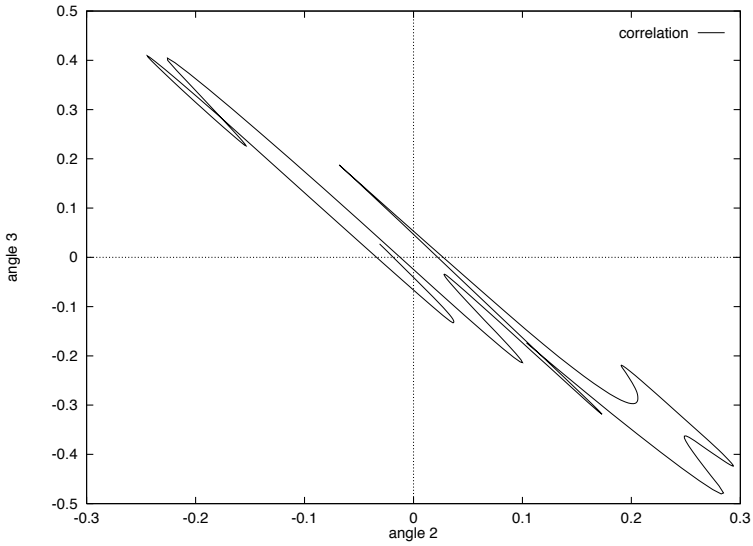

Figure 5.13: Correlation of joint angles for the spinner arms.

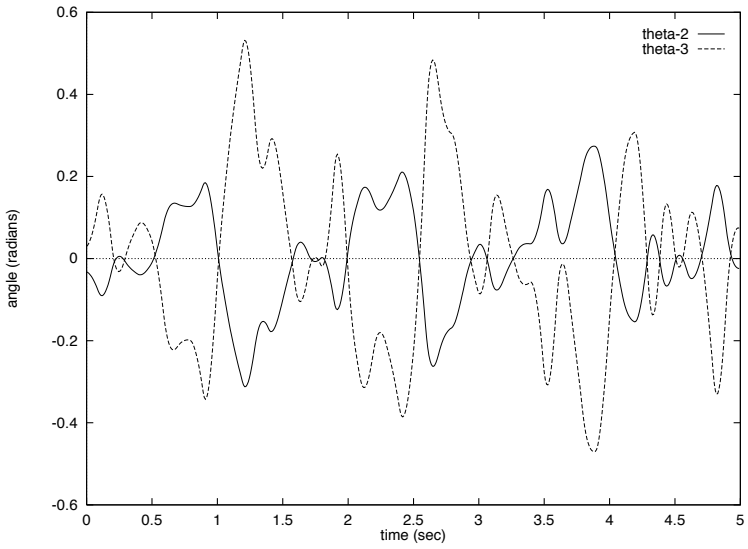

Figure 5.14: Spinner inverted double pendulum balance.

The graph shown in Figure 5.14 demonstrates the correlation of θ_2 and θ_3 for the successful balance. We can now modify the evaluation and pruning functions to be the following:

$$\begin{aligned} v_{sp1} &= n_{depth} - 5 \cdot n_{children} + |2.0 \cdot \theta_2 + \theta_3| \\ p_{sp1} &= (|\theta_2 > 0.3|) \vee (|\theta_3| > 1.2) \vee (|2.0 \cdot \theta_2 + \theta_3| > 0.1) \end{aligned}$$

The result is a drastic improvement with over 400 trials being accepted after 100 000 attempts. A sample of this motion is shown in Figure 5.14. We will see a further improvement by applying memory-based techniques, which will be discussed later.

5.5 Conclusions

Our search algorithm has been applied successfully to several difficult low-level control problems. Balance, hop, and flip motions were demonstrated for the acrobot; different swing-ups and a giant swing were demonstrated for the double pendulum and cart; and a balance sequence was demonstrated for the spinner. The motions were produced by encoding an evaluation function that captured desired intuitive features such as balance. A pruning function encoded constraints to eliminate configurations that led to failure. In some cases, these functions were refined by examining the relationships among the state variables for successful motions.

The motions described in this chapter were all controlled at a low-level—at the basic

level of torques and forces. In the next chapter, this algorithm is applied for high-level motion planning using the problem of Luxo as it plans a sequence of hops to traverse various terrains.

Chapter 6

High-Level Control Results

In the previous chapter, we used our search algorithm to synthesize low-level control of systems that were actuated using a sequence of single inputs. However, such a specification does not scale well for more complex creatures. Animation is often concerned with creatures having many actuators and with motions whose primitives are more complex. For example, even a simplified human model would have a large number of actuators. In this case, an animator would prefer to specify the motion of a human in terms of different types of steps rather than actuations at the different joints.

We now begin to tackle the problem of generalizing our search algorithm to solve high-level control problems. We first make the assumption that low-level controllers exist that can perform several variations of a motion. These controllers can arise from a set of discrete controllers or from one that is parameterized. In particular, we could *clip* or take samples of the motions that are learned from the low-level search and use them as control primitives. The search technique can then be used to concatenate these more complex primitives into appropriate sequences. We have performed some preliminary investigations with some promising results that we now present.

6.1 Path-Planning for Luxo

In this particular example, we shall deal with the motion of *Luxo*, a 3-link articulated figure with two actuators shown in Figure 6.1. The ground contact model for Luxo uses the same penalty method as used by the acrobot: A spring-and-damper system applies a force to pull the base of Luxo toward the initial point of ground contact, as shown in Figure 6.2.

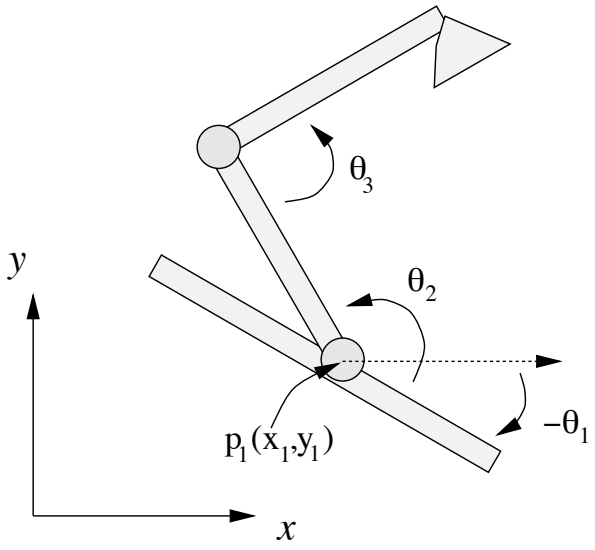

Figure 6.1: Luxo Lamp.

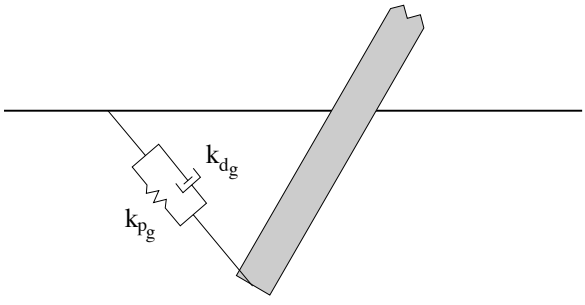

Figure 6.2: Simulation of ground contact.

The motion primitives to be used in this case are the control histories necessary to execute complete jumps. A total of 5 different types of jumps are used in creating the motions shown in Figure 6.3. Each jump consists of a sequential series of three states, where each state is of a fixed duration (0.13 seconds) and holds the control inputs constant. The control inputs are desired joint angles, which are fed to PD-controllers that are used to generate actuator torques. This is also known as pose-control.

Executing a given jump primitive will not always produce the same motion. This is because the resulting motion is also a function of the initial state. The search algorithm must thus determine a suitable order in which to apply the jump primitives. As with the other examples, this is done by planning ahead a fixed number of stages into the future where, in this case, the stages correspond to complete jumps. For the examples shown in Figure 6.3, it is sufficient to plan ahead a maximum of 4 jumps. As with previous examples, the search is conducted using a best-first strategy. Since the control primitives are selected from a discrete set, the random generation of control inputs is not necessary. Instead, we choose to evaluate all choices at each stage and prune failed motions as necessary.

The evaluation function used to compare nodes is the total distance travelled. The mechanism which is used to constrain the growth of the search tree is the pruning of branches that lead to *falls*. Any contact with the ground by a part of Luxo other than the base is considered a fall. In the given examples, the falls are sufficient to prune 50 to 66% of the search tree. More of the tree could be pruned by establishing a conservative upper bound on the distance that can be travelled in a jump. Branches which could not possibly obtain the current maximum distance, even with future maximum-size jumps, can then be pruned. This then becomes a type of branch-and-bound algorithm.

The motions shown in Figure 6.3 took between 4.5 and 9.5 minutes to plan on a 90 MHz Pentium PC, where the simulation itself requires 0.41 real seconds for every simulated second. The final motions clearly show anticipation of the upcoming terrain.

6.2 Animator Interface

The interface to the system consists of two steps: (1) specification of the terrain and (2) initiation of the search. For simplicity, we specify the terrain using a sequence of line segments. These are read by the simulator, which handles the ground contact model using the same

spring-damper system as used for the acrobot. The animator then initiates the search. The search completes either when a successful path has been planned, or when the terrain is found not to be navigable using the supplied primitives and search parameters.

6.3 Expanding the Search

In cases where the search algorithm cannot find a solution path, several remedies can be used. However, these remedies require additional search time. The first possibility is to introduce additional types of jump primitives. The second is to keep the existing jump primitives, but to allow some type of interpolation between them so that the control space becomes continuous. The last possibility is to force the search algorithm to use a longer-duration planning window, thereby doing a better job of avoiding dead-end situations.

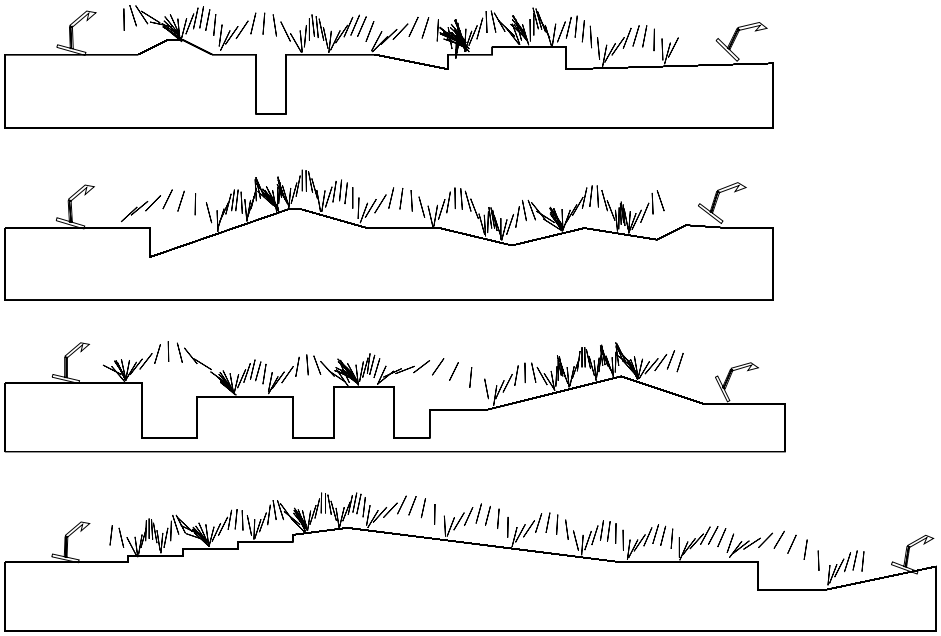

Figure 6.3: Luxo goes cross-country running

Chapter 7

Memory-Based Acceleration

A gymnast learns through experience by reinforcing the correct actions leading to successful manoeuvres. In the same way, the search algorithm can access stored experience in order to guide the computation of control inputs that will lead to a successful motion. If, for example, we have stored the data for balancing an acrobot for 100 seconds, surely this information could be used in continuing to balance. Ideally, this experience may lead to closed-loop control through the interpolation of a correct input for any given state. Alternately, it may be possible to generate and search through a look-ahead tree that uses experience to perform a type of "mental" simulation of possible outcomes. Compared to the cost of running trial simulations, this would result in a significant reduction in required computational resources.

This chapter provides a brief initial investigation into this experience-based acceleration. We first describe the representation of the stored experience. Following the definition of this representation, methods for accessing the information are addressed as well as relevant issues in defining how to measure the similarity between a pair of states. The chapter concludes by describing some promising experimental results.

7.1 Stored Experience

The results of the algorithm are a feasible control input history h_U that can be applied through simulation to yield a state-space trajectory h_X in the environment E. At the discrete decision points, we have successful samples of the mapping $X \times E \mapsto U$, which can be stored as *experience*. For an accurate physical simulation, it is not useful to simply store the mapping $x_t \mapsto x_{t+\Delta t}$ for two reasons: (1) For most problems, state-space is continuous

and so matching a state exactly is unlikely; (2) We require knowledge of the control that was successful so that it can be applied again.

In practice, we could generalize this experience by only storing selected features of the environment relative to the controlled system. For example, if Luxo is planning to jump across a chasm after it has successful climbed a hill, it no longer matters to the planner that the hill is in the environment. On the other hand, it matters greatly to Luxo that it may plunge into the oncoming chasm given the wrong decision. The planner should only focus on the features that are relevant to the current path being planned. In this case, the mapping would consist of the relevant components E' of the environment producing the mapping $X \times E' \mapsto U$. The relevant environment component E' could be considered as a set of sensory information. Thus, the experience could be interpreted to answer the question: "given the configuration of Luxo and the locations of immediate obstacles, what actions were performed in the past that allowed Luxo to continue hopping?"

7.2 Nearest-Neighbours

We use a nearest-neighbour approach as a lookup into this stored experience set in an attempt to interpolate control for an arbitrary state vector x. The success of this approach for arbitrary x depends on (1) the density of the sample points at x, (2) the existence of a solution at x, (3) the sensitivity of the system to small changes in the control input at x, and (4) the choice of interpolation function.

The density of sampling will increase as more experience is stored, so the system can be expected to perform *on-line learning*. If a solution does not exist at x, the search algorithm can simply direct the search elsewhere. If the system is highly sensitive to small changes in the control inputs at x, the lookup will yield results which are not likely to be any worse than stochastically generating an input. As with (1), increasing the stored samples will reduce this difficulty. Finally, different interpolation functions can yield very different search acceleration (or even deceleration!). The specific choice depends on the problem being solved and the locality of state-space being searched in the problem. We have found in practice that application of the *mean* of the control inputs of a set of nearest-neighbours has yielded good results.

7.2.1 Similarity Metric

To compute the nearest neighbours of a given state, we require the definition of a distance function that can be applied to state pairs. Consider the choice of the Euclidean distance metric whose simplest form is $d = \sqrt{\sum_i (x_{1,i} - x_{2,i})^2}$. Several issues arise when considering a distance metric:

- the state dimensions have different units,
- the state dimensions should not necessarily be equally weighted, and
- the state dimensions may be correlated in some way.

For example, the graph of Figure 7.3 demonstrates the differences in the rate of progress in searching for balance for the spinner using all dimensions equally weighted (6D projection) and using only the angles of joints 2 and 3, along with their derivatives (4D projection). In this case, the joint 1 angle and derivative are irrelevant, and so progress is better made by eliminating those elements from the distance metric.

Cross-validation is a technique used to overcome some of the problems that are mentioned above regarding differences in the significance of the dimensions. This technique takes a set of sample points with an assignment of weight to each dimension. For each sample point s, a local model without s is computed to estimate the value of the removed s. The sum of the discrepancies computed for all the points is known as the *cross-validation error*. A local minimum for this error can be found by using gradient descent techniques.

The resulting weighted distance function has the form $d = \sqrt{\sum_i w_i (x_{1,i} - x_{2,i})^2}$, where w_i refers to the *weights* of each dimension i. Atkeson and Lowe have proposed approaches for automating the selection of weights for continuous [Atk91] and binary [Low93] variables, respectively. They both use cross-validation.

Lowe uses conjugate gradient descent to select an optimal set of weights to minimize the cross-validation error. We implemented this approach, but find that, in practice, the cost of computing the optimal weights far outweighs the cost of additional simulation trials for the set of problems that we are investigating. We find it sufficient to simply hand-pick relevant dimensions and to ignore their units. In some cases, we perform some range normalization. For example, in the case of the acrobot, we normalize the angular velocities of the joints since they are an order of magnitude larger than the angular measurements.

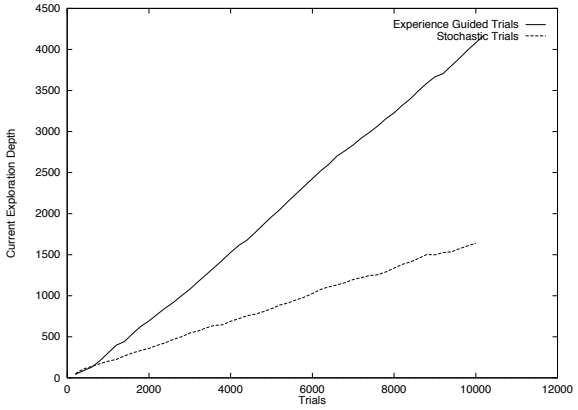

Figure 7.1: Acrobot balance accelerated using experience.

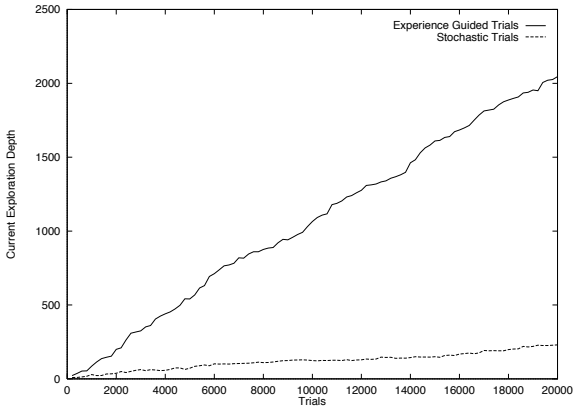

Figure 7.2: Acrobot cartwheel accelerated using experience.

We expect that optimizing the weights would be helpful, if the computation were less costly. Unfortunately, a single global set of weights would be inappropriate since the relevance of different state-space dimensions changes locally. As an example, consider the relatively high importance of the foot position of the acrobot during ground contact versus during flight.

7.3 Results

We have used a memory-based approach to successfully accelerate the search for input-histories for acrobot balance, acrobot cartwheel, and spinner balance. Various acceleration factors were experienced as shown in the plots of Figures 7.1-7.3. In these cases, relatively few (several hundred) state-input experience pairs were used.

In general, acceleration performs better where more experience points are available. In

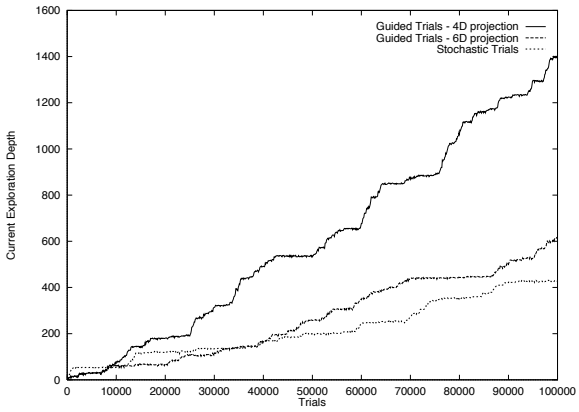

Figure 7.3: Spinner balance accelerated using experience and different similarity metrics.

fact, when there are few stored state-input pairs, the guided search can progress much more slowly than the stochastic search. This result makes sense, considering that the nearest neighbours are not likely near enough in state-space to yield an input that would be better than a stochastic input. In fact, the input can be worse than a stochastic input depending on the choice of interpolation function. For example, choosing an unweighted *mean* can yield a consistently poor value when few neighbours are available. For our trials, the acceleration was noticeable with even fewer than 50 sample points of stored experience.

7.4 Conclusions

We have proposed a method for storing and reusing experience in the form of state-action pairs. This experience is accessed using a Euclidean distance metric, with weights on each dimension. While we have found it sufficient to hand-pick these weights for efficiency reasons, the dimensional weights can be optimized globally or locally using cross-validation methods, which have been the subject of several studies in the field of Machine Learning. Several experiments were performed which demonstrate conclusively that even a sparse sampling of successful state-action pairs can drastically accelerate the search. These early, promising results suggest that further investigations into search acceleration should be worthwhile. Ultimately, the goal would be to produce closed-loop or nearly closed-loop control so that building the look-ahead search tree may no longer need to incur the computational costs of performing actual simulations.

Chapter 8

Conclusion

8.1 Contributions

A new type of control algorithm has been proposed for dynamic motion discovery, control, and planning for various actuated, articulated figures. This algorithm determines appropriate actions by exploring different possible results using simulation. It has been shown that various motions can be produced by using this decision-tree technique and by applying simple pruning and evaluation functions. The control for these motions can be stored as experience and accessed to accelerate future searches using a nearest-neighbour approach.

The approach we use in this thesis is to treat the problem of control synthesis as a path-planning problem. In particular, the algorithm searches for a physically-feasible trajectory through state-space. The path is physically-feasible in the sense that the simulator is capable of producing the motion using a sequence of control actuations applied to the model. The use of simulation guarantees dynamic correctness for the motions that are found.

8.1.1 Applications of the Algorithm

This algorithm is useful for motion synthesis and planning problems in both animation and robotics. For animation, the advantage of this approach over the traditional method of keyframing is that the produced motions are constrained by the simulator to be physically correct. However, because the specification of the motion is behavioural, exact space and time constraints cannot be placed on the models. Also, the algorithm is computationally expensive and many trial simulations are eventually discarded because they do not contribute to the final solution. However, this expense must be seen within the perspective

that the algorithm addresses a class of problems that are difficult or impractical to solve through other methods.

For robotics, the look-ahead simulations would be particularly useful for dealing with arbitrary environments, provided the environments can be modeled appropriately. Two major disadvantages are that (1) the simulation may not be accurate enough to produce a correct decision and (2) the computational requirements of the algorithm may prohibit its use for real-time operation using current technology. For systems that require dynamic control for balance, small inaccuracies in the simulation will be amplified over time. Branches of the search tree may not be reusable as the results become more inaccurate and consequently much of the tree may need to be regenerated. Some suggestions for future work that address ways of accelerating the search process are found at the end of this chapter.

Systems that are particularly suited for this algorithm are those that are actuated through a sequence of single-inputs and whose motions rely on planning. Several examples of control for underactuated mechanical systems are demonstrated; these systems rely on building appropriate momentum in the early stages of control in order to successfully complete a manoeuvre. Balance, locomotion, and path-planning problems are particularly well-suited.

The algorithm is not designed to search for low-level control for systems that have multiple actuators. However, by using high-level control primitives, the search algorithm can be scaled to handle such problems.

8.1.2 Results

We have produced new gymnastic motions for the difficult-to-control acrobot. Our results indicate that the automatic synthesis of control for finely-tuned gymnastics behaviours might indeed be achievable. Examples of balance, cartwheel, flip, and hop motions are demonstrated. To our knowledge, this is the first successful hop control strategy for the acrobot that does not require over-rotation of one of the links and is the first successful control strategy for any kind of flipping. We have demonstrated the generality of this algorithm in applying it successfully to various other control problems. Using more complex control primitives, this algorithm can successfully plan locomotion paths to navigate difficult terrains for Luxo the hopping lamp.

8.2 Visual versus Physical Realism

Up to this point, we have strictly adhered to physics-based solutions for the problems that we proposed. We should keep in mind that, for the purpose of animation, only visual realism is required. Minor violations of physical laws are not likely to be noticeable by the human eye, while their incorporation can greatly simplify a problem. In many cases, we can completely close the loop for the successful use of experience by "cheating" on the physical aspects of the motion and using the stored experience as motion clips whose end points can be smoothly interpolated.

For example, the reason Luxo may fail to jump across a chasm when success had been predicted is that the pre-jump state of Luxo may not have had a close enough match to that of the retrieved state-input pairing that yielded the successful prediction. Rather than abandoning this action, motion interpolation could be used to smoothly connect two already-calculated motion segments. For end points that are already very similar in state, this "cheating" would not likely be perceptible. This would, of course, be suicidal for a real robot!

8.3 Future Work

In this thesis, performed tests have been limited to figures having a single control input. We would like to see how multiple inputs affect the algorithm since the growth of the search space is exponential in the number of control variables. However, the space of desirable configurations can also grow, so it is unclear how the complexity of the search would increase. The growth in complexity will likely depend on the individual characteristics of each problem.

It would be useful to automatically categorize low-level control solutions so that the learned control can be reused for high-level control. For example, the application of the search algorithm to the problem of acrobot flipping produced many variations of the basic flip. The control that was used for the different flips could be used to create a set of flip primitives that could be used when planning at a high level.

Issues affecting the reuse of stored experience can be further explored. One issue is finding a distance metric optimization technique that can be efficiently computed. This would be particularly useful for discarding dimensions of state-space that are irrelevant to

control and that can in fact confound the interpolation of control. An example of this was demonstrated earlier in Chapter 7 with the spinner model. Removal of irrelevant dimensions would be particularly useful as sensory data is made a part of the stored experience. Sensory information is often redundant and a distance metric optimization technique should detect this. A second issue is appropriate nearest-neighbour selection. For example, we chose simply to select the nearest neighbours; it may be more useful to select neighbours whose bounding box encloses the sample point. A third issue is control-input extrapolation techniques. We have used simple statistical measures, such as the mean. While this has worked well for the problems we proposed, other measures should be explored.

With enough samples of successful motion control, it may be possible to estimate the results of applying control inputs without performing a simulation. It may then be possible to create a search tree that is based entirely on estimated data as opposed to simulated data. This could accelerate the search tremendously and make it useful for applications that rely on real-time operation.

Finally, the growth of the search tree could be better managed, which would increase the speed of the search. One way to do this is to define an evaluation metric that can be applied across entire branches of the tree. In this way, the criteria for selecting nodes for extension can be based on global characteristics of the tree rather than simply those of the local node. Another way to manage the growth of the tree is to penalize nodes for having children or grand-children that are pruned. Thus, nodes that always produce failing children could also be deemed useless and pruned.

Bibliography

[Atk91] Christopher Atkeson. Using locally weighted regression for robot learning. *Proceedings of the 1991 IEEE Internation Conference on Robotics and Automation*, pages 958–963, April 1991.

[BF92] Matthew D. Berkemeier and Ronald S. Fearing. Control of a two-link robot to achieve sliding and hopping gaits. *IEEE Conference on Robotics and Automation*, 1:286–291, 1992.

[BF94] Matthew D. Berkemeier and Ronald S. Fearing. Control experiments on an underactuated robot with applications to legged locomotion. *Proceedings, IEEE International Conference on Robotics and Automation*, pages 149–154, 1994.

[Bor92] S. A. Bortoff. *Pseudolinearization Using Spline Functions with Application to the Acrobot.* PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[BS92] S. A. Bortoff and M. W. Spong. Pseudolinearization of the acrobot using spline functions. *Proceedings, 31st Conference on Decision and Control*, pages 593–598, 1992.

[BSA83] Andrew Barto, Richard Sutton, and Charles Anderson. Neuronlike adaptive elements that can solve difficult learning problems. *Systems, Man, and Cybernetics*, SMC-13(5):834–846, September/October 1983.

[BW95] Armin Bruderlin and Lance Williams. Motion signal processing. *Computer Graphics Proceedings*, pages 97–104, August 1995.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[GT95] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. *Computer Graphics Proceedings*, pages 63–70, April 1995.

[HM90] J. Hauser and R. M. Murray. Nonlinear controllers for non-integrable systems: The acrobot example. *Proceedings, American Control Conference*, pages 669–671, 1990.

[HR90] Jessica K. Hodgins and M. H. Raibert. Biped gymnastics. *International Journal of Robotics Research*, 2:115–132, 1990.

[HSL92] Jessica K. Hodgins, Paula K. Sweeney, and David G. Lawrence. Generating natural-looking motion for computer animation. *Graphics Interface*, pages 265–272, 1992.

[HWBO95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. *Computer Graphics*, 1995.

[KOH94] Shigeyasu Kawaji, Ken'ichi Ogasawara, and Hidenobu Honda. Swing-up control of double pendulum using genetic algorithms. *3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, August 1994.

[Las95] John Lasseter. *Toy Story*. Walt Disney Pictures, 1995.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1991.

[Low93] David G. Lowe. Similarity metric learning for a variable-kernel classifier. Technical Report TR-93.43, University of British Columbia, November 1993.

[NM93] J. Thomas Ngo and Joe Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993.

[PAH92] M. G. Pandy, F. C. Anderson, and D. G. Hull. A parameter optimization approach for the optimal control of large-scale musculoskeletal systems. *Transactions of the ASME*, 114:450–459, November 1992.

[Rai85] Marc Raibert. *Legged Robots that Balance*. MIT Press, Cambridge, 1985.

[RFV94] Charles Russell, Michael Fallon, and Mark Verheiden. *The Mask*. New Line Cinema, 1994.

[RH91] Marc Raibert and Jessica Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[SC92] A. James Stewart and James F. Cremer. Beyond keyframing: An algorithmic approach to animation. *Graphics Interface*, pages 273–281, 1992.

[SCK93] Steven Spielberg, Michael Crichton, and David Koepp. *Jurassic Park*. Amblin Entertainment, 1993.

[Sim94] Karl Sims. Evolving virtual creatures. *Computer Graphics Proceedings*, pages 15–22, July 1994.

[Sut91] Richard S. Sutton. Reinforcement learning architectures for animats. *From Animals to Animats*, pages 288–296, 1991.

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. *Computer Graphics Proceedings*, pages 43–50, july 1994.

[TTG94] Demetri Terzopoulos, Xiaoyuan Tu, and Radek Grzeszczuk. Artificial fishes with autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Proceedings of the Artificial Life Workshop*, July 1994.

[vF93] Michiel van de Panne and Eugene Fiume. Sensor-actuated networks. *Computer Graphics Proceedings, Annual Conference Series*, pages 335–342, 1993.

[vFV90] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controllers. *Computer Graphics*, 24(4):225–234, August 1990.

[vFV93] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Physically-based modeling and control of turning. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 55(6):507–521, November 1993.

[vKF94] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Virtual wind-up toys for animation. *Graphics Interface*, pages 208–215, 1994.

[Web93] Merriam Webster, editor. *Merriam Webster's Collegiate Dictionary.* Thomas Allen & Son, tenth edition, 1993.

[WH94] Wayne L. Wooten and Jessica K. Hodgins. Simulation of human diving. Technical Report GIT-GVU-94-31, Georgia Institute of Technology, July 1994.

[Win84] Patrick H. Winston. *Artificial Intelligence.* Addison-Wesley, 2 edition, 1984.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

[WP95] Andrew Witkin and Zoran Popović. Motion warping. *Computer Graphics Proceedings*, pages 105–108, August 1995.

[WS89] Jane Wilhelms and Robert Skinner. An interactive approach to behavioral control. *Graphics Interface*, pages 1–8, 1989.