Solving the Hamilton-Jacobi-Bellman Equation for Animation

Gideon Amos

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University of London.

Centre for Advanced Instrumentation Systems

University College London

9th July 2002

Abstract

This thesis addresses the construction of a practical method for solving the Hamilton-Jacobi-Bellman (HJB) equation for the purpose of computer animation. Solutions of the HJB equation yield optimal feedback controllers which may be applied to dynamic models of simple robots or animals to generate motion sequences. Our implementation has proven effective at producing controllers for models with a phase space of up to four dimensions. An efficient swing-up and balance controller was obtained for the acrobot, a double pendulum with a single actuator at its middle joint.

A number of specific techniques are proposed to improve both the efficiency of the method and the confidence we may have in the solutions it produces. These include improvements to the convergence rate of the algorithm and a method to quantify the influence of boundaries upon the solution. An adaptive meshing technique, based upon a kd-tree decomposition of the solution space, is presented. It provides a more efficient solution than regular, non-adaptive, meshes.

We are optimistic that our approach may prove effective at generating lifelike motion, which is a key requirement for compelling computer animation. Techniques such as the coupling of simplified dynamic models to kinematic animation methods promise to extend the applicability of our approach to systems of higher dimension, such as animated humanoids.

Acknowledgements

The existence of this thesis owes everything to Beate's hard work and the encouragement she gave me. Though it is not visible, her contribution was immense.

I am grateful to my supervisors, Bernard Buxton and John Gilby. Bernard helped me clarify my ideas and provided a regular opportunity to discuss the research. John took the time to explain mathematics to me, and to understand the details. He provided many ideas and helped structure the research and the thesis.

Beate's parents, Gunther and Helga, and my parents, Walter and Freda, helped us in many ways, particularly when we decided to buy a house. I am thankful for their support; life would have been much more difficult without it.

Contents

List of Figures

List of Tables

List of Symbols

A	the set of available control vectors
a	control vector
a^*	the optimal control vector
α	the open-loop control input, $\alpha = \alpha(t)$
α^*	the optimal open-loop control input, $\alpha^* = \alpha^*(t)$
C^n	denotes continuity of n^{th} derivative of a function
D	the gradient operator
δ_n	the change in value of node, n, during an update
e	pointwise error estimate, used for mesh refinement, $e = e(x)$
f	system dynamics function, or plant, $f = f(x, a)$
g	acceleration due to gravity
γ	convergence threshold
h	integration timestep
$\hat{h}_i$	time required to step to a cell boundary from node position, x_i
I	the influence metric, $I = I(x)$
$\tilde{I}$	the direct influence metric
$\hat{I}$	the approximate influence metric
J	cost accumulated along a trajectory
J^{kin}	Jacobian matrix relating end effector velocity to joint angle velocity
k	number of dimensions in the computational domain
l	instantaneous cost function, $l = l(x, a)$
L	angular momentum
λ	discount factor
m	membership of the reachable set, $m = m(x)$, $m \in [0, 1]$
p	position of the particle in the Bush problem
q	generalised coordinates

Q	generalised force vector
s	vector of inter node spacing on a regular mesh (used to describe AGDP)
Σ_{vp}	number of additional value passes performed in one iteration
t	time
T	node update map, $T = T(U)$
Θ	vector of joint angles, used to describe the pose of an articulated figure
θ	joint angle, e.g. of pendulum or double pendulum
u	approximate value function, $u = u(x)$
U	vector of approximate value value function at nodes within domain
v	value function, $v = v(x)$
V	vector of value function at nodes within domain
w_i	weight associated with vertex i for interpolation, aka barycentric coordinate
Ω	the computational domain (the domain)
x	vector of coordinates in computational domain
X	vector giving the position and/or orientation of end effectors in cartesian space
y_x	coordinates of trajectory in computational domain, $y_x = y_x(t)$, $y_x(0) = x$

Chapter 1

Introduction

Computer models of the human body are used for many purposes. They feature in computer games, in films and television. They are used in the design and engineering of products such as vehicles and clothing. They are central to many applications of virtual reality. They are used, in the field of biomechanics, for medical and sporting purposes. Increasingly, we will see digital humanoids in a range of Internet applications, and in many other places besides.

There has been much progress in the appearance and accuracy of these models. Scanning devices are available, from companies such as C3D, Cyberware, Hamamatsu, TCTi, and Wicks & Wilson, to quickly capture accurate three dimensional surface models with photorealistic texture. Once scanned, it is possible to articulate these surface models by a process that amounts to inserting a model of the skeleton [106, 116]. The deformation that the skin exhibits, owing to the motion of underlying tissues as the skeleton articulates, has also been modelled at various levels of detail. At one end of the scale, multi-layered models have been constructed that explicitly model the geometry and deformation of underlying tissues and the elastic properties of skin [84, 109]. At the other end, simpler geometric methods produce algorithms that are suitable for real-time animation [5]. Methods for modelling cloth are becoming much more practical and can provide clothes for digital humans that drape and move realistically [8, 103, 104, 105]. Even hair has been modelled, using a physically based approach, so that it sways with head movements and can blow in the breeze [114].

In spite of all this work, much remains to be done. In particular, more automation is required to make the process of model construction quicker and more accessible. Problems, such as how to locate the skeleton and its joints within a scanned surface model, and the anatomical accuracy of skeletal models, require more work. However, if we seek to create lifelike animation with these models then the question of how to make them move may be more pertinent. It is not sufficient merely to have models that look good. They must also move well.

With the right kind of motion it is possible to create and sustain an 'illusion of life', almost regardless of how well the figures are rendered [45, 46]. If the motion is somehow 'wrong' then the illusion will not work. This is an important point and is discussed in more detail in section 1.1.

There are essentially two methods in widespread use today that are capable of providing lifelike humanoid motion. The first is to employ a skilled animator. The second is to use motion capture. Both of these are used in the production of films, computer games, etc. Sometimes they are combined, when an animator is employed to post-process captured motion data.

Motion capture allows us to record the motion of a real person and then replay it within a computer generated image sequence. Typically the actor must wear a special suit and is tracked either with a magnetic system, an electromechanical system, or with several strategically placed cameras [53, 26]. It is necessary to 'massage' the motion data before it will fit a computer model that never exactly matches the original actor. Once this is done the replayed motion looks highly realistic, because it is essentially real motion. It is also possible to process captured motion data, using a variety of techniques to add effects to the motion. Most notably, it is possible to process the motion so that it will look right when replayed using a model of quite different proportions from the original actor [31, 70]. Of course, there are limits to how far it is possible to 'stretch' a captured motion sequence; a walking sequence cannot simply be transformed into a running sequence.

Motion capture is useful for entertainment applications where pre-recorded motion can be replayed in front of an audience. Films, for example, can use motion capture to insert real motion into imaginary settings. Computer games also use captured motion sequences, but hand some control over events to the player(s). In this case the pre-recorded motion sequences are combined in some way, during play, to form the motion seen by the player(s). Motion capture is also useful for applications where there is a need to analyse real human motion, such as in sports science.

Motion capture is not so useful when there is no convenient source of appropriate motion data. In such cases a skilled animator may be employed. For example, the BBC series 'Walking with Dinosaurs' employed animators to design the motion of the dinosaurs, with advice from paleontologists and other experts.

Animators use computers to design motion by, in essence, specifying key frames and allowing the computer to fill in the frames in-between. This approach echoes the traditional cell animation process, in which senior animators would draw the key frames and have junior animators perform the tedious task of *inbetweening*, that is carefully redrawing the animated figure in each frame in-between [95]. Today, computer based animation tools do the inbetweening and a little bit more. For example, inverse kinematics tools facilitate the manipulation of three-dimensional articulated figures, as well as handling the interpolation between poses. Physically based modelling (simulation) can handle some parts of the animation, such as clothes, hair and passive or mechanical objects.

However skilled animators are rare and, despite the technological advances, the process is still labour intensive and therefore costly. To engage an audience, the animator must create a performance, effectively taking the role of an actor. Instead of using his or her own body to create the performance, the animator must coax it out of an abstract computer based figure. That is a skilled task.

1.0.1 Motion Synthesis

Some applications require motion to be generated automatically in response to unpredictable events or inputs. For example, computer games have animated characters that are either controlled by, or interact with, the players. The computer generates their motion in response to the unpredictable input of the players. The task of generating such motion automatically is sometimes called *motion synthesis,* and it is the subject of this thesis.

As in computer games, a requirement for motion synthesis arises whenever the aim is to create an illusion of life within an interactive context. Many virtual reality applications share this aim and hence the requirement for motion synthesis. For these applications, neither motion capture nor key-frame animation can, by themselves, provide appropriate motion without becoming repetitious. Often a library of short motion sequences is employed. These are concatenated, perhaps with some form of interpolation to smooth the transitions, to form the whole motion. The inherent repetition of stock 'moves' quickly makes the figure appear mechanical and tends to destroy the perception of life. What is required is a more flexible and powerful model of motion itself, capable of infinite variation, and of responding appropriately to each new situation.

The utility of motion synthesis for animation is not necessarily limited to interactive applications. In some circumstances it can be more appropriate or cost-effective than the alternatives. For example, flocks of birds and shoals of fish are handled well by Reynolds boids algorithm [79]. One day, motion synthesis may be more effective at generating motion data for dinosaurs than an animator is today.

There is another type of application for motion synthesis where the aim is to understand or predict human (or animal) motion. Examples include many of the applications of biomechanics, some machine vision applications, such as surveillance, and the development of legged robots. Obviously, these applications present different priorities for a motion model; accuracy and insight are generally more important than perception. Nevertheless there is substantial common ground between these two broad classes of applications. An accurate model of biological motion should also be capable of creating a perception of biological motion, though the inverse is not necessarily true.

It is thus our belief that motion synthesis will, one day, allow us to explore and interact with real or imaginary humanoid and animal characters within virtual worlds; and that it will do so in a more engaging and lifelike fashion than has previously been possible. We believe it would be useful both for *avatars*, controlled by a human participant, and for *agents*, controlled by a computer. Although avatars may have some motion information available, e.g. from headsets, gloves, etc., this information is generally incomplete and could usefully be supplemented by a motion synthesis system to increase realism and flexibility. Unfortunately, the motion synthesis techniques available today are either not sufficiently lifelike or are too specific or too fragile to be widely adopted, as discussed in the next chapter. The focus of this work, therefore, is upon general purpose and, as far as possible, automated methods for motion synthesis.

In this research the focus is upon the gross motion of the body, i.e. the large scale movements of torso, head and limbs. No effort is made to address the fine scale motion of facial features, fingers, hair, etc. These phenomena generally require a somewhat different approach[66, 114] and are beyond the scope of this thesis.

1.1 The Importance of Motion

The main motivation for the focus on motion quality for animation comes from the psychology literature[25, 24, 54, 55, 56] and also from the experience of animation studios[95]. The evidence from these sources helps to establish the extent to which motion quality affects our perception of animated human figures.

Research into the perception of human motion has sought to establish how much information is available to a human observer from motion cues alone. This question has been tackled using video clips that show only points of light positioned at a person's joints [45, 46]. In this way all other visual cues are removed and motion information is effectively isolated. This *point-light technique* originally involved strapping reflective bicycle tape around the joints, using spot light illumination and adjusting the monitor's contrast to maximum and brightness to minimum. Nothing is seen of the subject but bright moving blobs of lights. On first seeing such a video clip, it is perhaps surprising how efficient we are at interpreting this information. Without motion the display is meaningless, but almost any motion makes the display immediately recognisable as human.

Research in the late 1970s by Cutting and Kozlowski established that observers could accurately recognise themselves and their friends [25] and determine gender from point-light displays. Subsequently, Cutting synthesised human walkers using a computer and was able to demonstrate that an impression of gender can be created by varying the relative hip and shoulder width and the degree of movement at hip and shoulder [24]. Montepare et al studied the potential for motion cues to reveal certain emotions [54]. They found that subjects were able to identify sadness, anger, happiness and pride from gait information at better than chance levels. Their experiments also revealed that gait characteristics such as amount of arm swing, stride length, heavyfootedness, and walking speed differentiated the emotions expressed by the walkers. Later Montepare et al studied the effect of age on the impressions created by gait information [55, 56]. They found that observers' judgements of traits, such as happiness and strength varied systematically with the walkers age, even though age itself could not be identified accurately from point light displays.

There is perhaps only one published study on the importance of computer graphics rendering methods to perception of motion [39]. Hodgins et al measured the sensitivity of observers to variations in motion sequences using a stick figure representation and a more detailed polygonal model. Two video clips were shown with identical graphical representations but slightly different motion sequences, one reference sequence and the other with added 'noise' or with exaggerated torso rotation. Subjects were asked to say whether the sequences were the same or different. Subjects were found to be, on average, only slightly less sensitive to differences in motion sequences using the stick figure representation.

Further evidence for the perceptual significance of gross body motion comes from the experience of animators at Disney studios. During the 1930's animation was developed from simple beginnings to a sophisticated art form, thanks to the popularity of cartoons in cinema and to the efforts of those commercial studios, notably Disney, that fed the public's appetite for animation [95]. The animators learnt that their animations had to have certain qualities to make them look 'right' and hence engage the audience. They formulated a set of principles [95] for animation including:

- squash and stretch,
- anticipation,
- follow through,
- slow in and slow out,

- weight and balance.

If the animation did not satisfy these principles then an audience would not identify with the action and the sequence would fail to engage the audience. These principles appear to be natural consequences of the laws of dynamics as they apply to gross body motion. The animators were effectively trying to put the qualitative aspects of dynamic behaviour into their animations in order to create an illusion of life.

An interesting aspect of cartoon animation is its ability to create and sustain this illusion of life, even though a character's appearance is often quite dissimilar from their real life equivalents. For example, Mickey Mouse cartoons clearly create the perception that Mickey is a young boy, despite the obvious visual differences. Age and gender are both communicated through motion information.

In some cartoons, otherwise inanimate objects such as a sack of flour, or a chair, come to life and start walking. These objects not only look unlike any familiar animal, they do not even have the same structure as any animal. The motion that is given to them takes account of their apparent physical limitations, yet retains enough of the character of biological motion to create the perception that they are somehow alive.

1.2 An Approach to Engineering 'The Illusion of Life'

Dynamics are, as we have seen, an important factor in our perception of gross body motion. If we can get the dynamics right then the motion will look right because it will automatically respect the principles of animation. The importance of dynamics for lifelike animation may be compared to the importance of light for photorealistic image synthesis. In the latter case, it has become possible to generate highly realistic images by using a model of light and its interaction with matter [107]. In the same spirit, we have chosen to approach the problem of engineering 'the illusion of life' by building a model of the gross motion dynamics of living creatures.

We hope that such a model of gross motion dynamics may also prove useful as a model of biological motion, or at least to guide the construction of good, detailed models of biological motion.

One part of the problem, solving the equations of motion for articulated figures, has already been addressed [6, 7], and is now well understood. This knowledge can be applied to simulating the motion of passive figures and has been used in vehicle crash simulation since the 1970's [65]. Computer models for crash simulation are often based upon physical crash test dummies, whose dynamic properties are well known. Crash test dummies, in turn, are calibrated against data collected from cadavers. For crash situations, where there is little time to react, a dead body provides a reasonable approximation to a live one. However, in most other situations, we require a model of the *active dynamics* of our subjects. For this we need to know the muscular forces that drive motion. In other words, we must solve the control problem.

Whilst Newton's laws of motion are well established and accepted, control is different. There is no such such fundamental law by which control can be determined. However, there is considerable evidence that animal motion is generally near optimal [1, 2].

Certainly, it seems reasonable to expect that frequent behaviours, such as walking and running, will become optimised through practice. It also seems reasonable to expect that animals use behaviours that are a natural optimum for their own physical structure and capabilities. For example, people are not often observed hopping to catch a bus; it is more efficient to run.

One example of optimal human motion is the walking gait. The existence of so-called passive walking machines [52] demonstrates that walking is an efficient gait for humans. Passive walkers, with legs that are structurally and dynamically similar to human legs, can walk down a slight incline without any source of power.

The principle of optimal motion for animals turns the general control problem into a more specific problem. We seek the control input to apply to a passive dynamic model that optimises the resulting motion. This is the *optimal* control problem.

An important advantage of the optimal control approach is that it provides a powerful method to characterise the motion of living creatures. If we can solve the optimal control problem then we have succeeded in narrowing the search for lifelike motion. Instead of searching through the space of possible trajectories we need only seek an objective function that yields the desired results.

Moreover, although we do not know exactly what objective functions will yield lifelike motion in every case, there are some general principles to guide the selection of an appropriate function. Firstly, animals tend to conserve metabolic energy [2]. Secondly, goals, such as catching a bus, must be achieved in a timely fashion if they are to be of benefit. In general, the degree of urgency required to achieve a goal will vary and will depend upon context, for example more urgency is required if the bus is about to leave. Therefore an appropriate objective function will allow a balance to be chosen between conserving metabolic energy and achieving a goal quickly.

However, before we need worry about the finer points of choosing an objective function, we must solve the optimal control problem. That is where the real difficulty lies, and that is the problem we have attempted to solve in this work.

The work of Witkin and Kass on, what they termed, spacetime constraints [111] has already demonstrated that the combination of dynamics and optimal control can produce lifelike motion. Inspired by the animated short film, Luxo Jr. [67], they took approximate hopping trajectories, parameterised them and applied an optimisation algorithm. The optimised trajectories closely resembled the hopping motion previously generated by animators for the film, and look exactly like one would imagine a desk lamp might hop, if it could. The spacetime constraints approach is reviewed in more detail in Chapter 2.

A drawback of the trajectory optimisation approach of Witkin and Kass is that it is not well suited to online simulation. It can also have problems converging upon a suitable solution. For these reasons we have chosen to pursue an approach based upon Bellman's principle of optimality, often called the dynamic programming principle [11]. This provides a method for constructing optimal feedback controllers, which are suitable for online simulation and hence interactive applications.

Dynamic programming provides a unifying principle for the analysis and solution of optimal control problems [108]. When applied to continuous systems, such as human and animals models, dynamic programming leads to the Hamilton-Jacobi-Bellman (HJB) equation [10]. A solution to the HJB equation for a particular control problem provides an optimal controller in feedback form.

We have therefore addressed the optimal control problem by constructing an algorithm to solve the Hamilton-Jacobi-Bellman equation for animation.

The primary limitation of this approach is that the cost of such an algorithm, in both space and time, rises exponentially with the number of dimensions of the *phase space* of the model. The phase space is the space formed by the generalised position and velocity coordinates of the model, and therefore has twice the number of dimensions as the model has degrees of freedom. This exponential rise in computational cost is sometimes called the 'curse of dimensionality'. The problem is so severe that we do not present any examples with more than two degrees of freedom.

With so few degrees of freedom it may be difficult to see how this approach could ever be used to generate lifelike animation for something as complicated as the human body. Fortunately, problems with many dimensions can be often be decomposed into lower dimensional sub-problems [10]. Also, simplified dynamic models can be used to insert greater realism into the animation of complex articulated figures, without necessarily handling every detail of the animation [70, 100, 96, 21]. For these reasons we are optimistic that the HJB equation can play an important role in generating lifelike animation for models with many degrees of freedom.

1.3 Thesis Outline

1.3.1 Contributions

Our main contribution is to describe the design of an efficient method for solving the HJB equation for the purpose of animation. We describe a number of specific improvements upon published methods for solving the HJB equation including:

- a method to obtain rapid convergence using short integration timesteps by calculating the limit of an infinite series of updates to a single node in the computational mesh,
- a method to measure the extent to which an arbitrary boundary value distorts the solution, and to obtain information about the quality of the solution,
- a comparison of adaptive, kd-tree based meshing schemes for solving the HJB equation,
- a new method for calculating the approximate influence that each node in the computational mesh has upon regions of changing control within the mesh, known as the influence metric,
- a solution to the swing-up and balance problem for the acrobot, a double pendulum with a single actuator at its mid-joint, using a solution of the HJB equation.

1.3.2 Plan of Thesis

The remaining chapters of this thesis are laid out as follows.

2. Previous Work A review of the literature on motion synthesis is contained in this chapter. It starts with kinematic and mixed kinematic/dynamic motion synthesis. Trajectory optimisation techniques for obtaining open loop control are then described. Finally, dynamic methods that include feedback are reviewed.

3. The HJB Method This chapter explains the origins and mathematical basis of the Hamilton-Jacobi-Bellman equation. A simple method for solving the HJB equation numerically is described.

4. Convergence, Boundaries & Meshing Some limitations of the simple solution method described at the end of Chapter 3 are described and addressed. First, the question of how to obtain rapid convergence without sacrificing accuracy is explored and a solution is proposed. Second, the question of how to understand the influence of boundaries upon the solution is explored and a method is proposed to quantify the influence of those boundaries. Finally, an adaptive meshing scheme is described for improving solution efficiency.

5. **Building a Controller** Issues related to the construction of a controller, using a method for the solving the HJB equation, are addressed in this chapter.

6. **Verification and Behaviours** The performance of variations of the adaptive meshing algorithm are assessed. The HJB method is applied to generate a combined swing-up and balance controller for the Acrobot.

7. **Conclusion** The thesis is summarised and our main conclusions stated. Our contributions are discussed and recommendations for future work are given.

Chapter 2

Motion Synthesis Review

This chapter reviews previous work on motion synthesis. We begin with a description of inverse kinematics, and its application to motion synthesis. Some mixed kinematic and dynamic systems are described before discussing fully dynamic motion synthesis. Open loop techniques, based upon optimising a single trajectory, are described. Finally, methods incorporating feedback are reviewed.

2.1 Kinematic and Hybrid Methods

Inverse kinematics is a useful and popular tool for manipulating and animating articulated figures. It is a vital component of many motion synthesis systems. We now describe the fundamentals of inverse kinematics, in order to provide an overview of what problems it does and does not solve. We shall then describe its application to animation and to motion synthesis in particular.

2.1.1 Inverse Kinematics Review

The *forward kinematics* problem is to determine the position and/or the orientation, X, in Cartesian space of some part, or parts, of an articulated figure given the angular coordinates of the joints, Θ. In the robotics literature the part of interest is generally the *end effector*, i.e. the tool attached to the end of a robotic arm. However, in general, we may be interested in any part of the articulation. Thus, for a humanoid figure, we may be interested in the position of an elbow, the direction of gaze or perhaps the position of the centre of mass [20]. The term end effector has come to be used to refer to any of these parts. We shall use it to refer to any function of pose that may be useful for manipulating articulated figures. Figure 2.1 illustrates the joint angles and end-effectors for a simple articulated figure.

Forward kinematics generally has a single, well-defined, solution, which can be found by concatenating the chain of translations and rotations in a 4×4 homogeneous transformation matrix. The simplicity and speed of this calculation allows the pose of articulated figures to be

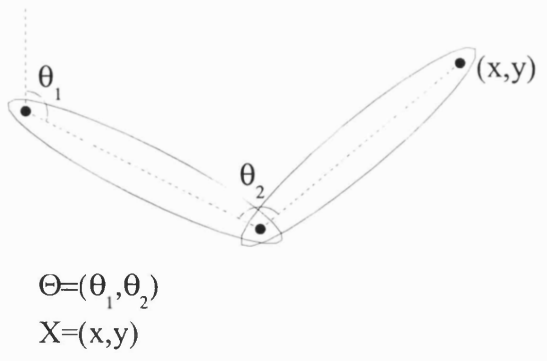


Figure 2.1: a simple articulated figure

conveniently stored as the set of joint angles. It is a relatively small burden to solve the forward kinematics problem each time the figure is redrawn in a new pose.

Unfortunately, the use of joint angles presents a problem. People find it difficult to manipulate articulated figures via the joint angles. It is easier to manipulate an end effector directly in Cartesian space. To support this, the inverse problem, *inverse kinematics*, must be solved, i.e. to determine Θ given X.

The method that is most widely used for animation, known as the pseudo-inverse solution, works by calculating the velocity of Θ given the velocity of X [107]. These are related by a Jacobian matrix, called the kinematic map, defined by

$$J^{kin}(\Theta) = \frac{dX}{d\Theta}, \tag{2.1}$$

and hence

$$\dot{X} = J^{kin}(\Theta)\dot{\Theta}. \tag{2.2}$$

There are two points to note about solving equation 2.2 for $\dot{\Theta}$. Firstly, it is generally underdetermined for humanoid and animal figures. Thus, the matrix, $J^{kin}(\Theta)$ may be rectangular with the result that there is some freedom in the choice of solution. Secondly, singularities can occur at certain configurations when a degree of freedom in X is lost. Such configurations can occur, for example, when an arm is fully extended or when it is folded up. A consequence of the presence of singularities is that equation 2.2 becomes ill-conditioned near singular configurations, which can result in unacceptably high joint angle velocities.

A damped least squares solution that minimises $\left\|\dot{X} - J^{kin}(\Theta)\dot{\Theta}\right\|^2 + \left\|\dot{\Theta}\right\|^2$ is commonly used to address both ill-conditioning near singularities and the fact that $J^{kin}(\Theta)$ may be non-square [107].

Armed with a solution to equation 2.2, it is possible to manipulate an articulated figure by defining a function for $\dot{X}$ and integrating Θ numerically. Equation 2.2 must be solved at each integration step to obtain $\dot{\Theta}$. For example, suppose X is the Cartesian position of a hand. To move that hand to a given position we can define $\dot{X}$ as a vector from its current position to the desired position. The hand will then move towards the desired position at a speed which is determined by the magnitude of $\dot{X}$.

A useful feature of the pseudo-inverse solution is that X can be composed of several distinct end-effectors. Hence, different parts of the figure can be manipulated simultaneously. Alternatively, constraints can be defined as additional components of X. These can be used, for example, to keep both feet fixed upon the ground by holding the velocity of the feet at zero.

One problem with the pseudo-inverse solution, as described above, is that it treats all joints equally. Hence, it might bend a humanoid at the waist to reach a nearby object, when that same task could have been accomplished more easily by moving only the arm. A weighted least squares solution to 2.2 can be used to address this problem. By adjusting the penalty associated with each component of $\dot{\Theta}$ we can, in effect, alter the relative stiffness of each joint.

An alternative to the pseudo-inverse solution is to employ a general nonlinear optimisation algorithm, as described by Zhao and Badler [117]. In this case, the objective function, $G(X)$, is formed as the weighted sum of a number of individual goal functions, $g_i(X)$, i.e.

$$G(X) = \sum_i w_i g_i(X). \tag{2.3}$$

The goal functions specify desired positions or orientations for end effectors. For example, one useful goal function is the square of the distance between a part of the figure and a fixed point in Cartesian space. Optimisation is carried out in terms of the joint angles. The Jacobian matrix defined in equation 2.1 is therefore used to find the gradient of the objective function with respect to the set of joint angles, i.e.

$$\frac{\partial G(X)}{\partial \Theta} = \sum w_i J_i^{kin} \frac{\partial g(X)}{\partial X}. \tag{2.4}$$

Zhao et al use the BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm for their optimiser [76]. BFGS is a quasi-Newton algorithm that builds an approximation to the Hessian (matrix of second derivatives) of the objective function. It supports constraints that can be specified as upper or lower bounds on each independent variable, which is useful for specifying joint limits. It does not support more general constraints.

Compared to the pseudo-inverse method, the nonlinear optimisation method is relatively efficient, since it does not require the repeated solution of a linear least squares problem. It also provides a method for finding a pose that is a best fit, in a least squares sense, to a number of competing goals. However, the path followed by the figure during the optimisation process is not smooth [117]. The pseudo-inverse method computes a smooth path through Cartesian space, which may be more useful for animation tasks.

An alternative method for positioning articulated figures is to use a physically based approach, and change the laws of physics to suit the task [4]. First order dynamics (Aristotelian dynamics), where $F = mv$, avoids the problems of oscillation and overshoot associated with Newtonian dynamics, whilst retaining an intuitive, physically based, response to user input. This method also provides a solution to the problem of finding a pose that satisfies hard constraints, such as the non-penetration of rigid objects, whilst simultaneously optimising softer goals, such as finding a comfortable or relaxed pose.

2.1.2 Inverse Kinematics for Animation

Inverse kinematics is used within keyframe animation systems for interactive manipulation and pose interpolation [107]. The animator can edit a pose by dragging end effectors around on screen to specify key frames. The system fills in the poses between the key frames by interpolation. However, the interpolation process is usually not as simple as interpolating the joint angles. In general there are constraints that the interpolated poses must satisfy, such as keeping feet in contact with the floor. Joint angle interpolation cannot guarantee to satisfy such constraints. The pseudo-inverse solution, however, can be used to perform a smooth interpolation, whilst simultaneously satisfying the constraints.

2.1.3 Mixed Kinematic and Dynamic Motion Synthesis

Inverse kinematics has been applied to higher level motion specification than the keyframe and interpolation model. Animation systems have been built that combine inverse kinematics with some form of dynamic analysis to automatically add elements of dynamic behaviour to the animation. That is the approach taken in the work described next.

Girard and Maciejewski describe a system for animating walking and running motions based upon the pseudo-inverse method and some simple dynamics [29]. Their system allows an animator to specify a curved path for a legged creature to follow, a set of gait parameters and a curved path for each foot to follow whilst in the air. Given these parameters the system will generate an appropriate animation. The gait parameters are: the relative phase of each leg, the time that each foot is on the ground and in the air and the distance travelled by the body whilst a given foot remains on the ground. The pseudo-inverse method is used to drive the animation by propelling the body forwards whilst keeping each foot either on the ground, or following the curved path through the air. The centre of mass is allowed to accelerate vertically under the influence of gravity and the simple rule that each leg provides an equal upward force whilst on the ground. The acceleration of the centre of mass horizontally and the rotational acceleration required for turning are accounted for by placing the feet in such a position that they could provide the necessary acceleration.

Girard and Maciejewski applied their animation system to bipeds, quadrupeds and creatures with more legs.

The simplified dynamics employed in this animation system capture the most visually obvious aspects of the dynamic nature of walking and running. With no dynamics the body appears to 'float' over the legs in a way that is obviously non-physical. However, visual problems related to the simplicity of the dynamic model remain. For example, limbs can still accelerate in ways that are not physically possible.

The following two models of human walking use knowledge about real human walking to drive an inverse kinematics engine and they also improve upon the dynamics of Girard's system.

Bruderlin et al describe a hybrid kinematic/dynamic approach to walking motion synthesis [21]. A simple dynamic model was constructed specifically for the purposes of human walking. It is essentially a 2D model where the 'stance leg' is treated as a straight piston arrangement, and the 'swing leg' as a double pendulum. The double support phase, when both feet touch the ground, is handled kinematically, as is the motion of the upper body. Biomechanical data is used to generate kinematic motion for the upper body, which interacts to some extent with the dynamic model. An iterative root finding procedure is used to find two torque values for the swing leg at each step in order to place the foot at the desired new position. The model appears to be largely successful in generating a range of walking motions, which can be parameterised in a number of ways (such as step length, pelvis rotation, etc).

Ko and Badler describe another mixed kinematic/dynamic walking model [47]. A similar sort of kinematic walk generation system is used. This one utilises biomechanical data to generate walking motions given a curved path to follow. An inverse dynamics system is used to find the joint torques required for this motion. Balance constraints can then be enforced by modifying the whole body motion so that the computed torque at the ZMP[1], between the supporting foot and the floor, becomes zero. Comfort constraints are also dealt with by further modifying the motion until joint torques fall within empirically determined comfortable levels. Heuristic techniques are used to modify the kinematic motion in order to comply with balance and com-

[1]ZMP stands for Zero Moment Point. This concept is sometimes used in the design of walking machines in relation to inverse dynamic analysis of gait. If the connection between foot and floor is treated as a joint, then the moment of force about that joint should be zero since no torque can exerted directly upon the floor. The ZMP can thus be calculated using inverse dynamics, as the point at which the torque between foot and floor vanishes. If the ZMP falls outside of the supporting foot, then the instantaneous state of the system (including velocities) is impossible.

fort constraints. The virtual human can be asked to carry a load or walk into a strong wind, and the motion will be suitably modified in order to satisfy the balance and comfort constraints. The kinematic motion system also has a large set of variable parameters, including step length, step width, pelvis and torso rotation.

Both the walking model of Bruderlin et al and that of Ko and Badler make considerable use of knowledge that is specific to human walking. A knowledge-based approach of this sort can be effective at dealing with a specific case but usually does not necessarily lead to an understanding of how to synthesise a different class of behaviours or a dramatic variation, such as a stumble. Much effort is put into careful tuning of these systems in order to produce the desired results. Therefore in order to incorporate a new class of behaviour, not only must the requisite knowledge be found, but the system must be tuned again.

2.1.4 Motion Interpolation

A completely different approach to kinematic motion synthesis is that of *motion interpolation*. That is, taking a set of similar motion sequences, often captured motion, and interpolating between them to get new motion sequences. One example of this approach is the work of Rose and Cohen [80, 87]. Their method works by defining an *adverb* space with one dimension for each of a chosen set of adverbs, such as happily, sadly etc.. Each of the example motions is assigned a position within this adverb space. New motions can then be generated by choosing a new position within the adverb space and using that to interpolate the example motions. Rose and Cohen used radial basis functions to interpolate within their adverb space. Temporal coherence between the example motions is ensured by identifying keytimes, such as heel-strike and toe-off for a walking sequence, and adjusting the timing of each sequence so that the keytimes correspond. Inverse kinematics is used to enforce kinematic constraints in the resulting motion, such as preventing a foot from slipping along the floor.

Motion interpolation has the important advantage that it can create an infinite set of motions from a finite set of examples. It is well suited for online motion synthesis, and provides a high level of realism if given a good set of example sequences. The main drawback of motion interpolation is that it can only reliably produce motion sequences that lie within the set of examples. No general principles of motion are derived that may be effective at predicting motion that lies outside the set of examples.

2.2 Direct Trajectory Optimisation

None of the motion synthesis systems described so far meet our objective of combining dynamic correctness (i.e. the motion obeys Newton's laws) with a principle of optimal motion. In general, motion that is driven by an inverse kinematics solver will not be optimal because that solver optimises only the instantaneous joint angle velocities or the instantaneous pose. When only the instantaneous state of the dynamic model is optimised there can be no anticipation or planning, and without planning it is not possible to move efficiently. For example, one cannot jump up efficiently without first crouching down.

To generate natural, efficient motion using inverse kinematics, additional guidance or control is required. The mixed kinematic/dynamic motion synthesis systems that we have described provide additional guidance but they all have their drawbacks. Girard's system [29] provides very limited dynamics and requires that a skilled animator design aspects of the motion, such as the path of the swinging feet through the air. The systems of Bruderlin [21] and Ko and Badler [47] are inflexible and apply only to human walking.

Motion interpolation does not necessarily provide dynamic correctness or optimality. Just because motion A is dynamically correct and motion B is dynamically correct it does not follow that the average of motions A and B is also dynamically correct. A similar argument applies for optimality.

One way to address these problems is to use a fully dynamic model, i.e. a simulator, together with an optimisation algorithm to generate and optimise whole trajectories. We call this approach *trajectory optimisation.* If we can obtain a useful solution to a trajectory optimisation problem, we shall gain dynamic correctness and optimal motion. Furthermore, the approach is very flexible and can be applied to a wide range of models and behaviours.

2.2.1 Spacetime Constraints

The Spacetime Constraints paper of Witkin and Kass [111] describes an attempt to apply trajectory optimisation to animation. They generated hopping motions for a desktop lamp, like the one in the computer animated short film Luxo Jr. [67]. The original film used keyframe animation and was successful in creating the illusion of a living, hopping desktop lamp. The lamp was rendered in a realistic style, hence motion cues alone were responsible for sustaining the illusion of life. The lamp hopped just as one would imagine a lamp would hop, if it could. Witkin and Kass were also able to generate convincing hopping motions using their Spacetime Constraints method, e.g. see figure 2.2, with the advantage that the computer was able to do much of the work that a skilled animator would otherwise be required to do.

The Spacetime Constraints method treats trajectory optimisation as a two point boundary value problem, i.e. the start and end points of the trajectory, as well as its duration, must be chosen first. The problem is discretised in time and an objective function, G, defined as a sum over all time steps. For example, to provide a 'least effort' solution we minimise power

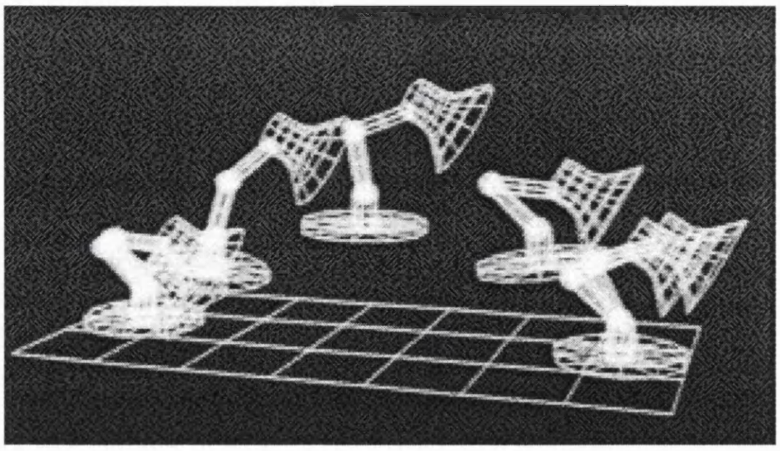


Figure 2.2: a hopping sequence generated by spacetime constraints (from [110])

consumption, e.g.

$$G = h\sum_i |f_i \dot{q}_i| \tag{2.5}$$

where h is the time step size, f_i is the vector of generalised forces produced by actuators in the system at step i (the 'muscles', if a lamp could have muscles), and q_i is the vector of generalised coordinates. The equations of motion and boundary conditions (beginning and end configurations) appear as constraints in the optimisation procedure, while the f_i and q_i are the independent variables to be determined. A finite difference scheme is used to relate q_i to its derivatives.

The problem can be put in canonical form, as a scalar objective function, a collection of independent variables and a number of scalar constraint functions. Once this is done a standard constrained optimisation algorithm can be employed to seek a solution. Witkin and Kass use a variant on sequential quadratic programming. This computes a second-order Newton-Raphson step in the objective function and a first-order Newton-Raphson step in the constraint equations. The two steps are combined by projecting the optimisation step into the space of steps that satisfy the constraint equations (the null space of the constraint step).

The Jacobian of the constraint equations and the Hessian of the objective function must be evaluated, and two linear systems solved, at each iteration. A sparse conjugate gradient algorithm is used to solve the linear systems with an $O(n^2)$ cost. Crucially, this takes advantage of the inherent sparsity of their problem formulation and can be used to trade accuracy for solution time. The examples they present took around ten minutes each to compute a two second sequence starting from a crude initial motion. A surprisingly short time on circa 1988 hardware.

Spacetime constraints is a very general and powerful method. The principles of animation (see section 1.1) are satisfied as a natural consequence of the use of optimal control. For example, Luxo crouches down before the hop then lifts its foot (base) up whilst airborne before extending it to absorb the impact. Behaviour can be controlled by selection of the optimisation criteria, or by imposing additional constraints upon the solution, for example if the minimum height at the midpoint of the hop is constrained then Luxo can be made to hop over an obstacle.

An interesting feature of this method is the fact that the equations of motion need only be satisfied approximately. This is fine for animation purposes, as the motion need only look right. High accuracy is certainly not required. Consequently, it is possible to obtain a trajectory that may appear reasonable, even if it is not physically possible. It is an open question, as to how far it is possible to 'stretch' the equations of motion in this way before believability suffers.

Obviously, the spacetime constraints method has its limitations:

- There are potentially very many independent variables and constraint equations
- The $O(n^2)$ cost of the conjugate gradients algorithm limits the number of time steps and degrees of freedom that can be handled (n in this case is proportional to the number of time steps and to the number of degrees of freedom).
- The user must supply a start point, end point, duration, initial trajectory guess and an objective function. If these are not well chosen then a reasonable solution may not be obtained. For example, the solver may converge to an undesired local minimum.
- It is only suitable for offline simulation.
- Implementation is complicated as it requires elaborate systems of equations and derivative matrices to be calculated.

2.2.2 Refinements to Spacetime Constraints

A number of refinements and variations on the spacetime constraints method have subsequently been published. Cohen focused on providing interactive control over the solution process [23]. He proposed four specific measures to this end.

1. The user is provided with the option to define *spacetime windows*; windows in time that allow the overall problem to be solved as a set of sub-problems.
2. Each degree-of-freedom was represented with a B-spline in place of Witkin and Kass's finite difference scheme.

3. The control forces were eliminated as independent variables using symbolic methods.

4. The ability to handle inequality and conditional constraints was included. This package of measures reduced the computational complexity of the method, and gave the user more control over the progress of the solution.

Liu et al reformulated the spacetime constraints method, using a hierarchical wavelet basis, in an attempt to reduce computational complexity [50]. The wavelet basis has not been adopted in later work on spacetime constraints.

Generating transitions between existing motion segments (from motion capture data) is a pragmatic application of spacetime constraints [81]. Motion transitions are conveniently short and often possess a simple structure, which makes them relatively tractable. Rose et al used a fast $O(n)$ inverse dynamics formulation to calculate joint torques from the trajectory. They used the BFGS algorithm (see section 2.1.1) to minimise the sum of squares of the joint torques. General constraints are enforced using the inverse kinematics technique of [117]. The torque minimisation and inverse kinematics are combined by simply applying them to different parts of the body, thus failing to meet the more general goal of minimising joint torques over the whole set of joints. They were able to generate motion transitions for a human body model with 44 degrees of freedom.

Spacetime constraints has also proved to be a useful tool for *motion editing*; that is modifying existing motion data (often motion capture data). Gleicher [30] borrowed the idea of motion displacement maps from earlier work on motion editing (see e.g. [112]). A motion displacement map is akin to an image warp, where a low frequency displacement is applied to a higher frequency signal. So, if $q(t)$ is the new trajectory of the generalised coordinates of the figure and $q_0(t)$ is original trajectory then

$$q(t) = q_0(t) + d(t) \tag{2.6}$$

where $d(t)$ is the low frequency displacement warp. This formulation allows detail in the motion to be preserved, whilst warping the overall motion. By using spacetime constraints to generate the displacement function, $d(t)$, Gleicher was able to apply constraints to the motion whilst retaining much of its original character. For example, constraints can be used to prevent unwanted effects, such as feet skidding across the floor or a hand losing contact with an object that it is meant to be holding. Constraints can also be used to edit motion in other ways, such as changing foot placement. Gleicher chose to dispense with the objective of minimising an energy term or the joint torques and chose instead to minimise the difference between the new motion and the original motion. If the modifications are not too dramatic then the resulting motion will retain most of the balance and dynamic appearance of the original. By dispensing with the need to solve an inverse dynamics problem and by using a relatively low frequency warp the computational complexity of the method is substantially reduced, making it practical for humanoid figures with many degrees of freedom. However, some judgement is required in selecting an appropriate frequency for the control points of the displacement function.

A practical application of this work by Gleicher is *retargetting motion* [31]. That is, taking motion intended for one individual (often captured motion) and applying it to an individual of different proportions. The method appears to work well for this application.

Popovic and Witkin put dynamics back into spacetime methods for motion editing [70]. To achieve this they addressed the scalability problem by simplifying the dynamic model, so that it has fewer degrees of freedom than the original figure. Motion from the original figures is mapped onto the simplified figure, the spacetime constraints problem is solved and the resulting motion mapped back onto the original figure. A method to map motion between figures with different numbers of degrees of freedom is therefore required. The method they present for this purpose uses 'handles', functions of pose such as the position or orientation of points on the body (i.e. end effectors). The mapping works by solving the inverse kinematics problem to equate handles on the two figures. Obviously the resulting motion is not precisely dynamically correct, and some judgement is required to choose a simplified figure and to choose a set of handles to perform the motion mapping. Nevertheless, the method has some attractive features. For example, Popovic and Witkin show how it can be used to generate a limping run, from a non-limping one, by hobbling the simplified model.

Torkos and van de Panne also used a simplified dynamic model to work around the problem of scaling the spacetime constraints method up to a model with many degrees of freedom [96]. Their method approximates the dynamics of a cat with two points masses, roughly in the middle of the pelvis and the shoulders, joined by a spring. The animator specifies footprints for the cat, and inverse kinematics is used to try to place the feet on those footprints. Optimisation is used to find a trajectory for the dynamic model that best fits the constraints of comfort and dynamic correctness, whilst meeting the constraints of placing the feet on the footprints.

In the robotics literature similar optimal control methods, based upon sequential quadratic programming, have been shown to be capable of synthesising a walking motion for a simplified humanoid model. Hardt and Kreutz-Delgado [35] used similar methods to the spacetime methods, based upon sequential quadratic programming. Their model was a two-dimensional, five segment model, with a torso and a pair of two segment legs.

2.3 Indirect Trajectory Optimisation

The spacetime methods are characterised by the fact that they search by directly manipulating the trajectory. We shall refer to this as direct trajectory optimisation. The other way to optimise a trajectory, is to search by manipulating the control input as a function of time and repeatedly solving an initial value problem to evaluate each candidate solution. We shall refer to this as indirect trajectory optimisation.

Direct methods generate a sequence of candidate solutions, each of which satisfies the boundary conditions, such as the beginning and end points of the trajectory. Optimisation is used to find a solution that is dynamically correct. By contrast, indirect methods generate a sequence of candidate solutions, each of which is dynamically correct. Optimisation is then used to satisfy the boundary conditions.

In some cases the control sequence can be described with fewer parameters than the trajectory, in which case there will be fewer degrees of freedom in the optimisation with an indirect method than with a direct method. In such cases indirect methods may be preferable. However, the fundamental difference between direct and indirect trajectory optimisation methods is that they explore different spaces. Direct methods search the space of trajectories that satisfy the boundary conditions, whilst indirect methods search the space of permissible control sequences. Depending upon the nature of the problem, a good solution may be easier to find in one of those spaces than the other. Consequently, the choice of direct or indirect trajectory optimisation probably boils down to what works best for a particular application.

For animation applications, it is often not essential that a trajectory obeys Newton's laws precisely. Consequently direct trajectory optimisation has a significant advantage because, if it cannot find a solution that satisfies the laws of motion precisely, it will try to find the trajectory that comes closest to satisfying the laws of motion. Such a solution may well be good enough.

Nevertheless indirect trajectory optimisation has been applied in animation, as well as in robotics. Some of the published applications of indirect trajectory optimisation are relevant to our research and hence we describe them here.

2.3.1 Walking and Jumping Motions

The work of van de Belt [97] provides an interesting comparison to that of Hardt et al, described above [35]. Van de Belt used indirect trajectory optimisation to find a walking cycle for a three dimensional humanoid model with eight segments (torso, hip, upper legs, lower legs and feet). The optimal control solution she used is based on Pontryagin's Maximum Principle [108], which is a necessary and sufficient condition for optimality on a trajectory. Using this method, a two-point boundary value problem is solved by iteratively solving initial value problems forwards in time from the beginning and backwards in time from the end of the trajectory.

One point in the walking cycle was considered to be fixed. Thus the beginning and end points of the desired trajectory are in fact the same point, i.e. the fixed point. By exploiting the left-right symmetry of walking it is only necessary to generate a trajectory that is as long as half a walking cycle, or one step. This single step trajectory was short enough to make the problem tractable for indirect trajectory optimisation methods.

Zhao et al applied indirect trajectory optimisation to animating a jump sequence [118]. They chose a simple piecewise linear basis for their control functions and use the control points of this simple spline as the free parameters for optimisation purposes. They derived a method for calculating the gradient of their objective function in terms of the control points. Armed with this gradient information an efficient, general purpose, optimisation code can be applied to the problem. Zhao et al used the public domain LBFGS-B code [119].

The derivation relies upon being able to obtain the partial derivatives of the plant equation with respect to the control input and the generalised position and velocity vectors. The trajectory and control input are discretised in time and the derivative vector of the objective function obtained through repeated application of the chain rule.

There are a couple of caveats that should be noted. Firstly, this derivation is only valid if the equations of motion are smooth for the period of interest. The presence of impulse forces will invalidate the derivation. Secondly, there may be some difficulty in acquiring the partial derivatives of the dynamic system equation. Zhao et al used an algebraic expression for their equations of motion that was analytically differentiable. If a more general technique, such as the Lagrange multiplier method [7], is employed for solving the dynamics then the derivation may be more difficult to obtain. For example, the Lagrange multiplier method relies upon the solution of a linear system each time the acceleration vector is calculated (i.e. each time the plant equation is evaluated).

2.3.2 Neural Networks for Trajectory Optimisation

A similar method for indirect trajectory optimisation was published by Grzeszczuk et al [34]. The principal difference is that the dynamic model was approximated by means of a neural network. Three layer feedforward networks were trained to emulate the dynamics of a system, i.e. to predict the state of the system (including velocities) at some time $t + \delta t$, given the current state and control input. Once the emulators had been trained, they were able to learn sequences of control inputs to perform a given action, such as parking a car or safely landing a moon lander, using the *backpropagation through time* algorithm [82]. This is essentially the same as the approach described taken by Zhao et al, except that a simpler optimisation scheme is used, where each step taken is proportional to the gradient of the objective function plus a proportion of the previous step (the 'momentum' term).

The use of a neural network makes it relatively simple to calculate the gradient of the objective function, by analytically differentiating the neural network emulator and using repeated applications of the chain rule. The derivatives are guaranteed to exist because the neural network has the property of smoothness everywhere. Also, the emulator may be used to rapidly evaluate each control sequence. These two facts, combined with the rapid optimisation scheme, produced exceptionally short learning times on the examples they describe (tens of iterations).

However, before the method of Grzeszczuk et al can be applied it is necessary to obtain a neural network that provides a sufficiently good approximation to the dynamics of the system. They only present examples with dynamics that can be simulated very efficiently using established methods (a triple pendulum, a simple car, a moon lander and a mass-spring model with 23 masses). No direct comparison is given between the efficiency of the neural network and the simulator upon which it was trained. Furthermore, the mass-spring model (of a dolphin) was not solved with a single network, but using a hierarchy of six smaller networks. This was done because a single monolithic network would require a very large number of neurons for this problem compared to the hierarchical network. It therefore seems likely that the task of building and training a neural network to emulate the dynamics of more complex dynamic models, such as those required for a realistic human body, would be a difficult undertaking and might be prohibitive.

2.3.3 Virtual Wind-Up Toys

Van de Panne describes an interesting twist on indirect trajectory optimisation to build computer based clockwork toys [98]. These toy's are driven by simple finite state machines, each composed of a sequence of control poses, which are in turn defined by a set of joint angles. Proportional derivative servos constantly drive each joint towards the current control pose. The toy cycles blindly through the control poses, producing a motion that is much like a child's clockwork robot.

These finite state machines are different from pure open-loop control, where control is applied as a function of time alone. In this case control is a function of time and the current pose. However there is no real feedback, as nothing affects the constant cycling of control poses.

This approach has two significant advantages. Firstly, it reduces the number of parameters to define the control input. The dimensionality of the search space is thus 'number of

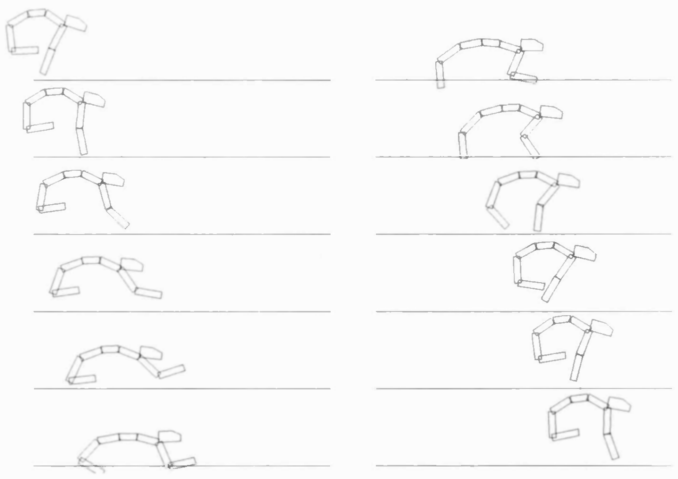


Figure 2.3: Van de Panne's cheetah creature running.
This animation should be read top to bottom, then left to right (from [98]).

poses' x 'number of actuators' (the gain constants for the PD servos are predetermined, as is the total period of the control cycle). Four poses appears to be sufficient to produce a simple bipedal walking motion. Secondly, it can produce a system that is robust to perturbations in the environment, such as an uneven ground, that might thwart a pure open-loop controller.

Van de Panne obtains a running motion for a two-dimensional, cheetah-like creature, with two legs and a flexible spine (figure 2.3).

In a later paper, he applies the same technique to a humanoid figure [102] with 13 joints, 14 segments and 19 degrees of freedom. A problem with this approach arises because the humanoid is unstable and any small disturbance to an unstable system will tend to grow exponentially over time. Consequently, as the length of the trajectory grows it becomes exponentially more difficult to find a controller that does not result in the humanoid figure falling over.

Van de Belt [97] did not suffer from this problem, of the figure falling over, because the trajectory was only one half of a single stride in length; short enough that the probability of a fall was low. Grzeszczuk et al [34] and Zhao et al [118] chose only models and behaviours that were unlikely or incapable of falling over. However, van de Panne uses trajectories of four seconds in length, which are long enough to make falling the most probable outcome.

To overcome the problem van de Panne uses a guiding influence to make the initial balancing easier. The guiding influence is then gradually removed, as the humanoid learns to walk. It also learnt skipping and running behaviours.

The guiding influence takes the form of an external torque applied to the torso to keep it upright. Van de Panne varies the level of support or guidance in three stages: full guidance, partial guidance and then no guidance. Whilst the choice of 3 stages may work in this example, more difficult problems may require more intermediate stages.

The instability of biped locomotion means that, in this search task, the space of useful solutions is small in comparison to the space of all possible solutions. For the learning process to be successful in this situation it is necessary to find a mechanism for rewarding partial progress towards the solution. This could be done by providing a 'smarter' objective function that can recognise such progress. However, the task of finding such a function may not be easy or obvious. Changing the system by providing external guidance makes learning easier by changing the shape of the objective function[2].

This approach has an appealing parallel with the guidance and support that a parent provides a baby as it is learning to stand and walk. In some sports, such as gymnastics, a similar kind of support may be arranged whilst the gymnast learns a new skill.

2.3.4 Look-ahead Methods

Another, distinct approach is to use a dynamic model to explore possible future trajectories from the current state and use the information gained to determine control input. We refer to this family of methods as *look-ahead methods*.

A successful application of this idea to the acrobot (see section 6.3) was explored by Huang and Van de Panne [43, 44]. They used a decision-tree search algorithm that works in the following way. Control input is chosen from a discrete set of available controls and applied for a fixed interval, in this case 0.02s. At each stage, a tree of future trajectories is constructed with a new branch for each available control input. The size of the search tree is limited by depth and by a set of pruning rules that reject undesirable trajectories. An evaluation function is used to rate the 'promise' of each node of the tree, and then the control input leading to the most promising leaf of the tree can be chosen. Because the tree is updated at each stage it effectively transforms an open loop control input into a closed loop system that can cope with external disturbances and the instability of the acrobot. The trick is to carefully choose a set of pruning rules and an evaluation function, so as to get the desired behaviour before the tree grows unmanageable. Huang and de Panne report success in generating balancing, hopping, cartwheeling and flipping behaviours for the acrobot by this method.

One major drawback of look-ahead methods of this type is that the computation required for each decision may be excessive, forcing the entire motion to be generated off-line. Huang and Van de Panne provide figures indicating that the search process took ~1000 times more CPU time than the simulation. Another drawback is that there appears to be no systematic way

[2]It is interesting to note that van de Panne's method of guiding the optimisation is an example of *invariant imbedding*, or a *continuation method*, invented by Richard Bellman.

to choose the evaluation function and pruning rules. It is likely to be very difficult to find an appropriate evaluation function and pruning rules for a model with many degrees of freedom such that the decision tree did not grow unmanageably large in a very short time.

2.4 Feedback Methods

Trajectory optimisation methods generally require substantial computation for each new trajectory that is generated. The amount of computation prohibits the use of trajectory optimisation directly within an interactive animation system that must respond in real time to unpredictable inputs from a user or the environment. It is possible that future developments in trajectory optimisation techniques could facilitate the use of such techniques in interactive animation systems, for example by performing the optimisation incrementally as the animation unfolds. However, at present, methods that incorporate some sort of feedback control appear to hold more promise for interactive animation. The control methods used for legged robots are therefore relevant to our research because they demonstrate how feedback control can be used to generate motion when applied to a dynamic model. In robotics that dynamic model just happens to be a physical robot.

Early attempts at biped locomotion for robots limited themselves to *statically stable gaits* in which the robot is balanced at each moment as if it were stationary. This strategy simplifies the problem of how to control such robots but places severe restrictions upon their performance. These robots are characterised by large flat feet, and a low centre of gravity. They are in many respects similar to the clockwork robot toy with which we are all familiar. In practice, powerful ankle joints are required to maintain balance and a firm footing on uneven ground. The walking speeds achieved by these robots are disappointing and their design does not provide much freedom in choosing a gait.

Later research explored *dynamic balance* for legged robots. These efforts are more relevant to our research, as they promise to provide far greater realism in the motion they produce than would be possible with statically stable gaits.

2.4.1 Hopping and Running Robots

Marc Raibert et al. [77] achieved a breakthrough in legged locomotion by focusing exclusively on dynamic balance. Inspired by the conceptual similarity between a pogo stick and hopping on one leg, they built a robot with a single, springy telescopic leg. In contrast to the statically stable walking machines, this robot was only stable in a dynamic sense. It had no ankle and maintained balance by continually hopping. The first 'hopper' was physically constrained to move in 2 dimensions. A second version worked in three dimensions and was capable of hopping at

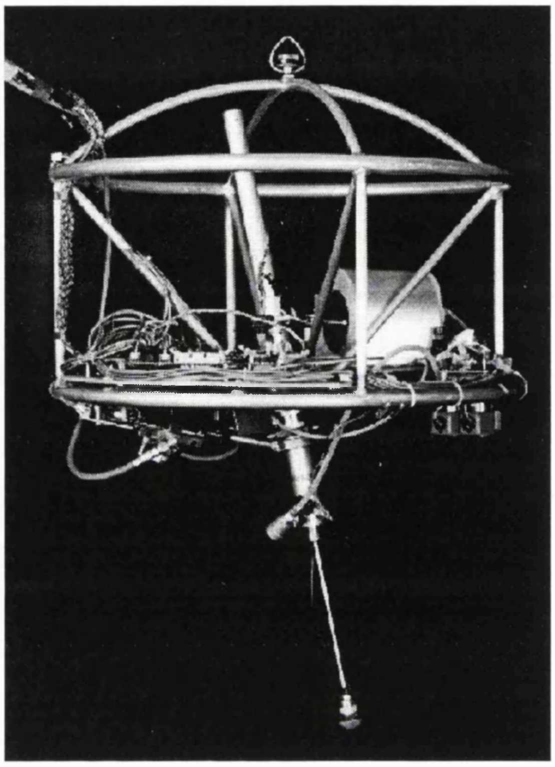
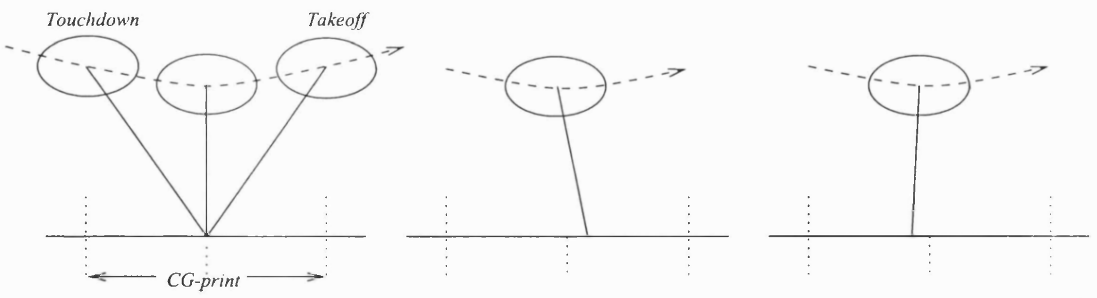


Figure 2.4: the original 3D hopping robot

speeds of up to 1.8 m/s.

The principles used to control the hopper and maintain its balance are very simple. Three things must be taken care of: hopping height, the attitude of the main body (the cage shown in figure 2.4) and horizontal velocity. If these things can be controlled then the robot will hop. The controller is designed as a state machine with two states: stance (when the foot is in contact with the ground) and flight. A pressure sensor in the foot switches the controller between the two states. The line traced by projecting the robot's centre of gravity onto the floor during stance is called the CG-print. By predicting the CG-print of the next stance phase and then planting the foot relative to the middle of the CG-print it is possible to balance the hopper and control its horizontal velocity (see figure 2.5). In this way the hopper can be made to hop in place or to travel at a stable velocity in any direction.

Raibert simplified the control problem by using a light leg attached near the centre of gravity of the robot. The heavier cage remains fairly stable during flight whilst the leg can be moved swiftly into position. It also becomes very easy to predict the CG-print from a knowledge of the stance period (which is roughly constant) and the horizontal velocity of the robot. In practice, the stance period is measured and used as a prediction for the next stance period. During flight, the leg is servoed to the required angle using a simple proportional derivative



Foot in centre of CG-print, stance phase is symmetric so horizontal velocity is unchanged

Foot forward of centre, the hopper slows down

Foot behind centre, the hopper accelerates

Figure 2.5: dynamic balance, and velocity control, via foot placement in hopping robot.

controller. This takes care of the horizontal velocity.

Two other things must be taken care of: the attitude of the main body (the cage) and hopping height. During stance the cage is servoed back to an upright position since it may begin to tilt during flight. Hopping height is maintained by injecting a fixed amount of energy into the pneumatic leg at the bottom of each hop. A pressurised air hose in the umbilical cable is used for this purpose.

Having perfected the hopping robot the next step was to build a two legged running robot. The key to this development lies in the understanding that running is just like hopping on alternate legs. The same principles which guide the hopping robot can be applied to a bipedal runner. Obviously some of the practical difficulties are new, such as dealing with the side to side swaying motion that arises in a biped and the need to make the non-stance leg shorter whilst it swings forward. However the bipedal gait has a clear advantage in locomotion since there is more time available to swing each leg forwards. As before, with the hopper, a 2 dimensional version of the running robot was built first (a boom was used to constrain the robot to run in circles). Finally the full 3D biped runner was built. Later on, the 3D biped was even programmed to perform a somersault [68].

The gait of this biped runner is not similar to a human, and is actually more reminiscent of a chicken. Nevertheless, the dynamic balance and speed of this robot makes it appear quite lively, if not exactly lifelike. The principal drawback of this approach, is that it only works for hopping/running motions and does not help with behaviours where the feet remain substantially on the ground, such as walking.

The approach adopted by Raibert and his associates was based on observation and experimentation. He built machines that permitted simple, approximate predictions to be made about their behaviour and focused attention on the principles of balance. He also tackled the problem in stages, adding complexity bit by bit.

2.4.2 Gymnastics and Other Sports

One of the people who worked with Raibert on the biped robots was Jessica Hodgins. She went on to develop simulations of various athletic and gymnastic events including running, cycling, high diving, vaulting and routines on the rings and asymmetric bars [36, 41, 38, 113]. Relatively human-like computer models were used for all of these simulations. The control module for the running motion is a direct descendant of the Raibert's hopping robots, albeit more complex. The running model contains 17 body segments and 30 controlled degrees of freedom.

The general approach for all the control modules is to use a detailed understanding of each particular motion to design the controller. Each controller is structured as a state machine with transitions between the states triggered by events such as *toe contact*, when the toe of one foot first touches the ground. A different set of control laws becomes active in each state. The control laws themselves are proportional derivative controllers, which may be used to set desired joint angles and angular velocities. Inverse kinematics is used to establish the desired joint angles and angular velocities in order to achieve a goal such as placing the foot correctly for balance during running.

The visual quality of the running motion is relatively good: the models have knees, ankles and even an articulated 'toe'. Hodgins et al show a frame by frame comparison of the male runner and a real male running on a treadmill [37]. They appear to be very similar. However, whilst viewed in motion, the simulated runner looks somewhat 'wooden' in the upper body. The level of articulation in the lower body apparently provides good realism, but the upper body is modelled with too few degrees of freedom to appear lifelike. For instance, the torso is treated as a single rigid piece and the shoulder blades are fixed relative to the torso.

An attractive feature of Hodgins approach is that a controller, once designed, need only be re-tuned for it to work with a similar model of different proportions. Furthermore, the process of re-tuning the controller can be automated. For example, the original controller for a male runner was re-tuned to work with a female and a child runner by applying an optimisation process to the tunable parameters of the controller [40]. The original controller was first adjusted according to the geometric and mass scaling between the models. Then, a simulated annealing algorithm was used to tune controller parameters in the female and child runners according to a mysterious evaluation function, of which few details are given.

Another attractive feature of this approach is that controllers can be parameterised. For example, the desired running speed can be adjusted between certain limits. Running speed

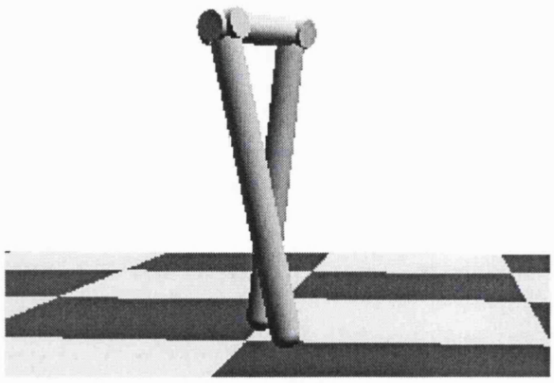


Figure 2.6: configuration of stilts type walker

becomes a parameter of the controller.

The approach of Hodgins and her associates can work well when there is sufficient insight into the design of an appropriate controller, as there is for running. However the controllers that result are very specific, and do not necessarily provide any assistance in the construction of controllers for less well understood behaviours. Even when sufficient insight is available, much effort is required to design, test and tune a controller for each new behaviour.

Constructing a controller for a dynamic biped walking machine is considered to be a more difficult problem (or a less well understood one) than constructing a controller for the more 'ballistic' motions [38], such as hopping, running, etc., tackled by Hodgins and her associates. Nevertheless, researchers have had some success building dynamic biped walking controllers. Some of the efforts are described next.

2.4.3 A Stilts Type Walking Robot

Probably the first successful attempt to build a dynamic biped walking machine was the stilts type walker of Shimoyama [86]. Although this robot had feet and ankles, the ankles were not powered and were simply there to provide a means of measuring the angle of each leg to the ground. The feet therefore acted like point feet and provided no balance control. Just as a person walking on stilts must keep stepping, so this robot had to keep stepping to maintain balance. Figure 2.6 illustrates the configuration of this robot.

Like the hopping robot the stilts type walking robot focuses attention on the principles of dynamic balance by eliminating the mechanisms of static balance.

The controller was designed by decomposing the walking motion into two distinct aspects: side to side rocking and forward motion, for simplicity these were assumed to be independent. Rocking is necessary to provide a phase of single leg support, long enough to swing the free leg forwards. A rather crude rocking motion was induced by designing a sinusoidal-type of trajectory for the hips and stabilising the rocking motion about that trajectory. Stable forward motion was achieved by controlling how far the free leg swings during the single support phase, and hence the stride length. This was found to be a successful strategy for stilts type walking.

It is clear that the motion of this stilts type robot will not appear similar to a 'normal' human walking motion. That would require a more human-like robot. However, the principles which guided the design of its controller should apply also to the control of a more human-like robot.

2.4.4 Walking Robots at MIT Leg Lab

Some of the most promising research on dynamic biped walking has taken place in recent years at the MIT leg lab, which was founded by Marc Raibert whilst working on the hopping robot (see section 2.4.1). Pratt and his associates explored the problem by first building a two dimensional walking robot (attached to boom) called Spring Flamingo [72]. Like a flamingo, this robot has knees that bend backwards, giving it a gait which is actually reminiscent of a real bird. Subsequently they have built a three dimensional walking robot with humanoid legs (but no arms) called M2. A controller has been built for M2 that enables the robot to walk stably in simulation, but, to the best of our knowledge, has not yet been applied successfully to the real robot.

Like Raibert, Pratt and his associate have tackled the problem by searching for a set of simple control rules that produce the desired behaviour, in this case walking [72, 75, 74]. They claim that six things are required for stable three dimensional walking:

1. Height Stabilisation
2. Pitch Stabilisation
3. Speed Stabilisation
4. Lateral Stabilisation
5. Swing Leg Placement
6. Support Transitions

Each of these is treated as essentially a separate problem, and a number of strategies were evaluated to address each one. Perhaps the most important and most general of these strategies is *virtual model control* [73, 71]. The idea of is to solve the control problem by inventing simple components attached to strategic points on the model. For example, to stabilise height we might invent a PD controller attached to the main body, holding it up. Of course this component does not physically exist, but it is possible to create the effect of such a component using the existing actuators. This can be achieved using the same kinematic map that is employed in inverse kinematics (see section 2.1.1). The kinematic map, J^{kin}, relates a generalised force vector at an end effector, F, to a vector of joint torques, τ, according to

$$F = J^{kin}\tau.$$

By using this relationship, it is possible to calculate the forces required at each real actuator to simulate the behaviour of the virtual actuator, provided that the behaviour of the virtual actuator is physically achievable.

Pratt et al try to exploit the natural dynamics of their robots in order to make control easier and produce motion that is smooth and natural looking [74, 75]. For example, the swing leg is, mostly, allowed to swing freely once it starts. The only control that is used is a little torque at the hip to speed the swing for faster walking, and some damping at the knee.

Pratt et al claim that, compared to trajectory following techniques, e.g. Shimoyama's stilt walker (section 2.4.3), their approach produces a walking machine that is more flexible and tolerant of variations in the environment, such as an uneven surface.

2.4.5 Limit Cycle Control

Yet another interesting approach to dynamic, three dimensional walking comes from the animation literature. Laszlo et al added feedback to van de Panne's virtual wind-up toys (described in section 2.3) to stabilise a walking motion about a limit cycle [48]. The virtual wind-up toys technique creates "mostly feed-forward" controllers based upon finite state machines that use proportional derivative servos. There is no overall balance control, so that when such a controller is applied to a humanoid figure it will take only a few steps before falling over.

To keep the walk stable a small number of regulation variables are chosen, in this case hip pitch and roll. The trajectory of these variables in state space is periodic and forms a limit cycle that is stable for a few steps. The control technique involves calculating the control perturbation required to return the system to its stable limit cycle. Numerical differentiation is used to construct a linear approximation that relates a control perturbation to its effect upon the model's state.

The quality of the motion is limited by the quality of the original "mostly feed-forward" controller, but the technique is generally applicable to any unstable periodic motion. The computational cost of the numerical differentiation scheme is too high for the method to be suitable for online simulation.

2.4.6 Dynamic Programming

Van de Panne has also explored the use of dynamic programming to generate controllers for animation purposes [101, 100]. The method, which he calls *approximating graph dynamic programming*, works by discretising the phase space of the system using a regular k-dimensional lattice. One of the lattice points is designated as a target, towards which the controller tries to drive the system. After running an iterative solution procedure based upon the dynamic programming principle, approximately optimal control values are obtained at the nodes of the lattice. These can be interpolated to provide approximately optimal control input to a dynamic model.

Examples presented using this strategy include a hopping desk lamp and a solution to the problem of parking a car in a tight parking space.

The advantages of this method include the fact that it has dynamic correctness *and* whole trajectory optimality built in. It is suitable for online motion synthesis (the controller is generated offline, but may be applied online). It is a very general solution that contains no assumptions about the behaviours that it creates. Also, it could provide a relatively well automated solution to the motion synthesis problem.

The principal drawback of this strategy is its complexity. The storage and time required to solve the dynamic programming problem increases exponentially with the dimensions of phase space. Van de Panne does not present any examples with a phase space which has more than five dimensions.

Van de Panne's approximating graph dynamic programming is described in much more detail in the next chapter because it has more in common with our own approach than any other method in the animation literature.

2.5 Properties of the Ideal Solution

Having reviewed the field of motion synthesis we find that each different approach has some desirable attributes. Unfortunately, each approach also lacks some important attributes that are present in a different approach. There is no single approach that is obviously superior to the others.

In order to explain why we have chosen one approach over the others, it is helpful to list the attributes that we are seeking. We believe that an ideal solution to the motion synthesis problem would be:

Dynamically Correct Motion obeys Newtons laws and the limitations of the system and its actuators (muscles), such as strength, joint limits, comfort and frequency limits.

Optimal Animals generally move in an efficient manner. A good solution to the motion synthesis problem would exhibit a similar efficiency.

Parameterised Attributes such as walking speed, may be adjusted.

Adaptable The system can cope with variations in the environment, such as a stiff breeze, a slippery surface or a steep gradient.

Interactive The system is suitable for online motion synthesis and responds appropriately to unpredictable inputs from users or the environment.

General The prior knowledge about specific behaviours embedded in the system should be minimal. It should be able to cope with a wide range of behaviours and individuals.

Lifelike Motion creates the illusion of life, or is quantitatively similar to real motion, depending upon the application.

Automatic The system requires minimal end user input or intervention.

Unfortunately, no motion synthesis system known to us even comes close to satisfying all these requirements. In practice there is often a trade off between these attributes. For example, motion synthesis systems that are interactive are usually not very lifelike because it is easier to achieve lifelike motion when each sequence can be refined offline. We are therefore forced to prioritise these attributes and to choose an approach that we believe may provide the best fit to our priorities.

In chapter 1 we explained why we expect that lifelike motion should arise from the combination of dynamic correctness and optimality. We have, therefore, chosen to build dynamic correctness and optimality into our approach at the beginning. We also place high emphasis on generality and automation, as we believe these are essential for any system to be widely used.

The direct trajectory optimisation methods (e.g. spacetime constraints) are therefore attractive because they embody the attributes of dynamic correctness, optimality, generality and automation. It has also been shown that such methods can produce lifelike motion. Finally, such methods are well suited to animation because they are allowed to 'stretch' Newton's laws of motion as far as may be required in order to fulfil the requirements (boundary conditions) of any particular motion sequence. However, such methods require optimisation to be performed in very high dimensional parameter spaces, and consequently they can be unreliable when the initial trajectory (which may be a guess) is not sufficiently close to the optimal trajectory. The optimisation may, for example, get stuck in a local optimum. Also, it is not clear that trajectory optimisation problems, for complex dynamic models such as humanoids, can be solved with sufficient speed and reliability to make them suitable for online motion synthesis.

The dynamic programming approach combines the advantages of dynamic correctness, optimality, generality and automation with the ability to deploy controllers generated via this method in an interactive system. Furthermore, the work on control strategies for legged robots, such as that described in section 2.4.4, provides many examples of how the control of a complex legged robot can be decomposed into a set of sub-problems that may be tackled by a dynamic programming approach. These are our reasons for choosing to pursue an approach based upon dynamic programming.

Chapter 3

The HJB Method

The 'HJB method' of the title is a numerical method for solving the Hamilton-Jacobi-Bellman (HJB) equation. The purpose of the method, in this context, is to generate an optimal feedback controller and hence solve an animation problem through the combination of dynamics and optimal control. This chapter sets out to explain both the equation and the method, their origins, application and mathematical basis.

The chapter begins with a discussion of dynamic programing since that is the foundation and historical origin of the method. Two example problems are given: one that has a discrete structure and one that is continuous. The first can be solved by a straightforward application of dynamic programming, while the second is more involved. The examples are used to illustrate the next section, which describes in detail a previous method, Approximating Graph Dynamic Programming (AGDP), that applies dynamic programming to animation. A discussion of the merits and drawbacks of AGDP is given. The HJB equation is introduced along with the theoretical basis for its solution. A discussion of the theoretical and practical problems of solving the HJB equation concludes the chapter.

3.1 Dynamic Programming

Our goal is to find the optimal control, at each instant, to apply to a passive dynamic model. In this way we hope to create natural looking animation. Recall that we are interested in trajectories that are optimal as a whole (section 2.2) and hence our objective function is a function of the whole trajectory. In some cases it may be possible to find the optimal trajectory analytically. However, animal motion is generally too complex to permit analytic solutions and we must rely upon numerical methods. One way to obtain optimal trajectories numerically is to use trajectory optimisation methods (section 2.2). If the control input is a vector with m dimensions and the trajectory is parameterised with n stages then the indirect trajectory optimisation problem has nm dimensions. Even for modest m and n this can be a prohibitively large number of dimensions. It is often difficult to find a global optimum in high dimensional optimisation problems, such as these, and a good initial guess may be required in order to obtain a useful solution. The computation required to generate each unique trajectory is substantial and does not lend itself easily to situations where the control must be calculated on-line.

An alternative to trajectory optimisation comes from the fact that an optimal trajectory has a special property. One may consider any point along an optimal trajectory as the starting point, and the remaining trajectory will still be optimal. This deceptively simple statement is an expression of *Bellman's principle of optimality*, sometimes called the dynamic programming principle. Using this principle it is possible to build up a solution in stages, thereby swapping an nm dimensional problem for a series of m dimensional problems. Of course there may be many difficulties to overcome before effectively making that swap. The various methods of solving problems by employing Bellman's principle are known collectively as *dynamic programming*.

The discipline of dynamic programming was pioneered in the 1950's and 60's by Richard Bellman, who also coined the term[1]. He described it as a conceptual framework for the treatment of many novel and interesting problems in the mathematical theory of *multi-stage decision processes* [11]. The animation problems we are concerned with are but one example of the problems that can be addressed by means of dynamic programming. In general, any process which can be controlled, or at least influenced in some way, is a candidate for a dynamic programming solution. An investment program, a robotic arm and a game of cards are all examples.

In each case, the system evolves over time, either in a continuous fashion (continuous time), as the robotic arm does, or in a discrete fashion (discrete time), as in the card game example where the state of the game only changes at each player's turn. In practice it is not

[1]The following quote comes from Richard Bellman's autobiography [12].

> An interesting question is, "Where did the name, dynamic programming, come from?" The 1950's were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place, I was interested in planning, in decision-making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word which has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has an interesting property as an adjective, and that is it's impossible to use the word, dynamic, in the pejorative sense. Try thinking of some combination which will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

necessary for the system to evolve in any real sense of time, so long as the problem may be cast as a sequence of decision stages. Now, at each stage we may choose an action, a, that affects the evolution of the process, such as which card to play, or how much to invest. Since these examples have many stages, and we must decide upon an action at each stage, they are examples of multi-stage decision processes.

The system may be described at any instant by a vector, x, which is called the state vector. For the robotic arm the state vector might be the set of joint angles and their associated velocities. In the investment program example, however, the state vector might contain components that are not simple measurements, but instead represent probabilistic quantities, such as the likely return on a given investment.

The evolution of the system may be deterministic or stochastic in nature. The robotic arm is an example of a deterministic system, whilst the card game is stochastic since it depends upon the random shuffling of the cards. On initial inspection, these two categories of dynamic programming problem seem quite distinct. However, they turn out to have very similar structure in their solutions. Fortunately, the systems we are concerned with here are of the simpler, deterministic type. Hence we may neglect the stochastic case, except to note its existence and close relationship to the deterministic case.

Associated with each system is a cost function, $l(x, a)$, which may depend upon the state of the system and/or the chosen action. The cost function represents the rate at which cost accumulates. Of course, it can also represent the rate at which reward accumulates, with the aid of a sign change, as it does in the investment program example. We wish to find the sequence of decisions that minimises the sum or integral of the cost function over the lifetime of the process. The policy that determines such a sequence of decisions is called the *optimal policy.*

The next section provides a formal description of dynamic programming in discrete time. Dynamic programming is conceptually simpler in discrete time than in continuous time. Hence it is sensible to examine the discrete time case first. This will also provide the necessary theory to solve an example problem in section 3.1.2, and to describe a previous attempt at applying dynamic programming to animation in section 3.3. We shall return to the continuous time case in section 3.4.

3.1.1 Discrete Time Dynamic Programming

Bellman states his principle of optimality thus:

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

In the discrete time case this may be formulated as follows. Let $A_n = (a_0, a_1, \ldots, a_{n-1})$ be the sequence of decisions taken in an n step deterministic process and let $(x_0, x_1, \ldots, x_{n-1})$ be the sequence of states. Let the accumulated cost be $J(x_0, A_n) = \sum_{i=0}^{n-1} l(x_i, a_i)$, then the optimal policy is determined by the minimum accumulated cost $v(x_0)$ which is given by

$$v(x_0) = \min_{A_n} J(x_0, A_n) \tag{3.1}$$

Bellman's principle may then be expressed as the following recursive relationship

$$v(x_i) = \min_{a_i} \{l(x_i, a_i) + v(x_{i+1})\} \tag{3.2}$$

Note that this formulation assumes that the evolution of the system is uniquely determined by the current state and decision, i.e. that it obeys an equation of the form

$$x_{i+1} = f(x_i, a_i). \tag{3.3}$$

The systems we shall consider are of this type, and are said to have *state structure.* In essence this means that 'history doesn't matter'. In seeking the optimal decision at any instant it does not matter how the system arrived at its current state, it only matters what the current state is now.

Note also that this formulation assumes that the optimal policy does not depend upon the current time. It is said to be *time homogeneous.* Most animal behaviours are time homogeneous, so we shall only consider this case. An example illustrates the distinction between the time homogeneous case and the alternative. Consider the problem of finding the best route to a railway station. If you simply wish to get to the station as quickly as possible, then the current time is irrelevant. You can never get there faster than the quickest route will take you. That is the time homogeneous case. However if you have some time before your train leaves, then the optimal choice may be to get a decent cup of coffee before going to the station.

If we do not have time homogeneity then, in order to preserve state structure, it is necessary for time to become a component of the state vector. The time homogeneous case is therefore computationally cheaper because it has fewer dimensions in the state vector.

State structure means that $v(x_i)$, which is called the *value function,* is uniquely a function of state. This is important because, given the value function and the current state, it is only necessary to compare the outcomes of the decisions available now in order to find the optimal one. The value function tells us everything we need to know about the relative merits of each available decision—without needing to look any further forward in time than the next step.

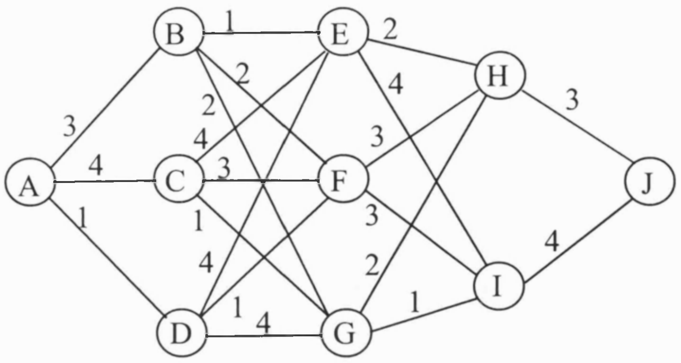


Figure 3.1: the stagecoach problem

Consequently, the optimal decision is also uniquely determined by the current state, and control is in feedback form. Hence, the aim of any dynamic programming method is to determine the value function. Once that is done the problem is effectively solved.

3.1.2 The Stagecoach Problem

The stagecoach problem has discrete structure in both time and space. It is the simplest kind of problem that can be solved by means of dynamic programming. The idea is to find the shortest route between two towns. In figure 3.1 each town is represented by one of the circles A-J. Town A is the starting point and J the destination. The length of the road between each neighbouring town is shown next to the adjoining line. We also require that all roads are one way so that it is only possible to travel from left to right in the diagram.

We solve this problem by calculating the minimum distance required to reach J from each town, working backwards from J in stages. In this way the minimum distance from each town to J is calculated before it is used to decide which is the best route to take from a town in the previous stage. Let us denote the current town as x, and the choice of which road to take from x as a. In the language of dynamic programming, the length of each road is its *cost*, $l(x, a)$, the minimum distance from any town to J is the *value*, $v(x)$, of that town and the shortest route is the optimal trajectory. The value of each town obeys the following functional equation

$$v(x) = \min_a \{l(x, a) + v[f(x, a)]\} \tag{3.4}$$

where $f(x, a)$ gives the town reached from town x, after taking the road corresponding to decision a. Equation 3.4 is the dynamic programming equation for this problem.

Figure 3.2 shows the value of each town. The optimal trajectory starting from A is therefore A-D-F-H-J. On the optimal trajectory the cost of each stage is equal to the associated change in value. In fact the trajectory can only be optimal if this condition is met, as may be seen from equation 3.4.

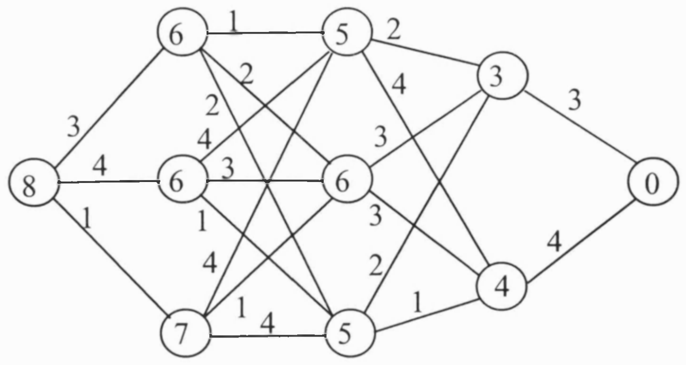


Figure 3.2: solution of the stagecoach problem

Dynamic programming is not necessarily the most computationally efficient solution to this problem. Priority first search [85] might be more efficient because it could find the solution without necessarily visiting every town. However, dynamic programming has the interesting property that it enables us to find the optimal trajectory starting from any town in the problem definition. We have found the solution to a specific problem (the best route from A to J), by solving a more general problem (the best route from any town to J). What is more, the solution to the general problem has been gained at dramatically lower cost than would be required to solve each specific problem separately ($O(n)$ instead of $O(n^2)$ where n is the number of towns).

3.2 The Bush Problem

The Bush problem is another example of a problem that can be solved by dynamic programming methods. However, since it has a continuous structure in both space and time, its solution is less straightforward and less obvious than the stagecoach problem. Here the problem is introduced without describing how it can be solved numerically. Section 3.3 describes one method that may be used to solve it, based upon an understanding of discrete time dynamic programming. In section 3.4.2 a second solution method is described, based upon a more rigorous treatment of dynamic programming in continuous time.

Consider a particle in 1 dimension, governed by Newton's laws of motion. The aim is to find a controller that will bring the particle to rest at the origin in the minimum time. Let us call the position of the particle p. The state of the system may then be described by the two dimensional vector, $x = (p, \dot{p})$, containing both the position and velocity. The space of such vectors, containing both the position and velocity of a system, is called a *phase space*. For simplicity, let us say that the mass of the particle is 1, there is no friction and the control is bounded in the range $a \in [-1, 1]$. By Newton's second law the following equation of motion

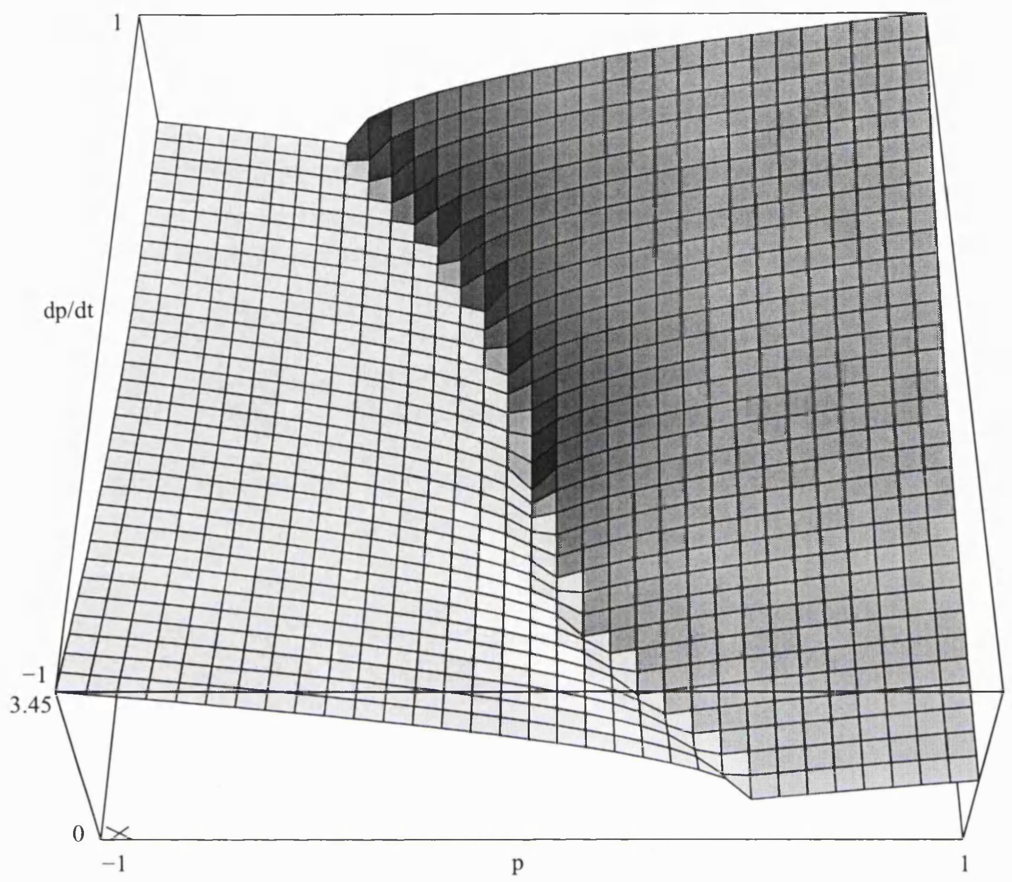


Figure 3.3: analytic value function of the Bush problem

applies

$$\dot{x} = \begin{pmatrix} \dot{p} \\ a \end{pmatrix} \tag{3.5}$$

This is the Bush problem. It is sometimes described in control textbooks [108] because it defeats the classical calculus of variations approach to optimal control. The reason is that the solution uses only the extremes of the control space: full acceleration or full deceleration. This is called 'bang-bang' control and is a common feature of optimal control solutions.

The Bush problem is also a useful benchmark because it is has a known analytic solution and, being two dimensional, is easy to visualise. Figure 3.3 shows its value function, the minimum time to reach the origin, computed analytically and displayed on a regular mesh. The interesting feature here is the S-shaped 'crease' running from top to bottom in the figure. This is the 'switching curve', the set of points at which the optimal control switches from -1 to +1. All optimal trajectories approach the origin, which is in the middle of the figure, via the switching curve. Hence the quality of a numerical solution depends upon locating the switching curve

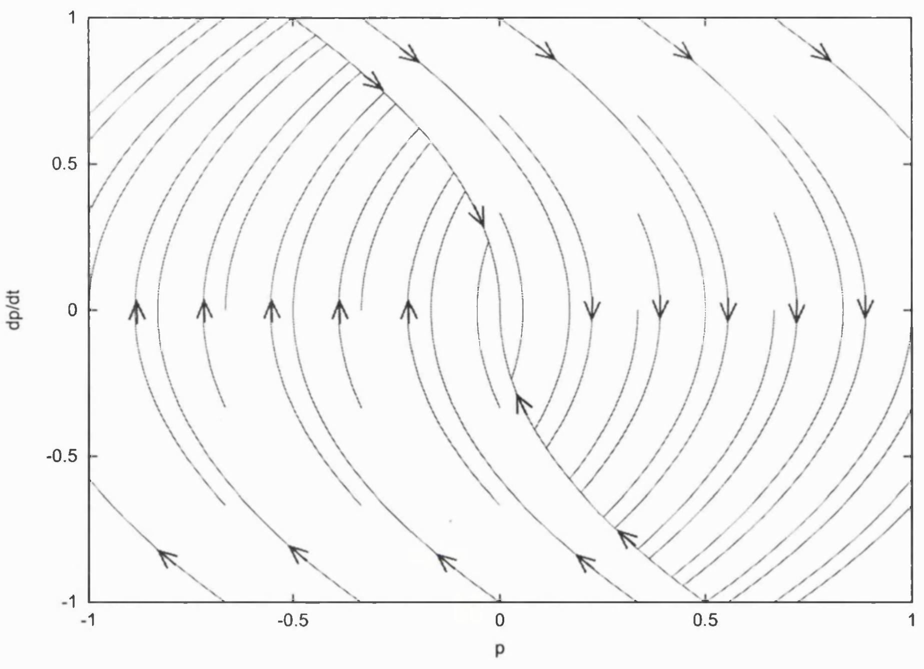


Figure 3.4: optimal trajectories for the Bush problem

accurately. Figure 3.4 illustrates the optimal trajectories from an evenly spaced set of starting points. Starting from any of the curve end-points in the diagram, the optimal trajectory follows the curve up to the switching curve, where it abruptly changes direction and follows the switching curve to the origin.

Analytically, the value function of the bush problem is obtained from the solution of a quadratic equation. If we define the following

$$\begin{aligned} r &= 2\dot{p}^2 - 4p \\ s &= \tfrac{-2\dot{p}+\sqrt{r}}{2} \end{aligned} \tag{3.6}$$

then the roots of that quadratic are real when $r \geq 0$ and non-negative when $s \geq 0$. When both these conditions are met, the point x is on, or to the left of the switching curve. Hence by the rotational symmetry of the function, the minimum time to go, $v(x)$, is given by

$$v(x) = \begin{cases} s + |p + s| & r \geq 0, s \geq 0 \\ v(-x) & \text{otherwise} \end{cases} \tag{3.7}$$

In general, such analytic solutions are rare and we must rely upon numerical methods. The next section describes one numerical approach.

3.3 Approximating Graph Dynamic Programming

There appears to have been only one previous attempt to apply dynamic programming to the control of physically realistic animation. Michiel van de Panne describes his method, which he calls approximating graph dynamic programming (AGDP), in [99, 100, 101]. AGDP is similar to the solution of the stagecoach problem, but with some additional 'wrinkles' introduced because of the continuous structure of the underlying problem.

The purpose of AGDP is to generate controllers for some simple animated 'creatures'. Van de Panne presents results for a desk lamp [101], similar to the Luxo Jr. character [67], and for a simplified human walking model [100]. In the stagecoach problem, the state of the system was a single, discrete value: the current town. For van de Panne's creatures the state of the system is a continuous valued vector. These creatures obey Newton's laws of motion, so both position and velocity information are required to fully describe their state.

The AGDP method constructs a regular lattice, or *mesh*, in the phase space of the system. The mesh is defined by a set of equally spaced subdivisions along each axis of the phase space. The whole mesh is therefore rectangular and composed of many small rectangular regions, or *cells*. The region of phase space that the mesh occupies is called the *domain* of the computation.

The vertices of the mesh, or *nodes*, correspond, loosely, to the towns in the stagecoach problem. One or more nodes must be chosen as the 'destination', just as there must be a destination town in the stagecoach problem. We call this 'destination' the *target set*. Decisions must be made about which control inputs to apply to the system. Usually, the control input corresponds to one or more torques which are applied to the joints of the model, to approximate the muscular action of the creature. Obviously, the trajectory of the system through phase space depends upon the control input.

The solution proceeds by finding the set of available transitions between the nodes of the mesh. Van de Panne achieves this by also discretising the control space, into a relatively small number of possible control inputs. For each node and for each possible control input, a trajectory is generated by integrating the equations of motion forward in time, until the trajectory crosses one of the boundaries of the current cell. A cost is associated with each of these mini-trajectories, or *arcs* as van de Panne calls them, as a function of duration and the control input. Of course, the arcs do not terminate exactly at another node, so the nearest node is treated as the logical end point of the arc, thus enabling the construction of a directed graph from the set of nodes and arcs. Figure 3.5 illustrates the situation for the Bush problem, with only the 'best' arcs shown. The value of each node can now be defined as the minimum cost required to reach the target set, from any node in the domain, via the set of available arcs. Given this graph, and the cost of each arc, the dynamic programming principle can be applied, just as in the stagecoach problem.

There are, however, a few significant differences. Firstly, because each arc does not terminate at a node, it is best to find an approximate value for the actual end-point of the arc, rather than simply using the value of the node nearest to its end-point. In other words, the notion of value is extended to all points within the domain, not just the nodes. In AGDP, the value function at an end-point of an arc is computed from the value of its closest vertex and the gradient of the value function at that vertex, $Dv(x)$, which is found from a finite difference approximation. Formally, the value function is approximated by

$$v(x) = v(x_c) + Dv(x_c) \cdot (x - x_c) \tag{3.8}$$

where x_c is the position of the closest vertex. The components of the approximate gradient vector are given by

$$(Dv(x))_d = \frac{v(x \pm \delta^d \cdot s) - v(x)}{s_d}$$

where δ^d is the Kronecker delta, s is the vector of inter-node spacing, $(Dv(x))_d$ denotes component d of the gradient vector and the sense of the $\pm$ sign depends upon which side of the node at x the end-point of the arc lies.

The dynamic programming equation then becomes

$$v(x_i) = \min_{a \in A} \{l_{i,a} + v(y_{i,a})\} \tag{3.9}$$

where x_i denotes the coordinates of node i, $l_{i,a}$ is the cost of the arc associated with node i and control a, and $y_{i,a}$ is the end point of that arc and, as before, A is the set of possible control inputs.

The second significant difference is required because of cyclic dependencies in the approximating graph. The value function approximation scheme means that each node depends upon several nodes for the finite difference gradient approximation. Cyclic dependencies can be seen in figure 3.5 where the optimal arcs circle around the origin. For example, node **c** depends upon nodes **d**, **e**, **f**, **g**, **h** and ultimately itself. In this situation, there is no correct order of evaluation. The solution is to use a relaxation process, i.e. to iteratively re-evaluate the nodes until they do not change any more.

The third and final significant difference arises because there does not necessarily exist a connected path from every node in the mesh to the target set. This can be seen in figure 3.5; no arc is shown for the node **b**, because *all* trajectories from this point lead out of the domain. This happens because the velocity of the system, at those phase-space coordinates, must carry the system to the right and hence out of the domain. It is not possible to calculate the value of a disconnected node such as this. At first glance this might not seem to be a problem. If no trajectory exists between a node and the target set then such a node can simply be disregarded? Unfortunately, it is not quite that simple. Some of the arcs from connected nodes depend upon disconnected nodes for a gradient estimate. For example, node **a** depends upon node **b** for a gradient estimate. Van de Panne's solution is to use an estimate of the maximum gradient throughout the domain when the gradient cannot be calculated from the finite difference approximation. The components of the gradient estimate are thus given by

$$(Dv(x))_d = \begin{cases} \frac{v(x \pm \delta^d \cdot s) - v(x)}{s_d} & v(x \pm \delta^d \cdot s) \text{ known} \\ \Psi_d & \text{otherwise} \end{cases} \tag{3.10}$$

where Ψ_d is the dth component of the maximum gradient estimate. AGDP is implemented in such a way that the value of a node can never go up, it only goes down. In the relaxation process, the value of a node is only updated if it is lower than the previous value held for that node. Hence any overestimate of the value of a node may be removed later on, but an underestimate can never be removed. In principle, a maximum gradient estimate permits only overestimates.

With these points in mind, we can describe the algorithm with the following pseudo code.

```
for $n \in N$ push $n$ onto $P$
repeat
  repeat
    n = pop $P$
    $u = v(x_n) \min_{a \in A} \{l_{n,a} + v(y_{n,a})\}$
    if ($u < n.v$)
      $n.v = u$
      add dependents($n$) to $Q$
  until $P$ is empty
  for $n \in Q$ push $n$ onto $P$
  clear $Q$
until $P$ is empty
```

Algorithm 3.1: approximating graph dynamic programming

In this algorithm N is the set of all nodes, $n.v$ denotes the value of node n, P is a priority queue of nodes ordered with lowest value nodes first, Q is an unordered set of nodes and dependents(n) is a function that returns the set of nodes dependent upon node n.

Figure 3.5 illustrates the application of AGDP to the Bush problem on a 5x5 mesh of the rectangular domain. Only the optimal arc from each node is shown.

3.3.1 Reconstruction of Control Function

Once the value function has been generated, a relatively straightforward method is employed to reconstruct the control inputs from the solution. The optimal control at each node is directly

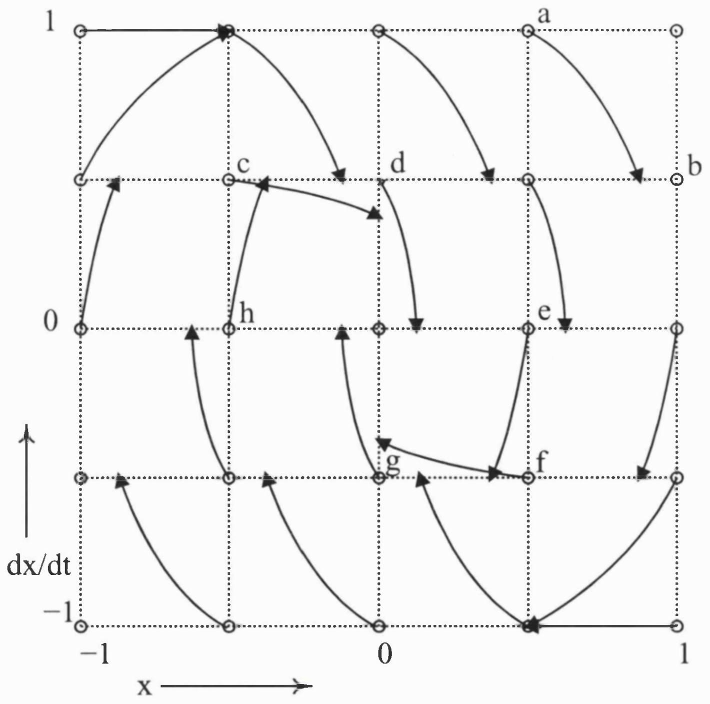


Figure 3.5: example AGDP solution of the Bush problem

available, as the one corresponding to the best arc from that node. Multilinear interpolation is employed to calculate the control, a_x, at any point x in-between the nodes. This can be expressed as

$$a_x = \sum_{v=1}^{2^k} w_v a_v, \tag{3.11}$$

where the summation is performed over the 2^k vertices of the k dimensional cell containing x, a_v is the control associated with vertex v and w_v is the weight associated with vertex v. The vertex weight factor, w_v, is obtained from

$$w_v = \prod_{i=1}^{k} \left(1 - \frac{|x_i - v_i|}{s_i}\right), \tag{3.12}$$

where the multiplication is performed over the k dimensions of the space, s is the size of the cell and v_i denotes the i'th coordinate of vertex v.

This scheme ensures that the vertex weight factors always sum to one, i.e

$$\sum_{v=1}^{2^k} w_v = 1, \tag{3.13}$$

and that each vertex weight factor satisfies $w_v \in [0, 1]$. In other words, multilinear interpolation acts as a *convex combination* of the vertices of one cell.

3.3.2 Discussion

The strength of AGDP lies in the fact that it produces optimal feedback controllers. The controller can produce a whole family of trajectories, each of which is an approximation to the globally optimal trajectory from some point in phase space to the target set. It is exceedingly difficult to offer any such guarantee when using any method other than one based upon dynamic programming. However, it should be noted that this advantage is not specific to AGDP, but a property of the dynamic programming approach to optimal control.

The primary drawback of AGDP is that the number of nodes required in the solution rises exponentially with the number of dimensions in the phase space. For example, if each dimension of the phase space is discretised with 10 node points, then in four dimensions 10^4 nodes are required, and in six dimensions 10^6 nodes are required, and so on. The exponential dependence of algorithmic cost upon dimensionality is often called the *curse of dimensionality*. In practice this severely limits the dimensionality of problems that can be tackled with this approach. Van de Panne presents no examples with more than a five dimensional phase space. This problem bedevils all dynamic programming solutions of optimal control problems and is not specific to van de Panne's AGDP method.

The method for interpolating the value function, when examined closely, is rather peculiar. It results in discontinuities in the value function where there should be none. This can be seen by considering the 2D case where the four nodes surrounding a cell are not co-planar. However, it is more accurate than simply using the value function at the nearest node.

The usefulness of the dependency graph is questionable. The value of a node depends not just upon the node nearest its optimal arc end-point, but also upon the nodes that are used to determine the gradient of the value function at that node. In the general 2 dimensional case each node will depend upon 3 other nodes. Some of those nodes may have a higher value than the dependent node. Combined with the presence of cyclic dependencies it is not hard to see that every node becomes dependent upon almost every other node.

Another problem is the fact that the value of a node is only allowed to decrease during the iterative procedure. If there are any circumstances in which an under-estimate of the value function may be obtained, then those under-estimates remain as errors in the solution. One potential source of under-estimates is the scheme for interpolating the value function. We shall discuss a similar iterative scheme later that allows the value function to move in either direction. This latter scheme is more robust, since it is not necessary to ensure that under-estimates never occur.

The use of a maximum gradient estimate also poses some problems. The gradient of the value function may in some places be very high and may even be discontinuous, with an infinite gradient. The Bush problem provides an example of a discontinuous value function. From some points within the domain, such as point **b** in figure 3.5, all possible trajectories lead out of the domain. We say these points are outside the *reachable set* since the target set cannot be reached from these points. There need be no continuity in the value function at the boundary of the reachable set because no possible trajectory exists to connect points outside the reachable set to those inside. In such cases the very notion of a maximum gradient estimate is flawed.

Finally, AGDP provides no method for deciding how fine the mesh needs to be. This judgement can only be made from educated guesswork, or from trial and error. In the four dimensional case, simply doubling the number of subdivisions along each axis results in a 16-fold increase in the complexity of the solution. This is inconvenient, but not in most cases not a computational impossibility. However, in k dimensions, the number of nodes increases by 2^k, with the result that such a simple method of mesh selection does not scale to high dimensional problems. A better mesh selection procedure would allow for the heterogeneous nature of phase space coordinates, which may have very different physical significance.

3.4 The Hamilton-Jacobi-Bellman Equation

We turn now to dynamic programming in the continuous time case. As we shall see, an understanding of the continuous time case leads to a more rigorous approach to the solution of optimal control problems, such as the Bush problem and the animation problems we are interested in.

We shall assume that any continuous dynamic system of interest can be cast in the following canonical form:

$$\begin{cases} \dot{y}_x(t) = f(y_x(t), \alpha(t)) \\ \quad y(0) = x \end{cases} \tag{3.14}$$

where $\alpha(t)$ is the *open loop control* of the system chosen from the set of permissible controls, A. Whereas previously we used x in the discrete equivalent of this equation, equation 3.3, we now choose to use y in order to make clear the distinction between a trajectory as a function of time, i.e. $y(t)$, and the value function as function of state space, i.e. $v(x)$.

Recall that the aim is to optimise whole trajectories. We therefore need to define a *cost functional*, $J(x, \alpha)$, as the integral of a cost function, $l(x, a)$, along a trajectory. We choose to use the *infinite horizon* formulation for our cost functional [10], i.e.

$$J(x, \alpha) := \int_0^\infty l(y_x(t, \alpha), \alpha(t))e^{-\lambda t}dt, \qquad \lambda > 0. \tag{3.15}$$

Notice that the integral is taken over an infinite period of time, the infinite horizon, and that a factor $e^{-\lambda t}$ has been introduced. λ is called the *discount factor;* its effect is to exponentially reduce, i.e. discount, the instantaneous cost with increasing t. If the cost function is bounded then the introduction of $e^{-\lambda t}$ ensures that J remains finite.

This formulation is different from the discrete time formulation used previously, where the process was considered to have ended when the target set was reached. Now we treat the process as if it continues indefinitely. This choice is convenient because it allows us to dispense with the target set altogether; an important advantage for many behaviours where the notion of a target set is artificial.

The value function is defined as the minimum of the cost functional with respect to the open loop control function

$$v(x) := \inf_{\alpha \in A} J(x, a). \tag{3.16}$$

Now, assume that an optimal open loop control, $\alpha^*_x(t)$, exists for each x, so that

$$v(x) = J(x, \alpha^*_x) = \int_0^\infty l(y_x(t, \alpha^*_x), \alpha^*_x(t)) e^{-\lambda t} dt.$$

We may split the integral at any value of $T > 0$ to obtain

$$v(x) = \int_0^T l(y_x(t, \alpha^*_x), \alpha^*_x(t)) e^{-\lambda t} dt + \int_T^\infty l(y_x(t, \alpha^*_x), \alpha^*_x(t)) e^{-\lambda t} dt. \tag{3.17}$$

From equation 3.14 we know that the evolution of our dynamic system depends only upon the current state and the applied control, not upon any previous state. Therefore the second integral in 3.17 is independent of the first and we must have

$$v(y_x(T, \alpha^*_x)) = \int_T^\infty l(y_x(t, \alpha^*_x), \alpha^*_x(T + t)) e^{-\lambda(t-T)} dt, \tag{3.18}$$

and so

$$v(x) = \int_0^T l(y_x(t, \alpha^*_x), \alpha^*_x(t)) e^{-\lambda t} dt + v(y_x(T, \alpha^*_x)) e^{-\lambda T}. \tag{3.19}$$

Therefore, under the assumption that an optimal open loop control exists for each x for $t \in [T, \infty]$, we need only consider optimising the open loop control over the finite period, T, giving the following relationship:

$$v(x) = \inf_{\alpha \in A} \left\{ \int_0^T l\left(y_x(t, \alpha), \alpha(t)\right) e^{-\lambda t} dt + v(y_x(T, \alpha)) e^{-\lambda T} \right\}. \tag{3.20}$$

Equation 3.20 is the dynamic programming equation for the infinite horizon problem.

If we assume that $v(x)$ is differentiable everywhere then we may divide both sides by $T > 0$ and let $T \to 0$ to obtain

$$-\lambda v(x) + \inf_{a \in A} \{f(x,a) \cdot Dv(x) + l(x,a)\} = 0 \tag{3.21}$$

where $Dv(x)$ is the gradient of v at x. Equation 3.21 is the Hamilton-Jacobi-Bellman equation for this problem. A more complete derivation of this HJB equation can be found in Chapter 3 of Bardi et al [10]. Note that if a different cost functional, such as in a finite horizon problem, is chosen then a slightly different HJB equation will be the result. Bardi lists four prototypical problems, infinite horizon, finite horizon, minimum time and discounted minimum time [9]. In this work, when we refer to *the* Hamilton-Jacobi-Bellman equation we are referring to equation 3.21.

The name is derived from its Hamiltonian structure, where the Hamiltonian is $H = \inf_{a \in A} \{f(x,a) \cdot Dv(x) + l(x,a)\}$; from the Jacobian term, $Dv(x)$, and, of course, from the fact that it is derived from Bellman's principle of optimality.

Unfortunately, the assumption that $v(x)$ is differentiable everywhere may not always hold. A simple example illustrates this point.

Consider a one dimensional system, with the following equation of motion, permissible controls and cost function,

$$\begin{aligned} f(x,a) &= a \\ A &= [-1,1] \\ l(x,a) &= \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases} \end{aligned} . \tag{3.22}$$

An optimal trajectory, for this system, is one which controls the system to the origin in the minimum time (this is equivalent to specifying a target set at the origin.) It is easy to see that the optimal control is given by

$$a = \operatorname{sgn}(x) \tag{3.23}$$

and, if we neglect the discount factor, that the value function is

$$v(x) = |x| \tag{3.24}$$

which is not differentiable at the origin.

3.4.1 Viscosity Solutions

In fact it is possible to define the notion of a solution for the Hamilton-Jacobi-Bellman equation, even when v(x) is not differentiable everywhere. To do so, the mathematical theory of *viscosity solutions* of partial differential equations [10] is used. It furnishes us with the necessary tools to analyse the non-differentiable case. The theory is built upon a generalised notion

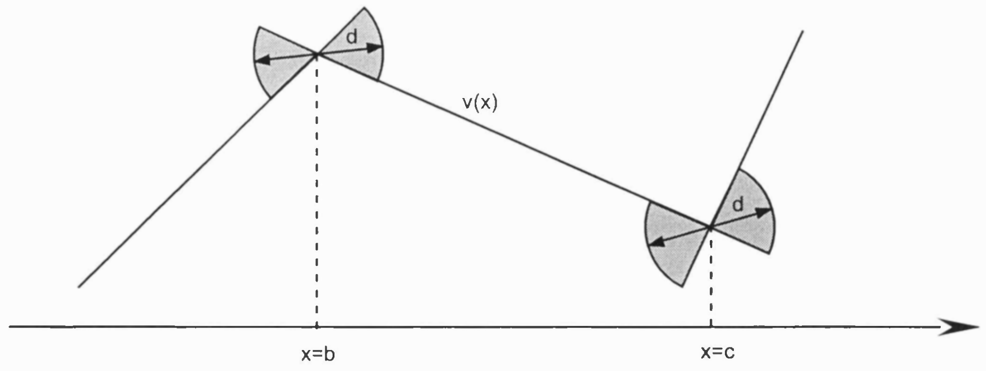


Figure 3.6: geometric interpretation of the superdifferential and subdifferential

The superdifferential of $v(x)$ at $x = b$ is defined by the set of tangents at $x = b$, such that the tangent remains $\geq v(x)$ within a small region of $x = b$. This is shown by the left hand bow-tie shape, formed by drawing each tangent in the superdifferential with length d. Likewise the subdifferential at $x = c$ is defined by the set of tangents at x, such that the tangent remains $\leq v(x)$ within a small region of $x = c$. This is shown by the right hand bow-tie shape formed by drawing each tangent in the subdifferential with length d.

of differentiability, defined by the following sets:

$$D^+(v) = \left\{p \in \mathcal{R}^N : \limsup_{y \to x} \frac{v(y) - v(x) - p \cdot (y - x)}{|y - x|} \leq 0\right\} \quad (3.25)$$

$$D^-(v) = \left\{q \in \mathcal{R}^N : \liminf_{y \to x} \frac{v(y) - v(x) - q \cdot (y - x)}{|y - x|} \geq 0\right\}. \quad (3.26)$$

These are called the *superdifferential* and *subdifferential* respectively. The geometric interpretation of these two is illustrated in figure 3.6. Where the function v is differentiable, both the subdifferential and superdifferential sets are equal to the differential of v at x, i.e. $D^+v(x) = D^-v(x) = \{Dv(x)\}$.

Now we may define the notion of a viscosity solution. Given a partial differential equation of the form

$$F(x, u(x), Du(x)) = 0, \quad (3.27)$$

then a continuous function u is a viscosity solution if both the following conditions are satisfied:

$$F(x, u(x), p) \leq 0 \qquad \forall p \in D^+u(x) \quad (3.28)$$

$$F(x, u(x), q) \geq 0 \qquad \forall q \in D^-u(x). \quad (3.29)$$

If equation 3.28 is satisfied then u is a viscosity subsolution of 3.27, and if equation 3.29 is satisfied then u is a viscosity supersolution.

In the case of the HJB equation the subsolution and supersolution conditions become

$$\begin{aligned} -\lambda v(x) + \inf_{a\in A}\{f(x,a)\cdot p + l(x,a)\} \leq 0 \qquad \forall p \in D^{+}v(x) \\ -\lambda v(x) + \inf_{a\in A}\{f(x,a)\cdot q + l(x,a)\} \geq 0 \qquad \forall q \in D^{-}v(x). \end{aligned} \tag{3.30}$$

It can be shown that if the value function v is continuous, then even if it is not differentiable, it satisfies both these conditions and hence it satisfies the HJB equation in the viscosity sense [10]. Furthermore, the value function is the *unique* viscosity solution of the HJB equation.

However, in some circumstances the value function is not even continuous. The theory of viscosity solutions has been extended to cover the discontinuous case and, under certain assumptions, the value function may still be characterised as the unique viscosity solution of the HJB equation. The reader is referred to [10] for the details.

What, then, are the conditions under which the value function is continuous? If both the plant, $f(x,a)$, and the cost function $l(x,a)$ are bounded and continuous in x then the value function will be continuous also[10]. However the proof of this point relies upon the assumption that all trajectories remain within the domain for all time. For the purposes of numerical solution it is often not possible to meet this condition. Continuity of the value function cannot then be guaranteed.

Whilst the above conditions for the continuity of v are sufficient, they are not necessary conditions. For example, the Bush problem has a cost function that is discontinuous in x (0 at the origin, 1 elsewhere), yet the value function is continuous within a domain of the right shape (the shape of the reachable set). It is more difficult to formulate a necessary and sufficient set of conditions, for the continuity of v, than a set that is merely sufficient. We are not aware of such a set of conditions.

3.4.2 Numerical Solution of the HJB

A numerical solution method for the infinite horizon problem is given by Falcone in Appendix A of [10]. We shall treat Falcone's method as the starting point for our exploration of methods for solving the HJB equation, as it has proven convergence properties and is perhaps the simplest possible method of its type.

We begin by forming a discrete time problem, with constant time step h. The system dynamics are approximated by means of an Euler integration scheme (rectangle rule) such that

$$y_{i+1} = y_i + hf(y_i, a_i) \tag{3.31}$$

and likewise the cost function is approximated by a rectangle rule, such that

$$J_{i+1} = (1 - \lambda h)J_i + hl(y_i, a_i). \tag{3.32}$$

The value function in this case is approximated by

$$v_h(x) = \min_{a\in A}\{(1-\lambda h)v_h(x+hf(x,a)) + hl(x,a)\}, \tag{3.33}$$

It can be shown that for $h \to 0^+$ the discrete approximation tends to the continuous case and $v_h \to v$.

The next step is to make a discretisaton in space. Assume for now that that there exists a bounded region of state space, Ω, such that for h sufficiently small

$$x + hf(x,a) \in \Omega \qquad \forall x \in \Omega\, \forall a \in A \tag{3.34}$$

i.e. Ω contains all possible trajectories for all time. Now construct a mesh in phase space with the nodes numbered $x_1, x_x, \ldots, x_L$ such that the approximating function is given by

$$u(y) = \sum_{j=1}^{L} w_j(y)u(x_j), \tag{3.35}$$

where the coefficients $w_j(y)$ form a convex combination and satisfy

$$0 \le w_j \le 1 \tag{3.36}$$

with

$$\sum_{j=1}^{L} w_j(y) = 1 \qquad \forall y \in \Omega. \tag{3.37}$$

Let us denote the vector formed by the approximating function at each node as $U = (u(x_1), \ldots, u(x_L))$. Now we may define the map $T(U)$, such that the following iterative scheme may be used to solve 3.33,

$$U_{k+1} = T(U_k).$$

Each component of T is thus given by

$$(T(U))_i = \min_{a\in A}\left[(1-\lambda h)u(x_i + hf(x_i,a)) + hl(x_i,a)\right]. \tag{3.38}$$

Finally, it can be shown that [10]

$$\|T(U) - T(V)\|_\infty \le (1-\lambda h)\,\|U - V\|_\infty, \tag{3.39}$$

where $V = (v(x_1), \cdots, v(x_L))$ and $\|\cdot\|_\infty$ denotes the max norm for vectors, i.e. the largest component of the vector.

The significance of 3.39 is that $T(U)$ converges to the solution of equation 3.33 as the number of iterations tends to infinity. The convergence rate is $(1-\lambda h)$, and hence convergence is obtained for $h \in [0, 1/\lambda]$.

The pseudo-code in algorithm 3.2 illustrates the method. Here, γ is the convergence threshold, N is the set of all nodes in the mesh, $n.v$ denotes the stored value of node n and x_n denotes the state space coordinates of node n. The function $u(x)$ returns the value function at an arbitrary point by interpolating the stored value at nodes, or, if the point, x, is outside the domain, by returning a boundary value.

```
repeat
  for $n \in N$
    $u^* = \min_{a \in A}\{(1 - \lambda h)u(x_n + hf(x_n, a)) + hl(x_n, a)\}$
    $\delta_n = |n.v - u^*|$
    $n.v = u^*$
until $\max_{n \in N}(\delta_n) < \gamma$
```

Algorithm 3.2: simplest solution method for the HJB equation

Figure 3.7 illustrates the result of applying the above algorithm to the Bush problem, with a discount factor of 0. This problem does not satisfy the assumption that the computational domain should contain all trajectories for all time. In the top right, and bottom left corners of the diagram the value function reaches an upper bound because no trajectories exist that can return the system to the origin without leaving the domain. They are outside the *reachable set*. Unfortunately, if we compare with the analytic solution in figure 3.3 we see that this effect distorts the solution for a significant region inside the reachable set. This problem will be examined in more detail in the next chapter. Nevertheless a reasonable solution is obtained for a large fraction of the domain.

3.4.3 Discussion

As with van de Panne's AGDP, the fundamental drawback of the HJB method is the exponential increase in computational complexity as the dimension of the state variable, x, rises. In this respect it offers no advantage over AGDP.

The HJB method, however, offers a mathematically rigorous basis for the treatment of continuous time problems within the framework of dynamic programming. The characterisation of the value function as the unique viscosity solution of the Hamilton-Jacobi-Bellman equation permits a degree of mathematical analysis that can inform the design of numerical solution procedures.

The HJB method converges to the value function regardless of the initial value given to the nodes in the mesh. They can start above or below the final solution. The uniqueness property of viscosity solutions of HJB equations means that the numerical procedure should always converge upon a unique solution. This is a very useful property.

As with AGDP, the HJB method offers no means of deciding how coarse or fine the mesh

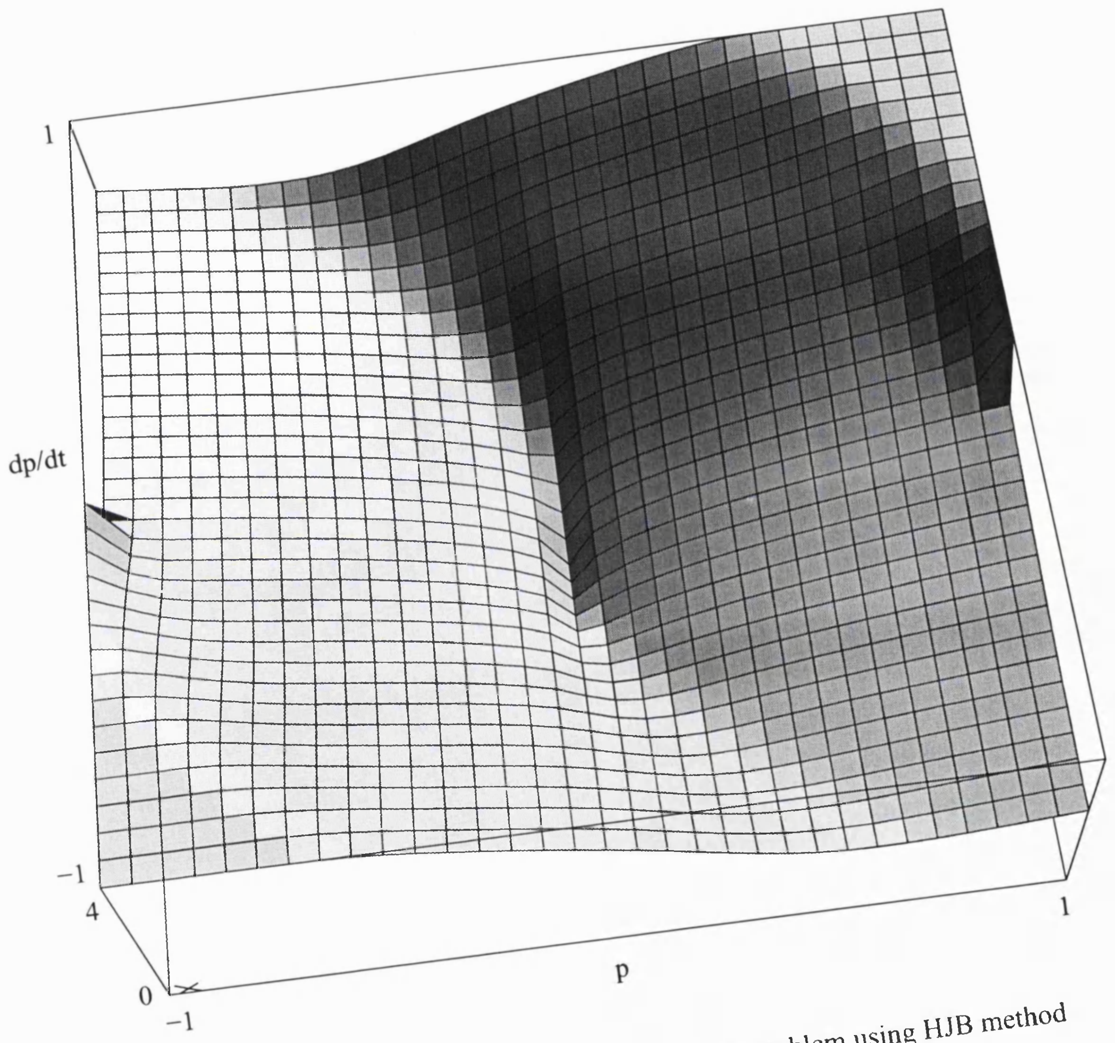


Figure 3.7: Numerical value function of the Bush problem using HJB method

needs to be. In two dimensions this may not seem like a problem but in higher dimensions it becomes more important.

Chapter 4

Convergence, Boundaries & Meshing

This chapter is about the design and construction of a practical algorithm for solving the HJB equation. We shall take the simple algorithm described at the end of the last chapter, algorithm 3.2, as our starting point. This algorithm is inadequate for tackling the animation problems in which we are interested. We shall examine the drawbacks of this algorithm in detail and suggest ways in which it can be improved. There are three areas of concern, and hence three sections in this chapter. The first deals with practical methods for accelerating the convergence of the algorithm. The second deals with the boundaries of the reachable set, and understanding the influence these have on the solution. The third addresses the topic of meshing. We describe a method for constructing efficient meshes adaptively, by refining the mesh only where it is most needed.

4.1 Convergence

Recall from section 3.4.2 that the inequality 3.39 provides $(1-\lambda h)$ as a lower bound on the rate of convergence. From this we might expect to see linear convergence with a rate proportional to the step size, h, and to the discount factor, λ.

It is easy to verify that the convergence rate is indeed proportional to h, for small enough h, by considering the update rule for individual nodes. If we calculate the change in value of a node from equation 3.38 and use the approximation $u(x+hf(x,a))-u(x) \simeq hDu(x)\cdot f(x,a)$ then we obtain

$$\delta_n \simeq h\left(-\lambda u(x_n) + \min_{a\in A}\{Du(x_n)\cdot f(x_n,a) + l(x_n,a)\}\right) \tag{4.1}$$

which demonstrates that the update is linear in h.

Equation 4.1 also shows that the update is linear in λ *provided that* $u(x)$ *is non-zero.* If $u(x)$ is exactly zero at some point within the domain then λ will not affect the rate of convergence at that point. An example of this is seen in the Bush problem where $u(x) = 0$ at the origin because, at this point, the optimal trajectory is stationary and the cost function is zero. In this case all optimal trajectories stop at the origin, hence the cost functional is guaranteed to be finite within a finite domain and convergence is still obtained for $\lambda = 0$.

Given that the convergence rate is linear in h, an obvious way to speed up convergence is to increase h. However, we expect that an increase in h will result in lower accuracy, because the semi-discrete approximation given in equation 3.33 tends to the continuous case as $h \to 0^+$. Apparently, there is a conflict between the need for accuracy (hence small h) and the need for fast convergence (hence large h). This begs the question of how to choose h.

One possible answer is to start with a large value of h, and hence quickly obtain an approximate solution. The accuracy can then be improved by iterating with smaller values of h. Whilst this method works, a little investigation shows that it is not the most efficient way to achieve an accurate solution. The next subsection describes an experiment that demonstrates why.

4.1.1 An Experiment with Convergence Rates in One Dimension

Let us consider the one dimensional problem defined in the last chapter by equation 3.22. This example is useful as it is very simple and serves to illustrate some principles for convergence of the HJB method that we shall later apply to higher dimensional examples.

An experiment was conducted to measure the effect of varying the timestep, h, upon the number of iterations required for convergence and the accuracy of the solution. Figure 4.1 shows the results of that experiment.

The conclusion is obvious. For this 1D example, there is a single optimal timestep, $h = 0.2$. A smaller timestep gives slow convergence, a larger one results in significant numerical error. There is no advantage to be gained by varying the timestep. The convergence rate does not improve with timesteps greater than $h = 0.2$ and the accuracy does not improve with smaller timesteps.

The explanation is straightforward. When the timestep is optimal, and $a = \pm 1$, the system moves a distance $hf(x,a) = 0.2$ in state space. This corresponds exactly to the distance between adjacent nodes. Convergence then hits a 'sweet spot' because the difference in value between adjacent nodes can be determined in one iteration. Of course it still takes a few iterations for changes to propagate across the mesh and hence to determine the optimal control function everywhere.

Accuracy suffers with $h > 0.2$ because a single step can take the system across the bottom of the V shaped value function. This gives an erroneous estimate of the local gradient of the value function and hence the wrong answer. In other words, when the timestep is too large, it

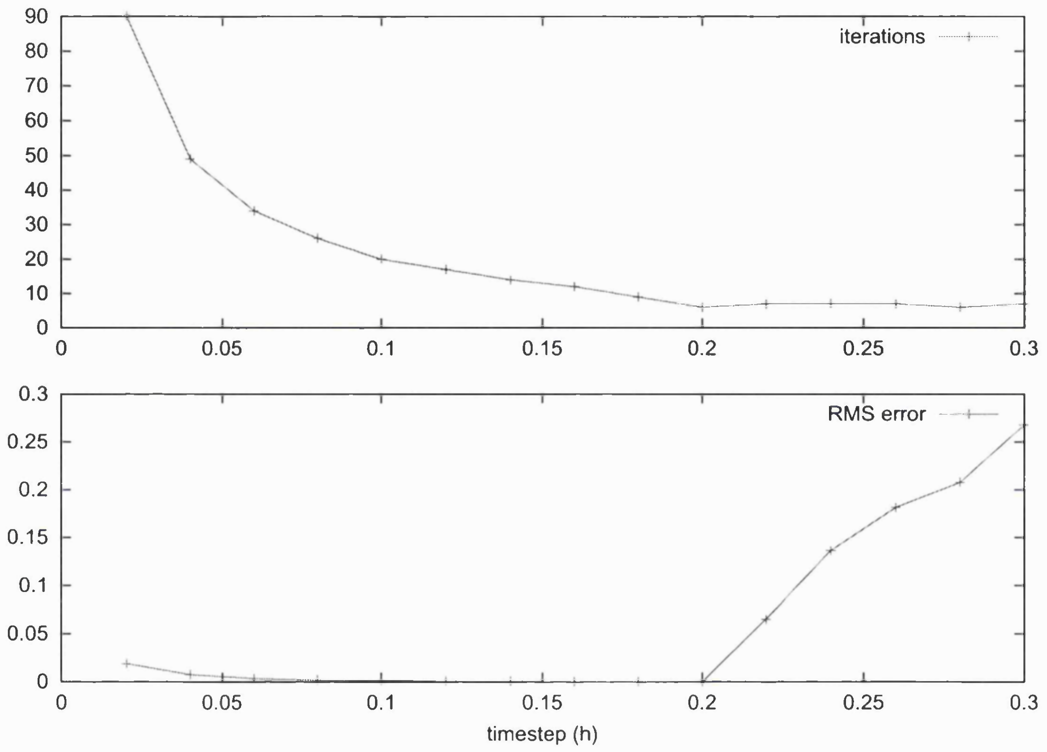


Figure 4.1: effect of timestep upon convergence and accuracy in 1D example

These results were obtained with a computational domain of $[-1, 1]$ discretised with 11 evenly spaced node points. Likewise the control space was discretised with 11 evenly spaced values in the range $[-1, 1]$. Algorithm 3.2 was used. The upper graph shows the number of iterations required for convergence, with the threshold in this case equal to 10^{-3}. The lower graph shows the root-mean-square of the pointwise difference between the numerical solution and the analytic solution.

can 'miss' important features of the solution, such as the bottom of the V. In practice, we have observed similar problems with large timesteps in problems with two and more dimensions.

This result, that there is a single optimal timestep, is very convenient for this restricted and rather special example. The question arises, how does it apply in higher dimensional problems? That question is addressed next.

4.1.2 Multiple Timesteps in Two and More Dimensions

In more than one dimension a single step will not, in general, terminate exactly at a neighbouring node, regardless of the choice of h. Even if it were possible for a step to terminate at a node, that step is not likely to correspond to the optimal control. The nearest equivalent would be to arrange for each step to terminate at the boundary of the region of influence of the starting node. If multilinear interpolation is used (as in the method for interpolating control values in AGDP, section 3.3.1), then that means stepping to the boundary of the cells connected to the starting node. Often, however, the time required to reach the cell boundary is so large that several integration timesteps must be taken in order to avoid significant numerical error.

This approach, of taking multiple timesteps to reach the cell boundary, is the one adopted by van de Panne for AGDP (see figure 3.5) and also by Munos and Moore in their work on applying very similar methods to optimal control problems [62, 58].

Unfortunately, the additional computational cost required for multiple timesteps can be substantial. In general, there is a significant cost associated with evaluating the plant equation, $f(x, a)$, and several evaluations of the plant equation may be required for each available control input at each node. There is also a significant cost associated with determining the time of intersection with the cell boundary. Hence the multiple timestep approach will generally achieve convergence in fewer iterations than would be obtained with a fixed step size, but each iteration will be more costly.

4.1.3 Local Iteration as the Limit of a Sequence

There is a way to retain the speed of a single Euler step whilst achieving a similar convergence rate to the multiple timestep approach. The insight required to understand how this can be achieved comes from the observation that convergence is slow whenever the update rule for a node depends in part upon the previous value of that node.

If w_n is the weight assigned to node n during the interpolation procedure that is used to lookup the value of the end point of a step then convergence is slow when $w_n \neq 0$. To understand why this is so, consider what would happen if node n were updated many times whilst the control input and the value of all other nodes are held constant. This is what we call the *local iteration*. In this case the interpolation procedure may be written as

$$u(y_n) = w_n u(x_n) + k \tag{4.2}$$

where y_n is the end point of the integration timestep taken from x_n and k is a constant given by

$$k = \sum_{j=1, j \neq n}^{L} w_j(y) u(x_j). \tag{4.3}$$

The update rule for a node becomes

$$T(U)_n = (1 - \lambda h)(w_n u(x_n) + k) + l(x, a). \tag{4.4}$$

All the terms in this update rule are constant except $u(x_n)$ hence the update rule is equivalent to the following simplified form

$$u_{i+1} = a u_i + b \tag{4.5}$$

where a and b are constants and u_i denotes $u(x_n)$ at update i. Thus the node values form a series, $u_0, au_0 + b, \ldots$, that should eventually converge upon a finite value. Clearly, if $w_n = 0$ then $a = 0$ and the series will not change after the first update, i.e. $u_1, u_2, \ldots = b$. However, if $w_n \neq 0$ then the series will continue changing. It will converge if $a \in [0, 1)$, which is always true in this case. The rate of convergence is a function of a, which in turn is a function of h, and this fact is sufficient to explain the slow convergence obtained with short timesteps.

To determine the value of a node more quickly we can use the limit of the series (equation 4.5) as the number of updates goes to infinity. It happens that the limit is given by

$$u_\infty = \frac{b}{1 - a} \qquad a \in [0, 1) \tag{4.6}$$

Hence it is possible to calculate the value of a node by expanding 4.6 to give

$$(u(x_n))_\infty = \frac{(1 - \lambda h)k + l(x_n, a)}{1 - (1 - \lambda h) w_n}. \tag{4.7}$$

With equation 4.1 we may now rewrite algorithm 3.2 to obtain the algorithm 4.1.

Notice that if $w_n = 0$ in equation 4.7 then it reduces to the single update rule in equation 3.38. This is true when $h \geq \hat{h}_n$, where $\hat{h}_n$ is the time required to step to the edge of the cell from node n. Consequently, when $h \geq \max_{i=1 \ldots L}\{\hat{h}_i\}$ algorithm 4.1 is identical to algorithm 3.2.

Notice also that the right hand side of equation 4.7 does not contain $u(x_n)$. Therefore the value of a node is obtained independently of its previous value, whether or not $w_n = 0$. We therefore expect that algorithm 4.1 will convergence at a similar rate for any choice of h.

```
function evaluate(node $n$, control $a$)
  $y = x_n + hf(x_n, a)$
  $u = \frac{(1-\lambda h)k(y)+l(x_n,a)}{1-(1-\lambda h)w_n(y)}$
  return $u$

repeat
  for $n \in N$
    $u^* = \min_{a \in A}\{\text{evaluate}(n, a)\}$
    $\delta_n = |n.v - u^*|$
    $n.v = u^*$
until $\max_{n \in N}\{\delta_n\} < \gamma$
```

Algorithm 4.1: limit-of-a-series modification

parameter	value
Ω	$[-1,1] \times [-1,1]$
mesh size	33×33
A	$\{-1,0,1\}$
h	0.01
γ	0.001
λ	0
boundary value	4

Table 4.1: Bush problem parameters

Similarly, we expect that the convergence rate of algorithm 4.1 will not depend strongly on the choice of discount factor, λ, because, unlike equation 4.1, equation 4.7 is not linear in λ.

To illustrate the effect of algorithm 4.1 upon the convergence rate for a two dimensional problem, the Bush problem was solved with the parameters in table 4.1 . When algorithm 3.2 was used, 422 iterations were required for convergence. However with algorithm 4.1 only 56 iterations were required. With a shorter timestep, $h = 0.001$, algorithm 4.1 still required 56 iterations. With a larger discount factor, $\lambda = 1$, and the original timestep, algorithm 4.1 required 44 iterations.

4.1.4 Order of Processing

An improvement in the convergence rate may be possible by considering the order in which nodes are processed. The idea comes from the fact that, provided the cost function is non-negative, an optimal trajectory will always trace a non-increasing path through the value function. Consequently, the value of a node will generally depend more upon lower valued nodes than upon higher valued nodes. Therefore if nodes are processed in increasing order of value, changes in the value function will propagate to higher valued nodes within the same iteration.

initial guess	unsorted	sorted	two pass
$v(x) = \frac{\lvert x\rvert}{2}$	6	4	2
$v(x) = 0$	6	6	6
$v(x) = -\frac{\lvert x\rvert}{2}$	9	9	9

Table 4.2: number of iterations required for 1D problem

This is essentially the same reason that nodes are sorted within van de Panne's AGDP method (section 3.3). It should take fewer iterations for the effects of a single change in the value function to 'ripple out' when processed in this order than if some other order were used.

To test the effect of processing nodes in increasing order of value the 1D problem was solved with three different initial guesses for the value function. One initial guess was the correct shape (but with the wrong values), one was simply zero and the final one was the negation of the correct shape. In principle, sorting nodes by value is only helpful if the nodes are in approximately the right order. These three cases therefore represent best case, neutral and worst case respectively. Table 4.2 shows the number of iterations required for convergence when the nodes were processed in left-to-right order (the *unsorted* column) and in order of increasing value (the *sorted* column).

Sorting the nodes provided some reduction in the number of iterations required in the best case. However, in the other two cases nothing was gained. It is not clear that sorting the nodes provides a significant improvement.

Observation of the solution evolving step by step reveals a useful fact. The algorithm is choosing the wrong control in many instances, even when the shape of the value function was initially correct. The reason is that the 'ripple' of modifications to the value function perturbs the local gradient of the function. The perturbed shape is used to determine the control at a node, and is therefore quite likely to get the wrong control. This 'ripple effect' can be addressed by using two passes through the nodes. In the first pass the control values are determined. In the second pass the value function is updated. In this way the 'ripple' effect is eliminated. Pseudocode for the revised algorithm, incorporating both these modifications is given in algorithm 4.2

The *two pass* column of table 4.2 shows the number of iterations required when the nodes are processed with a two pass algorithm and in increasing order of value. The best case performance is significantly improved, whilst the other two results remain unaltered. Since there is no error in the numerical solution, the final iteration merely confirms that convergence has occurred and does not alter the value function. Consequently, in the best case, the correct solution is obtained in a single iteration and cannot be improved upon.

```
procedure control_pass
  for $n \in N$
    $u^* = \min_{a \in A}\{$evaluate$(n, a)\}$
    $n.a = a$ giving $u^*$

procedure value_pass
  for $n \in N$
    $u^*$ =evaluate($n$,$a$)
    $\delta_n = |n.v - u^*|$
    $n.v = u^*$
  $\delta_{\max} = \max_{n \in N}\{\delta_n\}$

repeat
  call control_pass
  sort $N$ in order of increasing $n.v$
  call value_pass
until $\delta_{\max} < \gamma$
```

Algorithm 4.2: sort-by-value and two-pass modifications

initial guess	unsorted	sorted	two pass
$\frac{B(x)}{2}$	52	40	35
0	56	50	48
$-\frac{B(x)}{2}$	77	67	46

Table 4.3: number of iterations required for the Bush problem

Of course, this is an artificially simple example that can be solved in a single iteration. Higher dimensional problems, like the Bush problem, cannot be solved quite so trivially and provide a better test of the modifications we have described. Therefore, the same experiment was repeated with the Bush problem, using the parameters in table 4.1. This time the best case and worst case initial guesses were provided by the analytic solution and its negation, both divided by two. Table 4.3 shows the results of this experiment ($B(x)$ denotes the analytic solution of the Bush problem).

Clearly, the sort-by-value modification provides a small but significant improvement in all three cases. Although sorting the nodes imposes an additional computational burden, it is typically an insignificant cost when compared to the cost of additional iterations.

Similarly, the two-pass modification provides a reduction in the number of iterations in all cases. However this gain is offset by the additional computational cost imposed by the two-pass modification, which arises because another evaluation of the plant equation is required for each node in the `value_pass` routine in algorithm 4.2. In general though it is the number of control passes (calls to `control_pass`) that we wish to minimise, as this contains the inner optimisation loop. For the Bush problem it is sufficient to check three control values, $\{-1, 0, 1\}$,

rule[†]	iterations*
never	48(48)
$\delta_{max} > \gamma$	9(289)
$\Sigma_{vp} < 5$	14(84)
$\delta_{max} > \gamma$ and $\Sigma_{vp} < 5$	14(77)
$\delta_{max} > \gamma$ and $\Sigma_{vp} < 5$ and $\delta_{max} > \frac{\delta_{initial}}{10}$	13(71)

Table 4.4: rules for determining number of additional value passes

[†]Additional value passes were performed while the rule evaluated as true
*The number in brackets is the number of values passes.
Σ_{vp} is the number of additional value passes performed in the current iteration

hence `control_pass` is only three times most costly than `value_pass`. However, this is a rather special case and in general we cannot expect that the search for an optimal control will be so limited. Therefore in the general case the two-pass modification appears to provide a small net gain.

4.1.5 Multiple Value Passes

The fact that the `control_pass` routine is several times more costly than the `value_pass` routine suggests an additional optimisation. It may be more efficient overall to perform a number of value passes for each, relatively expensive, control pass. We call this the multiple-value-pass modification.

There are two questions to be answered before adopting the multiple-value-pass modification. Firstly, does it actually accelerate convergence? Secondly, how many value passes should be made for each control pass?

To address these questions, we have once again used the Bush problem as our benchmark, with the same parameters as before and incorporating the previous algorithm modifications: limit-of-a-series, sort-by-value and two-pass. The latter is in any case a prerequisite for the multiple-value-pass modification. Table 4.4 shows the number of iterations required when additional control passes were performed according to a number of different heuristic rules.

If the control pass is very much more expensive than the value pass, one might choose to perform the value pass until convergence is obtained, i.e. while $\delta_{max} > \gamma$ (the second row in table 4.4). In our experiment this rule used the fewest total iterations (9), but required a very high number of value passes (289), and is therefore unlikely to the be the most efficient rule in practice.

An alternative is simply to perform a constant number of additional value passes for each control pass (row 2 in table 4.4). There is no obvious way to determine the ratio of value passes to control passes, but, in practice, we have found that five additional value passes works well. With this rule the total number of iterations was slightly higher (14), but the number of value passes was much improved (84).

A small improvement was possible by combining both the previous rules (row 4 of table 4.4). In this case additional value passes were performed only while changes in the value function were above the convergence threshold and only up to a fixed maximum number of value passes. This resulted in a small reduction in the number of total value passes (77).

Another small improvement was possible by introducing the rule that additional value passes should be performed only while the maximum change in the value function is not small compared to the initial maximum change in the value function (the final row in table 4.4). The reasoning behind this rule is that if the value function is changing rapidly then the control function is probably also changing rapidly and hence there is little point in 'polishing' the solution before performing another control pass.

Incorporating this modification, using the most successful rule, into our algorithm gives algorithm 4.3. The additional value pass rule is effectively parameterised by two numbers, in this case we have used 5 and 10. These values are merely ones that seem sensible and appear to work well for the problems we have tried. For another problem, a different choice of these parameters may provide better performance.

```
repeat
  call control_pass
  sort $N$ in order of increasing $n.v$
  call value_pass
  $\delta_{initial} = \delta_{max}$
  $\Sigma_{vp} = 0$
  while $\delta_{max} > \gamma$ and $\Sigma_{vp} < 5$ and $\delta_{max} > \frac{\delta_{initial}}{10}$
    call value_pass
    $++\Sigma_{vp}$
until $\delta_{initial} < \gamma$
```

Algorithm 4.3: multiple-value-pass modification

4.2 Boundaries

The fact that the solution procedure takes place in a finite domain can, and often does, affect the outcome of the procedure. This is clear from figure 3.7 which differs significantly from the analytic solution shown in figure 3.3. So the question arises, how is the solution affected by the boundary and how significant is the effect?

Depending on the topology of the trajectories at the boundary there are actually four cases

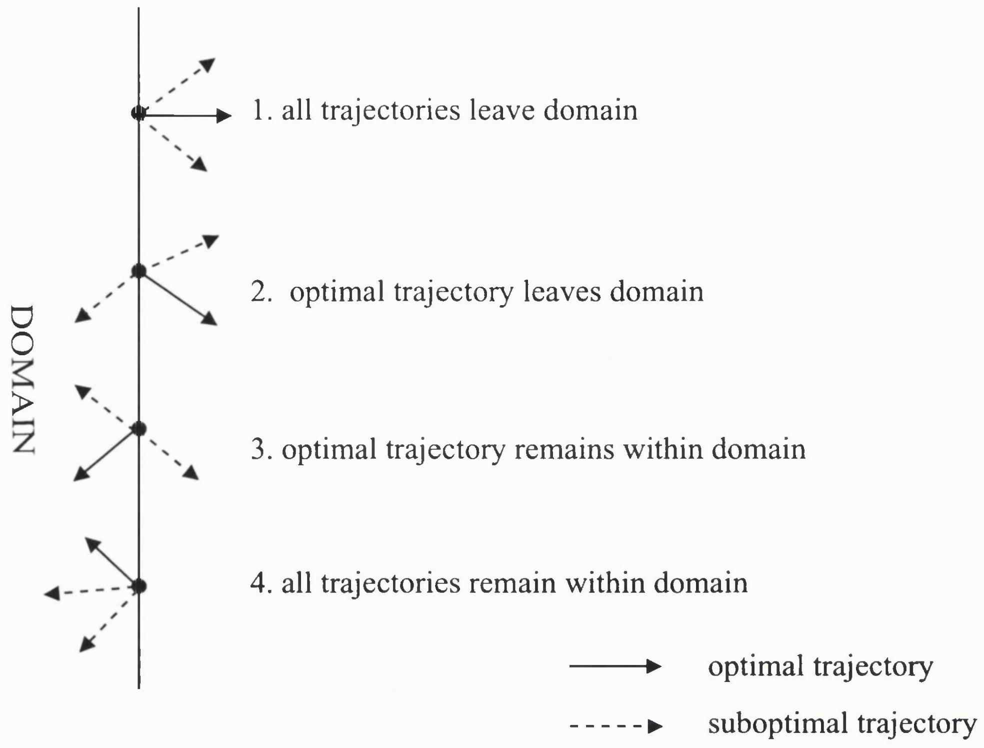

DOMAIN
1. all trajectories leave domain
2. optimal trajectory leaves domain
3. optimal trajectory remains within domain
4. all trajectories remain within domain
optimal trajectory
suboptimal trajectory

Figure 4.2: the four cases of boundary nodes

to consider, which are illustrated in figure 4.2. Recall from section 3.4.2 that the convergence proof depends upon the assumption that all trajectories remain within the domain for all time. This corresponds to case 4 in figure 4.2. If every node on the boundary of the domain belongs to this category then the presence of the boundary has no effect on the solution. However, it is usually not possible or practical to choose a domain that satisfies this requirement. The Bush problem is, once again, a good example here because, with the type of rectangular mesh we have used, cases 1-4 are all present.

Figure 4.3 illustrates which regions of the boundary of the Bush problem belong to which of the cases illustrated in figure 4.2. Case 4, 'all trajectories remain within domain', occurs at the boundary segments $\{p = 1, \dot{p} = [-1, 0]\}$ and $\{p = -1, \dot{p} = [0, 1]\}$ because the velocity of the particle along these boundary segments always drives it into the domain.

Case 3 in figure 4.2, where the optimal trajectory remains within the domain, is completely benign, like case 4. The solution depends only upon the optimal trajectory and hence is unaffected by the presence of boundary segments of this type. Case 3 occurs in the Bush problem along the boundary segments $\{p = [-0.5, 1], \dot{p} = 1\}$ and $\{p = [-1, 0.5]], \dot{p} = -1\}$ where the velocity of the particle is along the boundary segment and the applied control can either drive the particle into or outside of the domain.

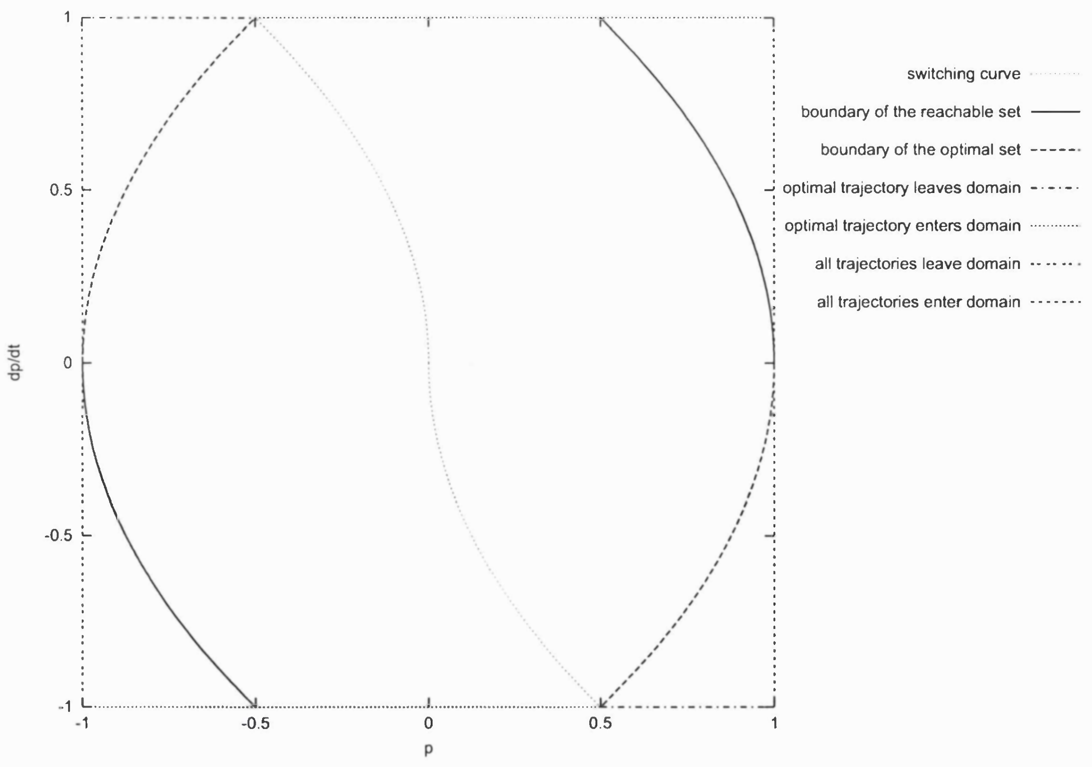


Figure 4.3: anatomy of the Bush problem

Case 2 in figure 4.2 is a little more complicated. In this case, the trajectory that would be optimal if the domain were larger leads outside the boundary of our finite domain, but one or more trajectories are available that remain within the domain. This occurs in the Bush problem along the leftmost quarter of the upper boundary, $\{p = [-1, 0.5], \dot{p} = 1\}$ and the rightmost quarter of the lower boundary, $\{p = [0.5, 1], \dot{p} = 1\}$. In such cases the solution procedure will choose the best available trajectory that still remains within the domain. Hence some part of the solution will be sub-optimal and, in this region, the value function will be higher than would be the case if the domain were larger. In figure 4.3, the regions that are influenced by these boundary segments extend to the curved line segments labelled 'boundary of the optimal set'.

Case 2 need not cause a problem, so long as the domain is well chosen. The resulting trajectories are still optimal in the sense that it is not possible to do better without leaving the domain. If such trajectories are acceptable for a given application then all is well. The convergence proof described in section 3.4.2 has been extended to show that convergence is still obtained when the boundary is of this type [10, 22].

The final case, case 1 in figure 4.2, where all trajectories leave the domain is more problematic. This occurs in the Bush problem in the upper half of the right hand boundary, $\{p = 1, \dot{p} = (0, 1]\}$, and the bottom half of the left hand boundary, $\{p = -1, \dot{p} = (0, -1]\}$.

Along these boundary segments all possible trajectories leave the domain because the velocity of the particle always drives it outside. In order to to assign a value to nodes on these boundary segments, the idea of a boundary value is introduced. All regions outside the domain are considered to have the same boundary value, which is chosen to be higher than the possible range of the value function within the domain. When all trajectories from a given node leave the domain, that node is assigned the boundary value. The high boundary value then propagates into the domain as far as the manifold formed by the set of trajectories that remain within the domain, the boundary of the *reachable set* (shown in figure 4.3).

We define the reachable set as the set of points within the domain from which a trajectory exists that (i) connects the point to the target set and (ii) remains within the domain for all time.

If we compare figure 3.7 with figure 3.3, we see that the numerical solution is significantly different from the analytic solution in the region outside the reachable set. Also, the high boundary value can be seen to blend into the rest of the solution across a region which is several cells wide. Hence, in practice, the effect of the boundary value extends some way into the reachable set.

In principle, there should be a discontinuity at the boundary of the reachable set. All trajectories outside this boundary eventually leave the domain and hence the value function outside the reachable set is not connected to the value function inside it. In practice, however, the mesh that we use cannot model this discontinuity, and at best can only model a finite region of high gradient. This raises the question, how do we know that the rest of the solution is still valid, and if it is, to what extent is the solution corrupted by the boundary value?

An answer to the problem emerges from a shift in viewpoint. The numerical procedure is unable to model the reachable set in a crisp, binary fashion with each point in phase space either belonging or not belonging to that set. If we instead treat membership of the reachable set as a continuous function ranging between 0 (not a member) and 1 (a member), then it is possible to model that function on the numerical mesh. This is equivalent to the notion of a fuzzy set. Let us define the membership function by

$$m(x) = \begin{cases} 1 & x \in T \\ 0 & x \notin \Omega \\ \sum_j w_j(y_i) m(x_j) & x = x_i \end{cases}$$

where T is the target set, Ω is the domain, x_i is the position of the i'th node and $w_j(y_i)$ is the interpolation coefficient associated with node j at $y_i = x_i + hf(x_i, a_i)$. The effect is that the membership of a node equals that fraction of its value which is determined by nodes within the reachable set.

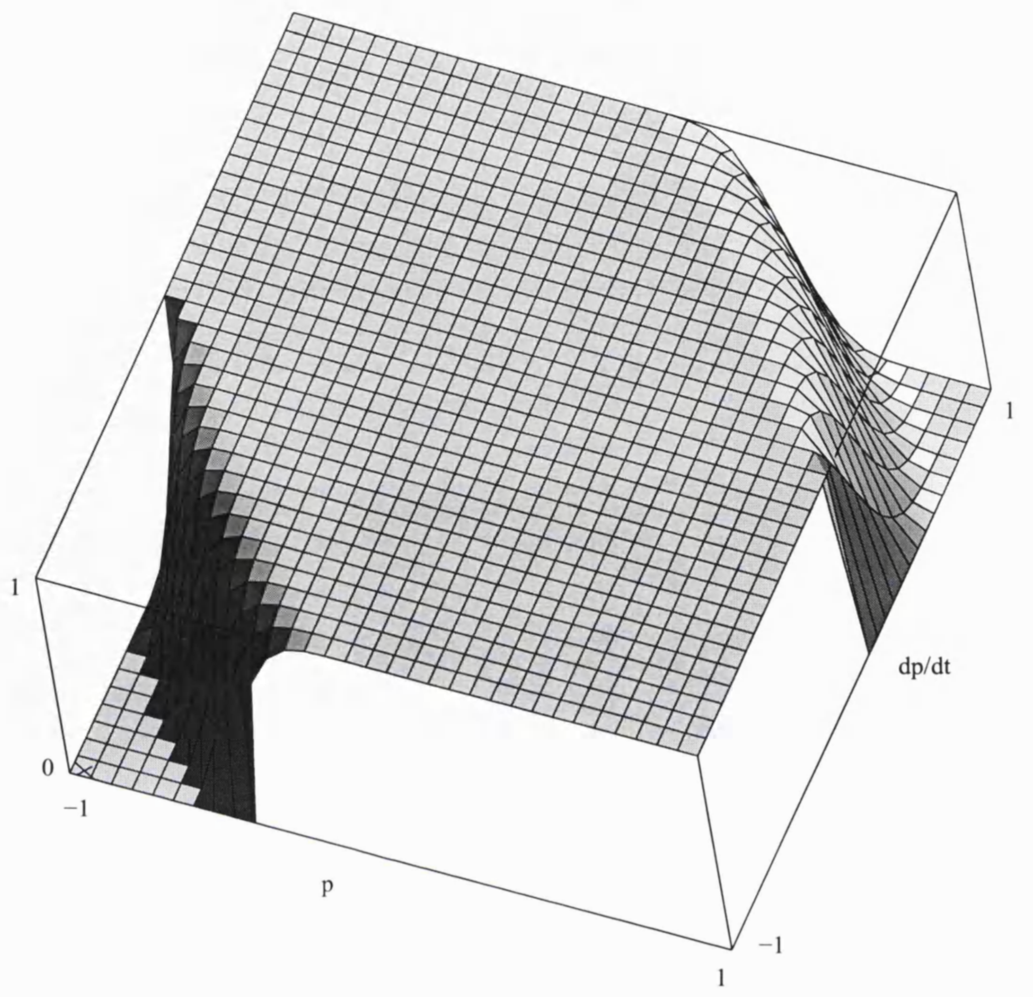


Figure 4.4: membership function for the Bush problem

Figure 4.4 shows the membership function for the Bush problem. Over most of the domain, membership is precisely equal to one. The influence of the boundary value on the value function is proportional to $1 - m(x)$, and hence most of the domain is completely unaffected by the boundary value. We have therefore found a way, not only to quantify the influence of the boundary value on any node, but also to demonstrate that the boundary of the reachable set does not corrupt the majority of the solution.

4.3 Meshing

So far, we have used meshes based upon dividing each dimension of the phase space into a set of equally spaced intervals, forming rectangular cells of equal size. Let us call these *regular meshes*. Regular meshes are inefficient because, in general, some regions of the domain require a higher density of mesh points than others. For example, the Bush problem requires a high density of mesh points around the switching curves, but relatively few points elsewhere. If it were possible to vary the density of the mesh across the domain then much computation could be saved. However, it is not sufficient merely to have the capability to vary the density of the mesh, we also require some means of deciding how that density should be varied. There are, therefore, two problems to be solved. Firstly, it is necessary to design a structure to represent the variable density mesh. Secondly, we need a method to determine how well each part of the mesh 'fits' the problem. These two problems are largely independent, hence we shall address them separately in what follows.

We have implemented an adaptive meshing strategy, that begins with a coarse mesh and then iteratively refines the mesh where it is most beneficial. In this way an efficient mesh may be discovered with no prior knowledge of the solution; an important attribute for most practical problems. We begin with a look at some of the alternatives for the structure of the mesh.

4.3.1 Adaptive Mesh Design

We believe that an ideal adaptive meshing strategy should satisfy the following requirements:

1. use the fewest nodes possible,
2. facilitate rapid lookup of the approximating function embodied by the mesh,
3. be scalable to higher dimensions,
4. support local adjustment of the approximating function,
5. support local refinement of the mesh, and
6. incorporate graceful handling of functions that are not differentiable or even continuous everywhere.

The first requirement is simple to explain, fewer nodes means less calculation and hence the ability to tackle more complex problems.

Rapid lookup is essential because the search for an optimal control at each node, which is the innermost loop of the control pass (see algorithm 4.2), requires looking up the value function many times. If value function lookup is slower than integrating the equations of motion over one step, then it becomes a limiting factor and the speed of the control pass will be roughly proportional to the lookup speed.

An adaptive mesh should be scalable to a modest number of dimensions. One could imagine tackling problems in up to six dimensions. However, such a mesh cannot, of itself, scale to very high dimensions. A lower bound on the complexity of these meshes comes from the fact that, for a rectangular domain, one node is required at every corner of the domain. Thus in k

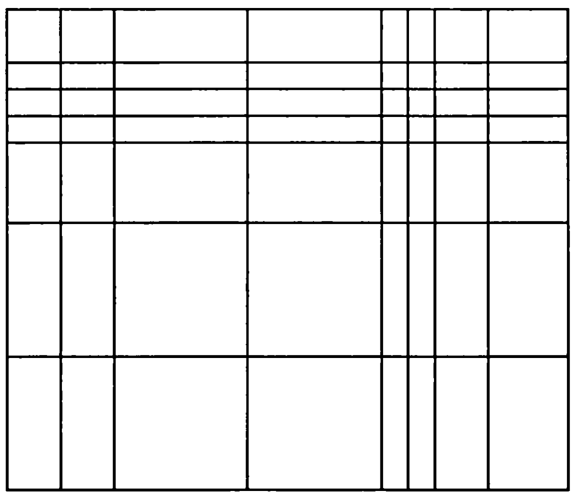


Figure 4.5: mesh formed by varying the spacing along each axis

dimensions 2^k nodes are required for the corners. In all but the most trivial cases the complexity of a variable density mesh will be higher than $O(2^k)$, but the data structures and algorithms used to implement that mesh should ideally not preclude the use of a mesh with 2^k nodes.

Requirement four stems from the fact that the HJB equation tells us something about the local shape of the value function. By finding the optimal control at a node we obtain the gradient of the value function at that node and in a particular direction. Therefore we wish to be able to modify the local shape of the value function in such a way that its gradient matches the information we have obtained without changing the shape of the value function elsewhere.

The requirement for local refinement of the mesh is really a prerequisite for adaptive meshing. We shall see in section 4.3.2 how to obtain information about where the mesh needs to be refined. To use such information effectively requires that we can increase the mesh density only within a neighbourhood of where it is required.

The final requirement (5) is due to the nature of the value functions that we are trying to model.

Having established the requirements, let us examine some of the options for building an adaptive mesh. One such mesh can be formed by taking the regular meshes we have used so far, and allowing the spacing along each axis to vary. This leads to a mesh like that in figure 4.5. The principle virtues of this scheme are that it is very simple to implement, and looking up values within the mesh is very quick. Given a mesh of this type, and a means of deciding where each axis needs to be subdivided further, it should be possible to discover a mesh that uses a minimum of subdivisions along each axis. This method is a significant improvement upon the regular mesh if the regions requiring high mesh density are confined to a small fraction of the length of one or more axes. Unfortunately, in many cases it is only marginally more efficient than a regular mesh. For example, the Bush problem requires high mesh density along the length of its switching curves. Using the parameters described in section 3.2, the switching curve stretches the entire length of the velocity axis, and half the length of the position axis; yielding only a small reduction in the number of nodes. In short, this method does not fully support local refinement.

4.3.2 Gruene's Unstructured Adaptive Simplicial Mesh

One alternative, that does fulfil the requirement for local refinement, uses an unstructured mesh of simplices, like that shown in figure 4.6. Gruene used such a mesh in an adaptive meshing scheme for solving HJB equations [32]. He proposed a pointwise error estimate, $e(x)$, for the purpose of deciding where to refine the mesh. The error estimate provides information about how well the function embodied by the mesh, the *approximating function,* $u(x)$, fits the true value function of the semi-discrete problem $v_h(x)$. It is defined by

$$e(x) := |u(x) - T(x, u(x))| \tag{4.8}$$

where $T(x, u(x))$ is the value function update rule, given in equation 3.38.

Unstructured simplicial meshes pose two specific problems. Firstly, such a mesh provides no built-in method to determine which simplex encloses an arbitrary point within the domain. Something must be done to solve this problem efficiently in order to provide rapid value function lookup. Secondly, an explicit record must be kept of each simplex in the mesh, and the number of simplices required to mesh the domain may not scale well. There is no known way to decompose a hypercube into fewer than 324 simplices in six dimensions, 13136 simplices in eight dimensions, or 928780 simplices in 10 dimensions [90]. We therefore expect the scalability of an unstructured simplicial mesh to be far worse than is the case for a mesh based around decomposing a rectangular domain into smaller rectangles.

4.3.3 Spatial Subdivision

A more scalable approach is to build a mesh based upon a spatial subdivision such as the k-dimensional equivalent of a quadtree [83]. For the sake of convenience, we shall refer to all k-dimensional quadtree equivalents as quadtrees, even though they are actually 2^k trees. The idea is to start with a rectangular region, and then divide it in two along each axis, forming 2^k rectangular sub-regions, or *cells*, inside the original. Each cell can then be further sub-divided in a recursive fashion, forming a tree-structured hierarchy of rectangular regions. Figure 4.7 illustrates the quad-tree.

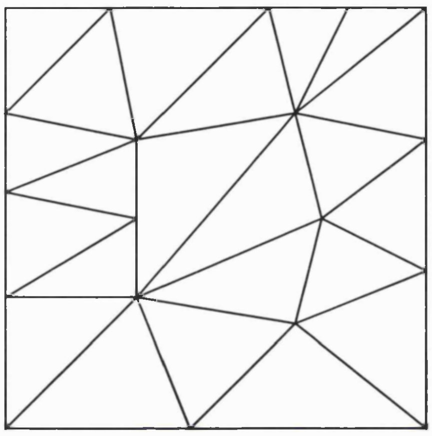


Figure 4.6: an unstructured mesh of simplices in 2D

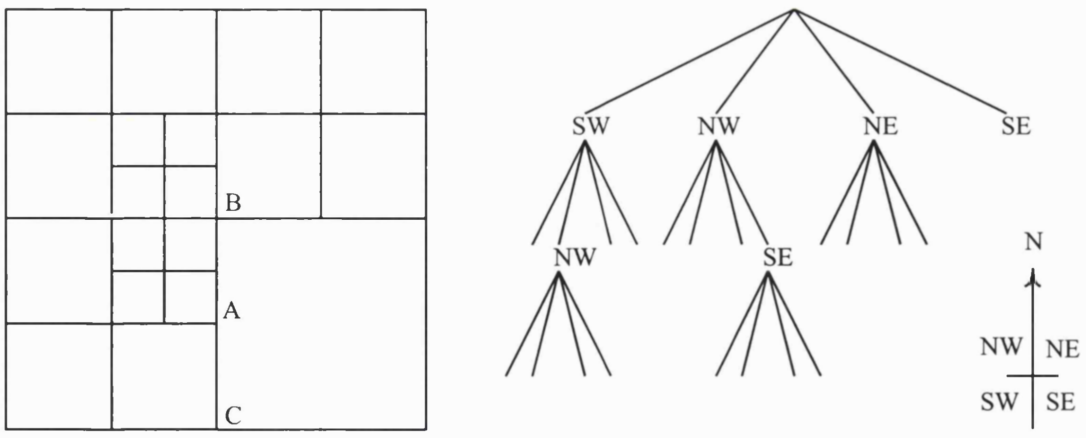


Figure 4.7: the quadtree

The letters SW,NW,NE or SE denote which quadrant each sub-region belongs to.

The quadtree allows a mesh to be refined locally and provides rapid value function lookup. It is not necessary to store an explicit list of simplices if multilinear interpolation, or an implicit decomposition of the cell into simplices, is used. The Kuhn triangulation, illustrated in figure 4.9, is one such implicit decomposition.

Lookup speed is slower with the quadtree than the regular mesh, or the mesh formed by varying the spacing along the axes (figure 4.5), since it requires a descent of the tree structure. However, the lookup overhead is small and an accurate mesh can be constructed with far fewer nodes than with either of the aforementioned methods.

One new problem has been introduced with the quadtree. Some of the nodes of the mesh lie at the boundary of two cells, where one cell is subdivided more than its neighbour. The node labelled 'A' in figure 4.7 is an example of such a node. The value of A may not be coincident with the value obtained by interpolating between nodes B and C, thus creating a discontinuity in the approximating function. Note that this discontinuity may be introduced even though there should be no discontinuity in the value function at that point. We call nodes such as A, *hanging nodes.* One way of dealing with hanging nodes is to treat them as a special case and constrain their value, such that there is no discontinuity. Thus, in this example, the value of node A would be defined by interpolation between nodes B and C. When the discontinuities that may occur at hanging nodes have been eliminated the mesh is said to be *conformant.*

In fact, a close relative of the quadtree, called the kd-tree [83] is preferable for this application. The kd-tree is a binary tree that divides a rectangular region in two along a single dimension at each node of the tree. Successive subdivisions can be made along the available dimensions in arbitrary order. This is a more natural representation for a spatial subdivision in an arbitrary number of dimensions because the data structure, that represents nodes of the tree, is independent of the number of dimensions. Figure 4.8 illustrates the structure of a kd-tree.

The kd-tree is more efficient than the quadtree in two ways. Firstly, the quadtree may create more cells (leaves of the tree) than are required because each new subdivision creates 2^k new cells. This may be inefficient if there is any overhead associated with each cell (leaf of the tree). By contrast the kd-tree only creates two new cells at each subdivision, and hence can avoid creating too many cells. Secondly, the kd-tree supports *anisotropic subdivision*, meaning that each dimension need not be subdivided equally. If the function that we seek to approximate varies more rapidly along one dimension than another then it is best to use more subdivisions along the more rapidly varying dimension. Phase spaces have heterogeneous dimensions that are unlikely to produce equal variation along different axes, hence the kd-tree is likely to be more efficient for these problems than the quadtree.

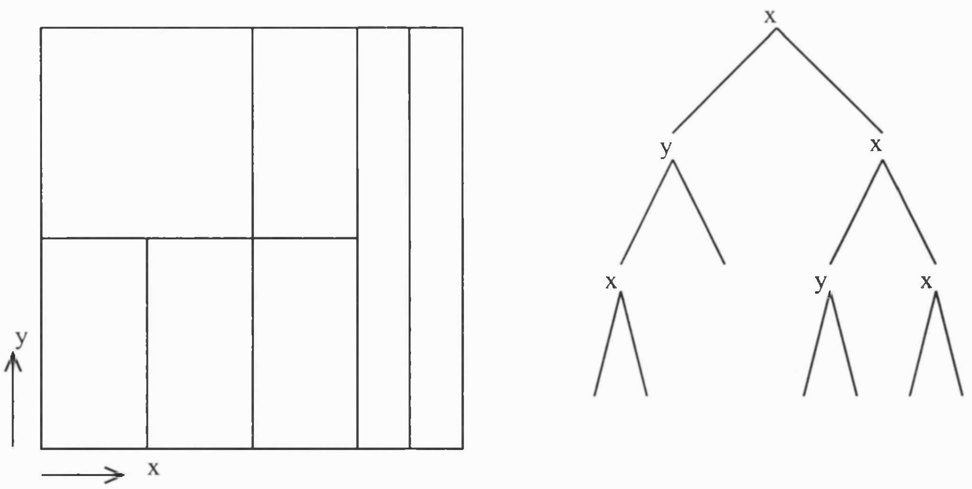


Figure 4.8: the kd-tree in two dimensions

The letter x or y in the right hand diagram denotes which dimension is split at each subdivision. By convention, if a region is split along the x axis then the left hand branch of the tree contains the sub-region with x coordinates less than the splitting value, and the right hand branch contains the sub-region with x coordinates greater than or equal to the splitting value.

4.3.4 Munos and Moore's kd-tree Mesh

Munos and Moore describe a kd-tree based meshing scheme for solving optimal control problems in [61, 60, 59, 58]. Their scheme is isotropic; the direction of each subdivision is chosen such that it follows the direction of its parent subdivision in sequence. This eliminates the need to decide which dimension should be subdivided next, but fails to exploit the efficiencies that may be possible with anisotropic subdivision.

To interpolate within cells, they use a decomposition of the rectangular cells into simplices, known as a Kuhn triangulation. This provides a quick and simple method to find the vertices of the simplex containing any given point, without the need to store any additional data. Figure 4.9 illustrates the Kuhn triangulation in three dimensions and explains how to find the simplex containing a given point.

Compared to multilinear interpolation (section 3.3.1), the Kuhn triangulation is more scalable because interpolation has complexity $O(k+1)$, in k dimensions, whilst multilinear interpolation is $O(2^k)$. It is therefore significantly faster than multilinear interpolation in any more than two dimensions. For example, it is about three times faster in four dimensions. However, the Kuhn triangulation may be less reliable than multilinear interpolation. There are 2^{k-1} different possible orientations of the Kuhn triangulation and hence there are 2^{k-1} different possible results of interpolation. In general, some of the possible orientations will 'fit' the

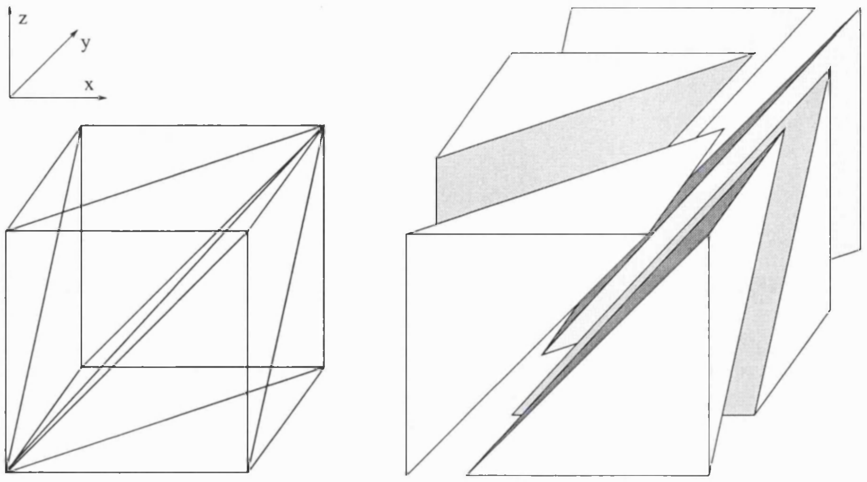


Figure 4.9: the Kuhn triangulation in three dimensions (from [60])

To determine which vertices form the simplex containing a point, x, within the cell, use a local coordinate system with the origin at one corner of the cuboid and $(1, 1, 1)$ at the opposite corner. The first vertex is the origin. Subsequent vertices are found by adding a unit basis vector to the previous vertex, starting with the unit basis vector corresponding to the largest component of x, then the second largest component of x, and so on. For example, the point $(0.3, 0.7, 0.5)$ yields the following simplex: $(0, 0, 0), (0, 1, 0), (0, 1, 1), (1, 1, 1)$.

underlying function better than others. Multilinear interpolation, which is uniquely defined, avoids the worst behaviour of the Kuhn triangulation where the orientation provides a poor fit to the underlying function.

In [63, 60, 61] Munos and Moore describe a variety of means to determine which cells to subdivide, including:

1. a measure of the absolute differences in the value function at the corners of each cell,
2. a measure of the variance of the value function at the corners of each cell,
3. a measure of the absolute differences in the control function at the corners of each cell,
4. combinations of the above, and
5. a measure of the *influence* that regions where the control function is changing have upon each individual cell, multiplied by a measure of the variance in the value function at subsequent steps in optimal trajectories taken from the corners of each cell.

Munos and Moore assess the relative performance of these options and conclude that option 5 provides the best performance.

None of the measures tested by Munos and Moore provide an estimate of the curvature of the value function. Apart from the *influence* measure, they are all more closely related to the gradient of the value function. As we are trying to approximate a curved function with a piecewise linear or multilinear function then the the primary determinant for subdivision should be curvature. For this reason we prefer Gruene's error estimate [32], because this does provide information about curvature.

Munos and Moore's meshing scheme is not conformant, meaning that it permits discontinuities such as those that occur at hanging nodes. Munos et al justify their non-conformant mesh with a mathematical result presented in [57, 59, 58]. This result proves the convergence of the HJB method even in the presence of such discontinuities. We state this result next.

4.3.5 Munos and Moore's Result

For some discounted optimal control problem, whose value function obeys equation 3.21, let the domain Ω be discretised, such that

$$\forall x \in \Omega, \quad \inf_{n \in \Omega} \|x - x_n\| \leq \delta$$

where x_n is the position of node n. In other words, no point within the domain is further than δ from the nearest node. Also, let $\eta = x_n + hf(x_n, a^*)$ where a^* is the true optimal control and let $\eta_j = x_n + hf(x_n, a_j^*)$ where a_j^* is the control chosen as optimal at iteration j. If the approximate value function after j iterations, u_j^δ, is determined by an update rule of the form

$$u_{j+1}^\delta(x_n) = (1 - \lambda h)u_j^\delta(\eta_j) + hl(x_n, a_j^*)$$

then Munos and Moore prove that

$$\lim_{j \to \infty, \delta \to 0} u_j^\delta = v$$

where v is the true value function, so long as the following condition is met

$$w_i(\eta_j) = w_i(\eta) + O(\delta)$$

where w_i are the mesh interpolation coefficients (see section 4.3.4 associated with point η.

The result demonstrates that a mesh with discontinuities may be used, provided they are $\leq O(\delta)$.

4.3.6 Gruene's kd-tree Mesh

Gruene et al also describe a kd-tree based meshing scheme for solving optimal control problems [33, 27]. Unlike Munos, they obtain a conformant mesh by constraining the value of hanging nodes, as described in section 4.3.3.

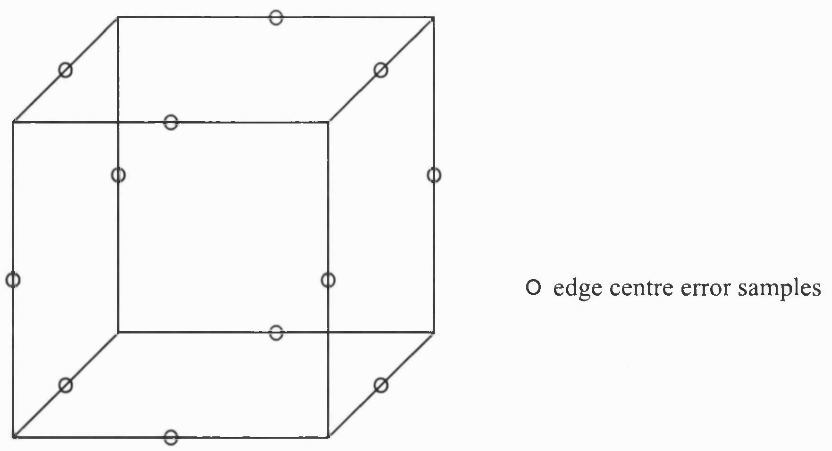


Figure 4.10: edge centre sample points for the error metric

Their scheme is anisotropic, so when a cell is subdivided a decision is made about the direction in which to split the cell. Gruene uses the error estimate, given in equation 4.8, to sample the error at the centre of every edge. Figure 4.10 illustrates these edge samples in the three dimensional case. Depending upon these error measurements, a cell may be subdivided in one or more directions.

Gruene et al require that no k dimensional cell can share any of its $2k$ faces with more than two neighbouring cells. Let us call this the 2:1 property. The 2:1 property ensures a progressive transition from coarse to fine mesh, which can make the adaptive meshing strategy more robust since the neighbourhood of a region of high mesh density is itself more likely to require high mesh density than the average. The 2:1 property reduces the likelihood that an adaptive mesh strategy will arrive at a mesh density that is too low in places. The disadvantage is that the total number of nodes in the mesh is likely to be higher than if the property were not imposed.

Multilinear interpolation is used within the cells of Gruene's mesh. This is required to ensure continuity of the approximating function at cell interfaces in three or more dimensions. Figure 4.11 shows an example where simplicial interpolation yields a discontinuity at the cell interface.

4.3.7 Neural Networks

The use of neural networks to approximate the value function was explored by Munos et al [78]. They used three layer feed-forward networks and derived the gradient descent rule for integrating the Hamilton-Jacobi-Bellman equation inside the domain. A poor approximation to the value function was obtained when compared to the solution obtained with a regular mesh. Munos et al attribute the poor approximation to the fact that their gradient descent method converges to one of an infinity of so-called generalised solutions of the HJB equation. Thus, their

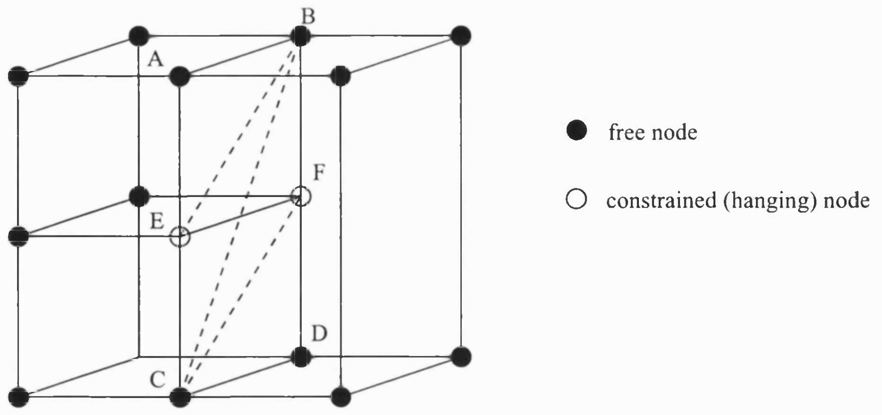


Figure 4.11: example where simplicial interpolation yields discontinuity

At the cell interface defined by the plane ABCD there is a discontinuity if simplicial interpolation is used. The dotted lines EB and CF show the boundaries between simplices in the left hand cell structure at the plane. The dotted line CB shows the equivalent boundary for the right hand cell structure. Continuity cannot be guaranteed because these lines do not coincide. There is no discontinuity with multilinear interpolation.

method may fail to find the unique viscosity solution for a given optimal control problem.

4.4 Adaptive Mesh Implementation

Having described the reasons for our choice of an adaptive meshing scheme based upon the kd-tree, and some related work, we shall now describe our implementation.

The scheme is built upon three fundamental data structures (classes in our C++ implementation). The first represents nodes of the mesh. These are stored as key-value pairs within a dictionary type structure, which is implemented as a hash table. The coordinates of the node form the key, while the other properties make up the value part. In C++ syntax the value part can be expressed as

```
struct node {
  float value;
  float membership;
  signed char control[m];
};
```

where `m` is the number of dimensions in the control space of the problem. The control vector is stored as a set of 8-bit integers because that provides sufficient resolution for our purposes, and is relatively efficient on storage space. Functions are provided to convert the integer control vector to and from its floating point equivalent. Usually, these are related by a simple scale factor.

Domain coordinates are also stored as integers, but for a different reason. Each subdivision splits a region in two, and division by two is both efficient and accurate if the coordinates are integers and powers of two. Hence the domain, which is the root of the kd-tree, is always assigned a size, in integer coordinates, based on powers of two. This also provides an implicit lower limit on the size that any cell can take since a cell with a width of 1 in any dimension cannot be subdivided further along that dimension. Once again, functions are provided to convert to and from a floating point representation.

The kd-tree itself is a hierarchy of rectangular regions, which we call cells. It is possible to define these rectangles by storing the coordinates of just two of their corners. However, the nodes of the tree actually store the subdivisions, rather than explicitly storing the two corners. This is a very compact representation because each subdivision is perpendicular to an axis, hence it is only necessary to store the index of that axis and (optionally) the coordinate of the subdividing hyperplane along that axis. Our implementation uses a data structure similar to the following:

```
struct subdivision {
    subdivision* left;
    subdivision* right;
    short   split_index;  // index of axis that is split
    short   split_value;  // coordinate of splitting hyperplane
};
```

The `split_value` field is optional because we have chosen to always split each cell into two equal pieces. It could easily be recalculated each time it is required, but in our implementation it is slightly more efficient to store it.

Given the two corners of the domain, which is the root of the tree, the corners of any cell within it can easily be recovered by maintaining the two corners of the current cell as we descend the tree. The following data structure serves that purpose.

```
struct cell {
    subdivision* split;
    short lower_corner[k];
    short upper_corner[k];
};
```

The `lower_corner` is the corner of the cell with the lowest valued coordinates in all of its components. Similarly, the `upper_corner` is the corner with the highest valued coordinates.

The computational domain, which is the root of the tree, is stored as a `cell` structure. Thus to find the cell which contains a given point, x, we first initialise our current cell with the root cell. Then, at each subdivision, if the `split_index` component of x is less than `split_value` the left branch is taken and the `split_index` component of `upper_corner` is set to `split_value`. Otherwise the right branch is taken and the `split_index` component of `lower_corner` is set to `split_value`.

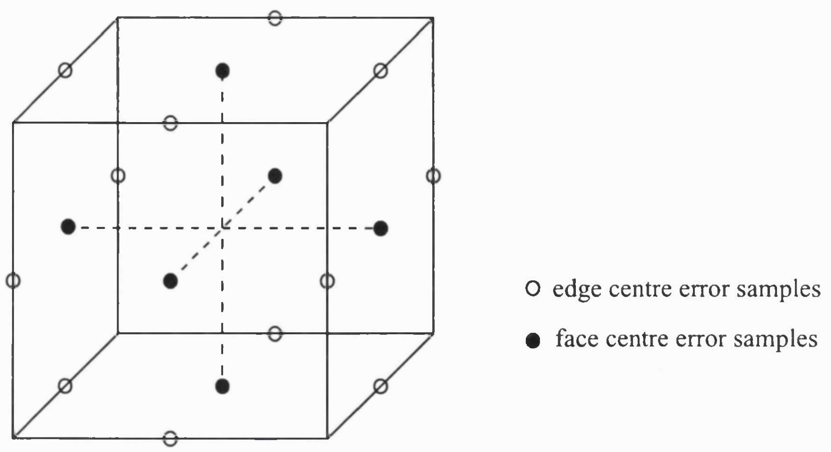


Figure 4.12: face centre and edge centre error sample points in 3D

Having outlined the structure of our meshing system, the next step is to address the question of how subdivisions may be chosen in order to adapt a mesh to some function. We shall explore this question using known analytic functions first, before attempting to apply our meshing scheme to the discovery of unknown functions via the HJB method.

4.4.1 Face and Body Centred Scheme

Our first attempts at building an adaptive, kd-tree based, meshing scheme were implemented before discovering the kd-tree based schemes of Munos and Gruene. Our method worked by sampling the error in the mesh at the centre of each face and at the centroid of the cell. Figure 4.12 shows these face centre sample points in three dimensions and illustrates the distinction between face centre samples and the edge centre samples used by Gruene [33, 27].

The advantage of face and body centre error sampling is that it scales linearly with the number of dimensions ($2k + 1$ samples in k dimensions) and provides information about the magnitude of the error in each direction.

To model a known analytic function, the error, $e(x)$, that we are sampling is just the absolute difference between the mesh value and the known analytic function at a point, i.e.

$$e(x) = |g(x) - u(x)| \tag{4.9}$$

where $g(x)$ is the known analytic function and $u(x)$ is the function embodied by the mesh. The decision to subdivide a cell is made if the maximum error at any of the sample points within the cell exceeds a threshold. Once a cell is marked for subdivision, a second decision must be taken about which of the k dimensions to split. This decision is taken based upon the curvature of the underlying function, i.e.

$$e_j = g(x^c) - \frac{g(x^f_{j+}) + g(x^f_{j-})}{2} \tag{4.10}$$

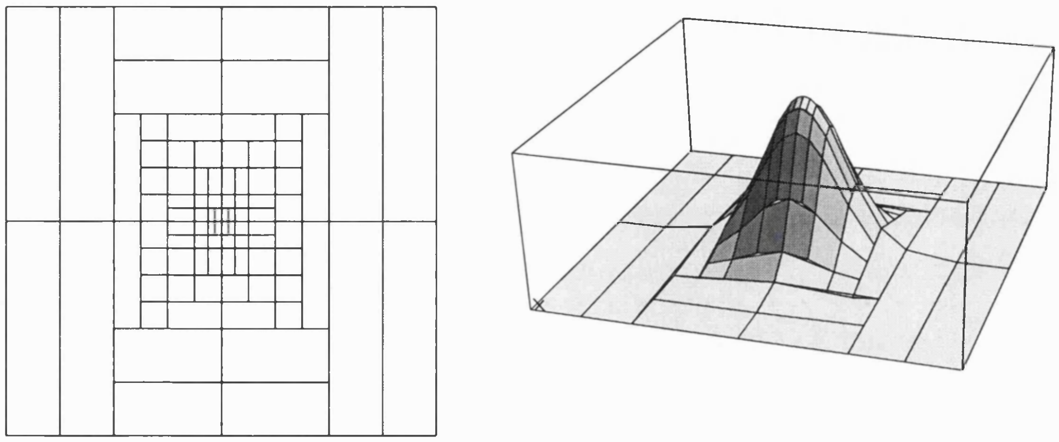


Figure 4.13: Gaussian modelled with kd-tree mesh.

where e_j is the cell error associated with dimension j, x^c is the centre of the cell, x^f_{j+} is the centre of the cell face in the positive j direction, and x^f_{j-} is the centre of the cell face in the negative j direction. The dimension with the largest value of e_j is subdivided.

Figure 4.13 shows the result of modelling a Gaussian function, using the subdivision criteria above. The mesh was refined iteratively by subdividing all cells with an error exceeding a threshold, then repeating the process until no more subdivision was necessary. An initial mesh of 2×2 cells was used as a starting point, with an error threshold of 0.02. Note that this mesh is not conformant and small discontinuities appear in the approximation, just visible around the half-way height of the Gaussian and elsewhere. These occur at the hanging nodes.

We chose to modify this meshing scheme to enforce conformance. This was achieved by introducing constrained nodes, whose value is determined as the mean of the two nodes on either end of the edge on which the node was introduced. The modified conformant algorithm works by sampling the error in all undivided cells. If the maximum error in a cell exceeds the threshold then it is marked for subdivision. A cell is subdivided only if all the nodes at its corners are free, i.e. not constrained. In which case, any new nodes required by the subdivision are added as free nodes. If any of the corner nodes are constrained then they are converted to free nodes. After all the cells have been processed, the mesh is made conformant by introducing additional subdivisions around the new free nodes, adding only constrained nodes where required. The process is then repeated until no more subdivision is necessary.

Figure 4.14 illustrates the results of applying this modified, conformant, meshing to the same Gaussian function as before. The mesh is now conformant but more nodes and cells are

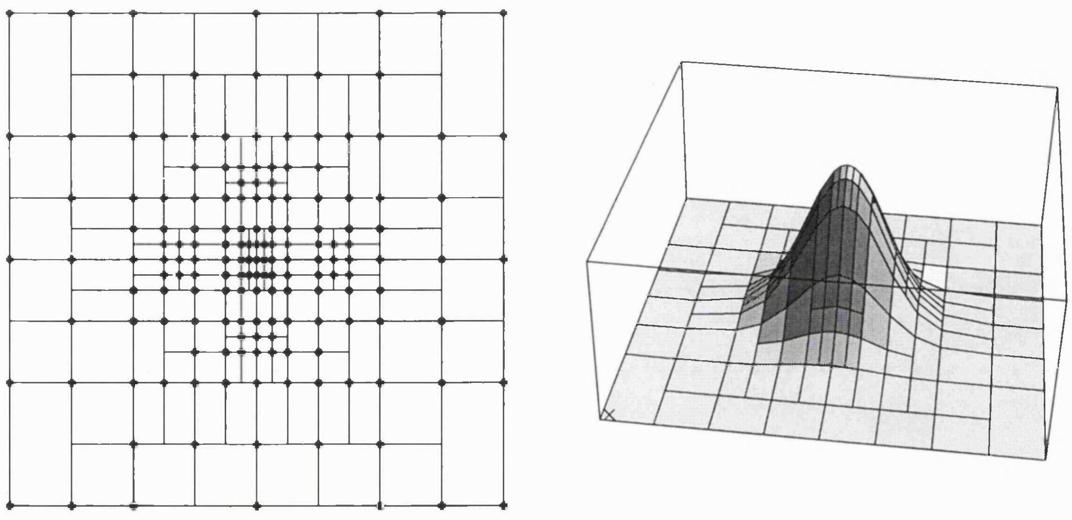


Figure 4.14: Gaussian modelled with conformant mesh

Free nodes are shown with a dot. Constrained nodes have no dot

required than before (compare figures 4.13 and 4.14).

4.4.2 Meshing Scheme Comparison

It turns out that the conformant meshing scheme that we have just described (face and body centred) is less efficient than it could be. The reason for the inefficiency is that free nodes are placed at every corner of the subdividing hyperplane whenever a decision is taken to subdivide a cell. We call this property of the algorithm the *full split* property. The full split property usually results in too many free nodes being added to the mesh.

Figure 4.15 illustrates the inefficiency of the face and body centred scheme. The error sample taken in the centre of the middle vertical edge is above threshold, triggering subdivision of the cells on either side. Three free nodes are added with these subdivisions and two additional subdivisions are then added to make the mesh conformant.

By comparison, Munos' meshing scheme uses fewer total subdivisions but makes the same mistake of adding free nodes all around the new subdivisions. In this case, three free nodes are also added, two of which are hanging. In fact the two hanging nodes contribute little to the accuracy of the mesh because the edges on which they were placed were sufficiently accurate before subdivision.

In general, we wish to minimise the number of free nodes because of the need to search many times for an optimal control at each free node, which is computationally costly. By comparison, the burden of additional subdivisions is small.

Clearly then, Gruene's meshing scheme is the most efficient of the three, since it requires

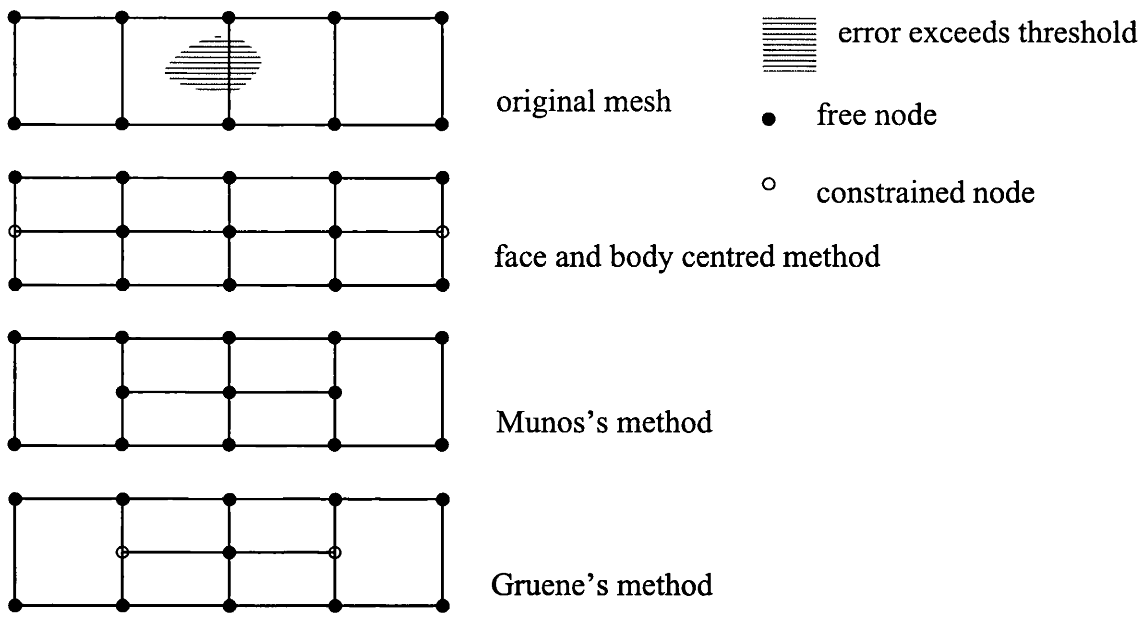


Figure 4.15: meshing scheme comparison

only one new free node and two new subdivisions. The efficiency of this method can be explained by the fact that it only adds free nodes to the specific edges where the error is above the threshold.

4.4.3 Edge Centred Scheme

To address the shortcomings of the face and body centred scheme, a meshing scheme, similar to Gruene's, was implemented. We shall refer to this meshing scheme as the *edge centred scheme* because it uses edge centred error sampling, as shown in figure 4.12. This method does not explicitly build a list of cells that require subdivision, as the face and body centred method does, instead it builds a list of *potential free node sites* where the error is above the threshold.

Free nodes can be placed either on an existing edge or on the site of a constrained node (replacing the constrained node). These are the potential free node sites. Therefore the error is sampled at these points to determine where to place free nodes. The algorithm can be summarised as follows:

1. sample the error at every constrained node and edge centre, store a list of those above threshold,

2. sort the list in descending order of error magnitude,

3. for every site in the list: add a free node, ensure conformance by adding neighbouring subdivisions as necessary,

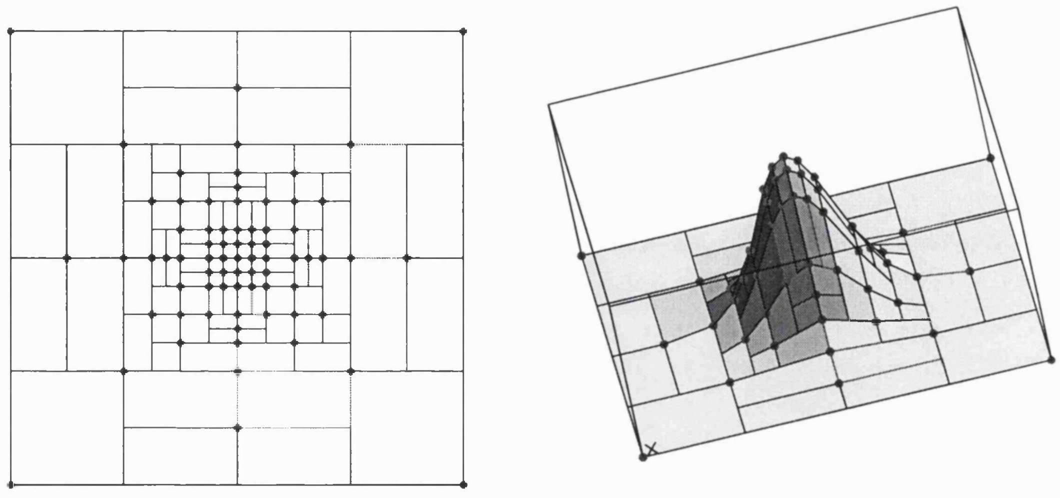


Figure 4.16: Gaussian modelled with edge centred scheme

4. repeat from step 1 until no more nodes are added.

The fact that free node sites are processed in descending order of error ensures that cells are subdivided first in the direction associated with the maximum error. The effect is similar to the method of estimating the underlying curvature used in the face and body centred meshing scheme.

The most significant difference between this meshing scheme and Gruene's is that this one abandons the 2:1 property that Gruene used. We have found that enforcing the 2:1 property produces less efficient meshes, and in practice offers little benefit. Experimental evidence presented in section 4.4.5 reinforces that viewpoint.

Figure 4.16 shows the result of applying this edge centre meshing scheme to a Gaussian function. This mesh is conformant and uses fewer free nodes than either of the meshes shown in figures 4.13 and 4.14.

Figure 4.17 shows the result of applying the edge centre meshing scheme to a top hat function. This sort of discontinuous function is of interest because of the discontinuous nature of the value functions that we wish to model. It is also important for the purpose of modelling the control function, which is often discontinuous.

Figure 4.18 illustrates the mesh obtained with the edge centred meshing scheme when applied to a Gaussian function in three dimensions.

4.4.4 Adaptive Meshing and the HJB Method

To apply adaptive meshing to the solution of HJB equations we choose to adopt Gruene's pointwise error estimate, described in section 4.3.2. This error estimate replaces the error measure-

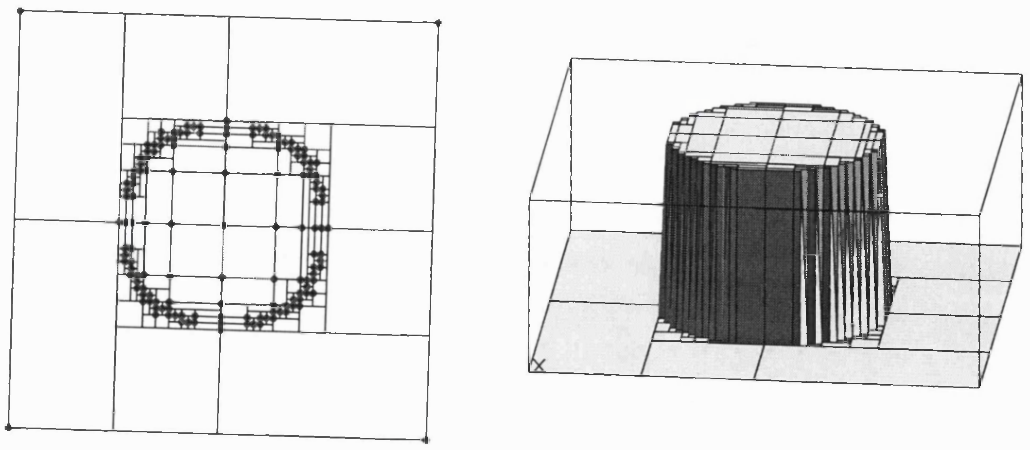


Figure 4.17: top hat function modelled with edge centred scheme

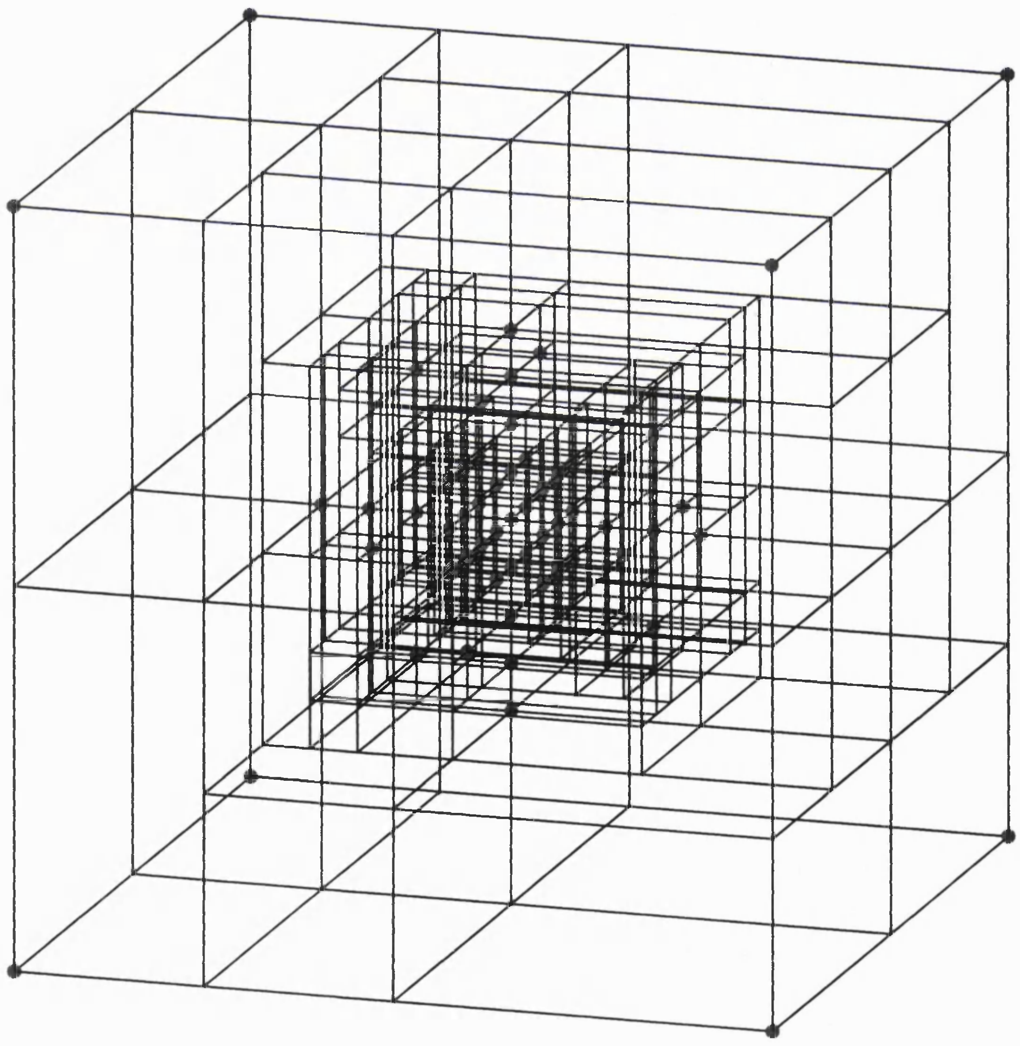


Figure 4.18: mesh for Gaussian function in 3 dimensions

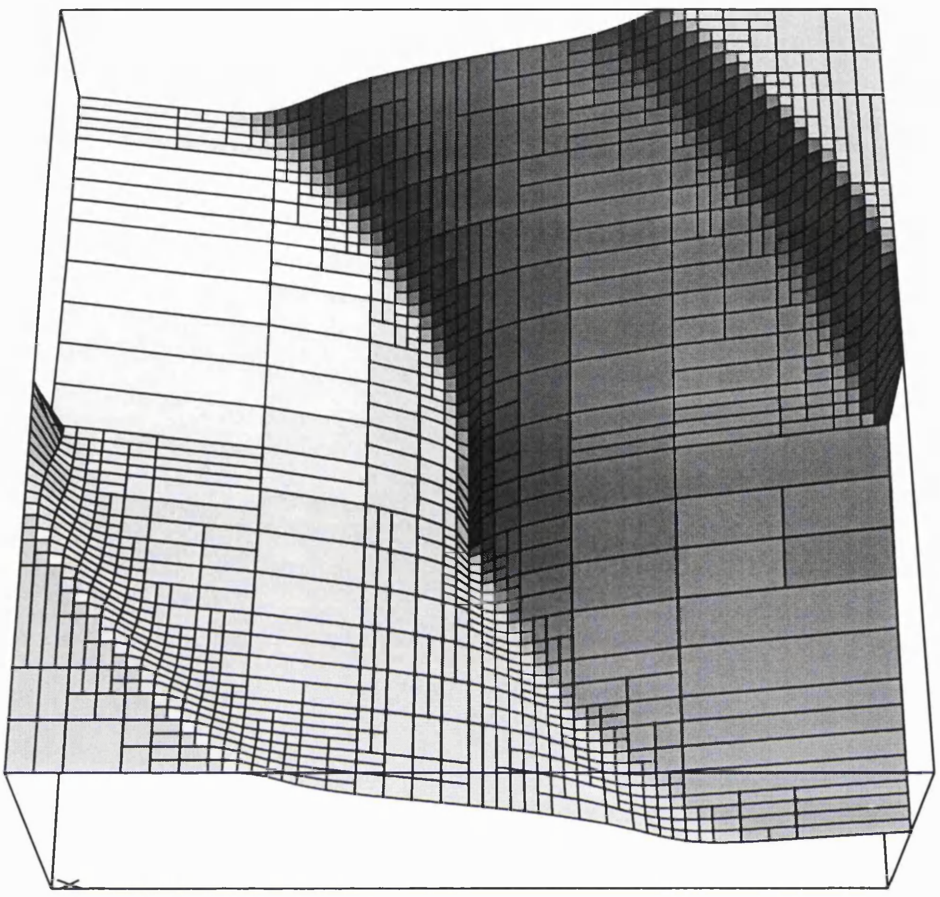


Figure 4.19: adaptive meshing applied to the Bush problem

ments that we used to explore meshing strategies. The HJB equation is first solved on an initial coarse regular mesh. The mesh is then refined once and the HJB equation is solved again using the new mesh. The whole process is repeated until the no more nodes need be added to the mesh.

Figure 4.19 shows an example of the edge centre meshing scheme applied to the Bush problem. Notice that the mesh is most refined around the switching curve and boundaries of the reachable set.

4.4.5 Meshing Scheme Measurements

To help quantify the relative performance of the meshing schemes, some variations on the meshing schemes that we have described were applied to meshing a Gaussian function and a top hat function in four dimensions, and to solving a four dimensional version of the Bush problem.

Efficiency was measured by counting the number of free nodes and leaf cells (undivided cells). Accuracy was checked by sampling the error in the mesh, or Gruene's error estimate (section 4.3.2) for the 4D Bush problem, at a million randomly chosen sites within the domain. From this, the percentage of random error samples above the subdivision threshold was calculated.

The four dimensional version of the Bush problem is formed by allowing the particle to move in two dimensions (giving a four dimensional phase space) and by allowing control to be a two dimensional vector. Since the particle dynamics are separable, this is actually equivalent to two two dimensional problems within a single domain. It is therefore not an interesting problem, per se, but it does provide a useful test of meshing performance in four dimensions.

In addition to the two meshing schemes we have already described (edge centred and face and body centred) three additional meshing schemes were tested. The first of these, which we have named *cell centric*, is identical to the face and body centred scheme described in section 4.4.1 except that the 2:1 property is not enforced and the *full split* property, whereby free nodes are added at every vertex of a subdividing hyperplane, is abandoned. In the cell centric scheme, free nodes are only added at those vertices where the error is above the threshold.

The second additional meshing scheme, which we have named *Munosesque*, was inspired by Munos' meshing scheme. The Munosesque scheme is identical to the face and body centred scheme (section 4.4.1), except that conformance is not enforced and the 2:1 property is not enforced.

The final additional meshing scheme, the *edge and body centred* scheme, is identical to the edge centred scheme except that the error was also sampled at the centre of each cell. If the error at the centre of the cell was above the subdivision threshold but the error at all the edges was below the threshold, such that the cell would otherwise have remained undivided, then this scheme ensures that the cell is subdivided and a single free node is added at the edge with the largest error.

Table 4.5 summarises the properties of our meshing schemes and those of Munos and Gruene.

Table 4.6 shows the relative performance of the meshing schemes on the three test meshing tasks.

We used these results to select the most promising meshing scheme from the ones we tested. The face and body centred scheme was discounted because it used a high number of free nodes in each test and is clearly inefficient when compared to the other schemes. The Munosesque scheme had a significantly higher number of error samples above the threshold

	anisotropic	conformant	error sampling	2:1	full split
Munos	no	no	n/a	no	yes
Gruene	yes	yes	edges	yes	no
face and body centred	yes	yes	faces & cell centre	yes	yes
cell centric	yes	yes	faces & cell centre	no	no
Munosesque	yes	no	faces & cell centre	no	yes
edge centred	yes	yes	edges	no	no
edge and body centred	yes	yes	edges & cell centre	no	no

Table 4.5: properties of the various meshing schemes

scheme	nodes	leaf cells	max depth of tree	samples > threshold
	Gaussian			
face and body centred	14351	14269	18	0%
Munosesque	2713	1136	18	1.05%
cell centric	969	6144	20	0.0001%
edge centred	2059	10600	24	0%
edge and body centred	1026	6108	32	0%
	top hat			
face and body centred	79242	82032	20	1.1722%
Munosesque	50217	29008	20	1.1761%
cell centric	11745	29634	20	1.1578%
edge centred	9338	26341	20	1.1547%
edge and body centred	9338	26341	20	1.1585%
	4D Bush problem			
face and body centred	11071	10568	17	0.0068%
Munosesque	1273	288	15	1.9869%
cell centric	3283	23160	38	0.5514%
edge centred	761	6119	30	0.0817%
edge and body centred	59705	29853	44	0.0811%

Table 4.6: meshing scheme efficiency and accuracy

Gaussian: standard deviation 0.3, height 1 at origin, error threshold 0.05

top hat: $g(x) = \begin{cases} 1 & \|x\| \leq 1 \\ 0 & \|x\| > 1 \end{cases}$, depth of tree limited to 20

4D Bush problem: depth of tree limited to 44, error threshold 0.4

with the Gaussian and 4D Bush problem than the other schemes. This casts doubt upon the accuracy and reliability of the Munosesque scheme. The edge and body centred scheme used nearly 80 times as many free nodes on the 4D Bush problem as the edge centred scheme without any significant improvement in accuracy (as measured by the number of error samples above the subdivision threshold). Therefore, the choice boils down to the edge centred or cell centric scheme.

Based upon the results from the Gaussian and top hat function there is little to choose between the edge centred and cell centric scheme. However, the edge centred scheme provided better performance and better accuracy on the 4D Bush problem. We therefore selected the edge centred scheme; and this is the meshing scheme that is used in subsequent chapters wherever adaptive meshing is employed.

4.5 Summary

In this chapter we first discussed several methods for accelerating the convergence of the HJB method. The following list summarises our findings.

1. Rapid convergence can be obtained, even when the timestep is small, by treating the local iteration as an infinite series and finding its limit.

2. A modest improvement in the convergence rate can be obtained by processing the nodes in increasing order of value.

3. Another modest improvement in the convergence rate can be obtained by separating the process of finding the optimal control (the control pass) from the process of updating the value function (the value pass).

4. A further reduction in the number of iterations is possible by performing several value passes for each control pass. A heuristic rule can be applied to choosing the number of additional value passes.

Taken together these provide a very substantial reduction in the amount of processing required, summarised in table 4.7. For the particular benchmark problem that we used, the original naive implementation required 422 iterations. That was reduced to 13 iterations with 71 value passes.

The question of how the boundaries of a finite domain affect the solution was then discussed. The solution tends to become distorted near the boundary of the reachable set. We proposed the membership function to quantify the magnitude and extent of the distortion. The membership function reveals that such distortion rapidly falls to zero within the reachable set.

algorithm	iterations
3.2	422
4.1	56
4.2	48
4.3	13 (71)

Table 4.7: Summary of convergence results

It can also be used to provide some idea of how large the effective reachable set obtained with a particular solution may be. Thus the membership function can be used as a quick check of the progress of the solution procedure.

We argued for the adoption of an adaptive meshing scheme based upon a kd-tree decomposition of the domain. Related work was described and assessed. We choose a conformant, anisotropic, kd-tree based, meshing scheme that works by estimating the error in the solution at the centre of each edge. The performance and accuracy of this scheme was compared with an earlier scheme and with several other possibilities motivated by the methods used in the literature.

Chapter 5

Building a Controller

The previous chapter described the design of a practical algorithm for solving the HJB equation. The algorithm was designed to provide a good approximation to the value function, without regard to the control function. As it happens, the algorithm also yields the optimal controls at the mesh nodes, and hence an approximation to the optimal feedback controller can be obtained by applying the same interpolation methods used for the value function. However, as we shall see, this control function may not be satisfactory and additional care must be taken to ensure the quality of the control function. Ultimately, it is the control function that is important, the value function is merely a useful tool for obtaining the control function.

This chapter first addresses the question of how to measure the performance of control functions obtained by the HJB method, particularly when there is no known analytic solution. Such measurements help us to make informed choices about the design of the algorithm. Alternatives for the construction of the control function are described and assessed. The choice of subdivision criteria is then examined and alternatives are suggested and assessed in terms of their effect upon controller quality.

5.1 Controller Assessment

There are essentially three questions to be answered when assessing a controller:

1. Does it work?

2. How well does it perform?

3. Does the resulting motion look right?

The first is simply an assessment of whether the controller performs its intended task in an acceptable fashion. For example, if the task is to balance a pendulum upright, we might define an acceptable balancing behaviour as one that remains within some tolerance of upright for a certain period. The assessment can be made by running a simulation of the pendulum with the controller and checking whether the conditions are satisfied.

To answer the second question we must measure controller performance. The natural way to do this comes from the definition of the HJB equation in terms of the integral of a cost function along a trajectory (equation 3.15). Hence, for any given starting point in phase space, we can measure controller performance by running a simulation and integrating the cost function along the resulting trajectory. This measurement provides the basis for a fair comparison between different controllers because it measures the same quantity that the HJB method is designed to optimise.

Of course, a single starting point can only be used to measure controller performance over a small part of the domain, i.e. along the resulting trajectory. To provide reasonable coverage of the whole domain many starting points must be used. We choose to provide this coverage with a large set of starting points distributed *uniformly* and *randomly* throughout the domain. The overall performance measure is then the sum of the individual performance measurements at all the starting points. The uniform distribution of starting points ensures that equal weight is given to the performance of the controller for trajectories that start in all parts of the domain. The starting points are distributed randomly to preclude the possibility that a regular spacing of starting points might interact in some way with the structure of the mesh.

A complication arises because the trajectories from some starting points will inevitably leave the domain. These starting points are outside the reachable set of the controller. In these cases the performance measure does not provide any useful information. Therefore a subset of starting points must be found that is inside the reachable sets of all the controllers to be compared, i.e. the intersection of the reachable sets. The comparison can then be made by summing the performance measure over the subset of starting points.

By counting the number of starting points outside the reachable set we obtain an approximate measurement of the size of the effective reachable set. This, in itself, is another useful measurement of controller quality. For example, if the purpose of the controller is to balance an unstable system, then a larger reachable set provides greater stability for the controlled system. Hence, the process of measuring performance provides two distinct ways to compare controllers.

The third and final question that we would like to ask about controller performance concerns an observer's perception of the resulting motion. Our original intention was to produce lifelike motion, therefore we would like to make a higher level assessment based upon whether the motion fulfils that goal. We have made no attempt in this work to quantify the degree to which the motion appears lifelike, nevertheless a subjective judgement may be made by viewing an animation.

5.1.1 The Modified Bush Problem

We shall, once again, use the Bush problem to illustrate some general principles that arise from the consideration of controller quality. Later, in chapter 6, we will apply the lessons learnt from this simple problem to solving more complex problems.

For the Bush problem, which has an explicit target set, we might choose to assess the controller by running simulations until the target set is reached. Since it is a minimum time controller, the time taken to reach the target set is the performance measure. Of course, as the target set is infinitesimally small, we must use a numerical tolerance, ϵ, to determine when the target set has been reached. Herein lies a problem: the choice of ϵ turns out to have a significant and unpredictable affect on measured performance.

One reason why ϵ affects measured performance is that many trajectories overshoot the target set before coming back to it from the opposite side. The tendency to overshoot is a consequence of numerical error in the control function. Some trajectories will get very close to, or even inside, the target set before overshooting. Consequently, a tiny change to a trajectory can determine whether or not the overshoot is included in the performance measure and hence can make a significant difference to the final measurement.

We require a performance measure that does not make such unpredictable changes with small variations in the corresponding trajectory. The Bush problem does not produce the desired continuity when its performance is measured. However, continuity can be achieved by modifying the Bush problem slightly.

If we use a discount factor we can avoid the need to make a decision about when the target set has been reached. Any overshoot will then always be counted in the performance measurement. The discount factor allows us to determine a finite horizon beyond which future costs can always be neglected. For example, with a discount factor, λ, of 0.5 we only need to run each simulation for around 20 seconds because we know that

$$\int_{t=20}^{t=\infty} e^{-\lambda t} dt \simeq 5 \times 10^{-5} \int_{t=0}^{t=\infty} e^{-\lambda t} dt \tag{5.1}$$

Furthermore, if we use a continuous cost function, such as

$$l(x, a) = |p|, \qquad x = (p, \dot{p}) \tag{5.2}$$

then we avoid the step change in the cost function at the target set, and we no longer require an arbitrary tolerance ϵ.

domain	$[-1,1] \times [-1,1]$
maximum resolution	65×65
available controls	$\{-1,0,1\}$
convergence threshold	0.001
timestep, h	0.01
boundary value	2
discount factor, λ	0.5
cost function	$l(x,a) = \lvert p \rvert, \quad x = (p,\dot{p})$
subdivision threshold	$\frac{h}{10}$

Table 5.1: parameters for the modified Bush problem

The use of a continuous cost function has a second important advantage. When the system dynamics and the cost function are continuous, the value function will be differentiable along any optimal trajectory. Consequently, the value function will, in general, be smoother and easier to approximate with a piecewise linear, or multilinear, function.

The use of a discount factor and a continuous cost function together ensure that measured performance changes gradually with changes to the trajectory. This permits a fair comparison to be made between different controllers based solely upon the parameters of the original problem specification, without the need for an arbitrary numerical tolerance.

5.1.2 Performance Results

The modified Bush problem, as described above, was solved on a regular mesh of 65x65 nodes. Table 5.1 shows the parameters used for this problem. The performance of the controller was then measured using 1000 randomly chosen start points. For the simulation runs, an Euler integrator was used with the same timestep that was utilised for solving the HJB equation. Of the 1000 starting points 99 starting points resulted in trajectories that left the domain, i.e. 99 starting points were outside the reachable set. The total performance score for the remaining 901 starting points was 323.0.

For the sake of comparison, the performance measurement was repeated using the optimal control function. This control function was obtained from the analytic solution of the Bush problem, but modified to be optimal for the finite domain in use, i.e.

$$a^{mod}(x) = \begin{cases} 0 & x = ([-1,-0.5],1),([0.5,1],-1) \\ a^{bush}(x) & \text{otherwise} \end{cases}$$

where $a^{bush}(x)$ is the optimal control function in an infinite domain. With this control function,

controller, integrator	performance*	failures†
numerical, Euler	323.0	99
numerical, Runge Kutta	319.8	98
analytic, Euler	315.5	96
analytic, Runge Kutta	311.5	95

Table 5.2: controller performance on modified Bush problem

*total performance score summed over 901 commonly successful start points
†number of start points from which trajectory left domain

96 starting points were outside the reachable set, and the total performance score, with the same 901 starting points as before, was 315.5.

These results indicate that the performance of the controller found by solving the HJB equation (numerical controller) is close to that obtained analytically. However, it is conceivable that the performance of the numerical controller only appears to be close to the analytic controller because it was optimised using the Euler integration method, which tends to introduce consistent integration error. Hence the controller should perform best on a system that includes the same integration error. The analytic controller is based on the assumption of zero integration error, and hence will not be perfect when applied to an Euler integrator.

To check the effect of integration error upon these results, the performance measurements were repeated using a fifth/sixth order Runge Kutta method [76] that is far more accurate than the Euler method. Table 5.2 summarises the results.

The performance of both the numerically generated and analytic controllers improved with the Runge-Kutta integrator. The analytic controller improved slightly more than the numerical controller. We can conclude that the performance of the numerically generated controller is indeed close to the analytic one, and the close performance is not merely a coincidental consequence of integration error.

The question that remains is why the analytic solution provided a slightly larger reachable set than the numerical solution. We shall look at the reasons for this next and propose a modification that will improve the reachable set of the numerical controller.

5.1.3 Control Function Extrapolation

If we look at the numerically generated control function for the modified Bush problem, figure 5.1, we see a ridge along the edge segment $\{p = 1, \dot{p} = [0, 1]\}$. This should not be surprising because all trajectories from these nodes lead outside the domain. Hence, the available controls all appear to have equal merit and the optimal control cannot be determined. In this case the

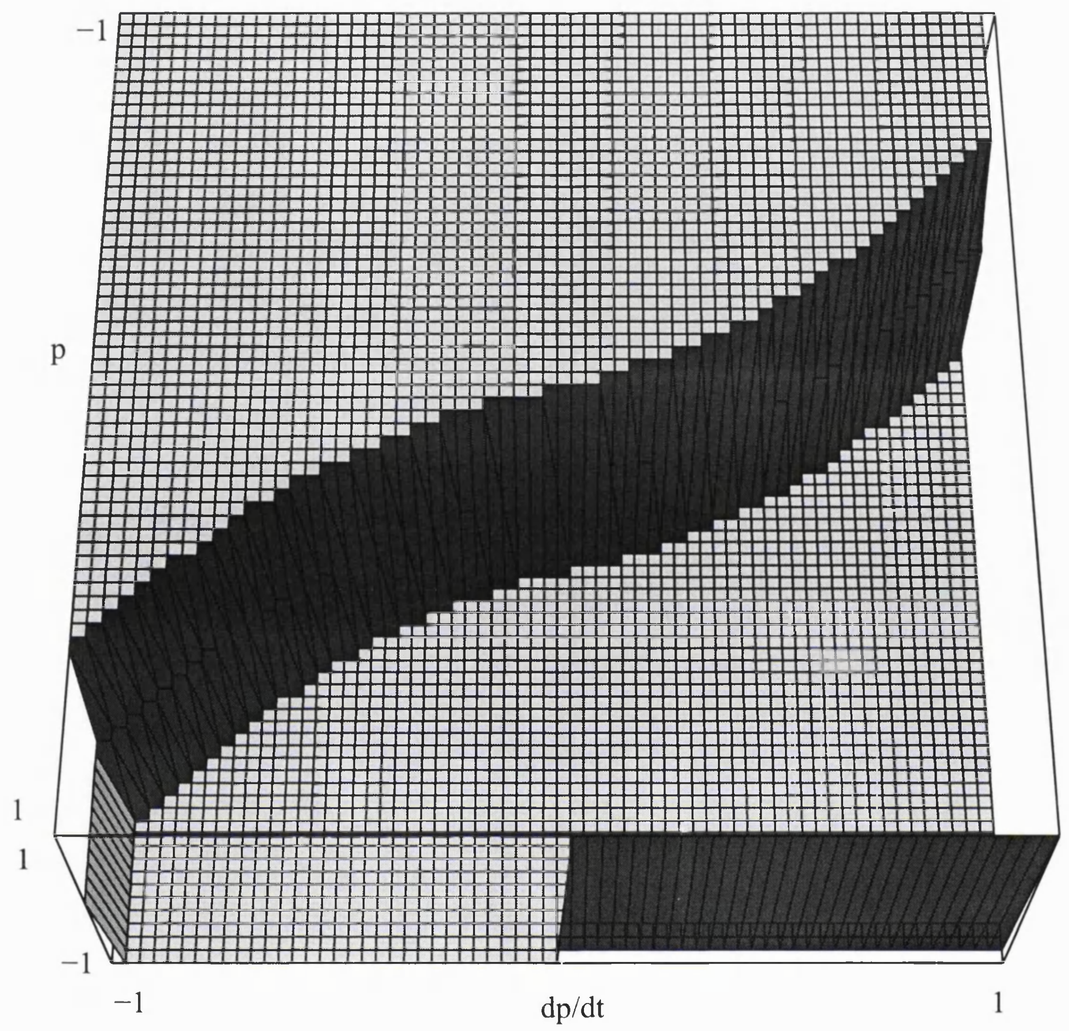


Figure 5.1: control function from modified Bush problem

HJB method has arbitrarily chosen a control that stands out from the control function in the surrounding domain.

Because control is interpolated, the arbitrary control along the edge segment extends some way into the domain. Consequently, a small number of trajectories that pass very close to the edge of the domain (near $(1, 0)$) receive the wrong control input and end up leaving the domain. Hence the reachable set is slightly reduced by the presence of this ridge in the control function.

In fact, control may be indeterminate wherever the value function is locally flat. This can occur at any point where the membership function is zero (not necessarily on the boundary) because the value of nodes with zero membership is always identical to the boundary value. It just happens, in this example, that control is only indeterminate along the boundary.

To avoid an unnecessary reduction of the reachable set we would like to arrange that nodes with indeterminate control should not disrupt the control function in adjacent cells. Therefore, we have chosen to extrapolate the control function from regions where the optimal control may be determined, into regions where it is indeterminate.

The method we employ to achieve this is based upon the notion that the control should not change *along an optimal trajectory* unless the change is well determined. Using this idea we may assign a control to a node, where it is otherwise indeterminate, by checking where a trajectory must have come from in order to arrive at this node. We integrate the equations of motion backwards in time, using an Euler step. The value of the control function at the end point of that step is then assigned to the node. With each iteration of this process, the control function propagates, from inside the reachable set, further into regions outside the reachable set.

The update rule, $W(a_n)$, for control at an indeterminate node, n, may be written as

$$W(a_n) = a(x_n - hf(x_n, a_n)).$$

After implementing control extrapolation, the modified Bush problem was solved again with the result that the ridge in the control function disappeared, i.e. the control function along that edge segment was -1, the same as the control function in the surrounding domain. When the performance of this new controller was tested, the reachable set was found to be identical to the analytic solution, and measured performance was identical to that given in table 5.2.

In several other experiments, such as those to be discussed later, we have also found that this method of control function extrapolation is an effective way to improve the reachable set.

5.2 Adaptive Mesh Controllers

The previous chapter looked at the application of adaptive meshing to solving the HJB equation. The mesh was refined where the value function was most curved in order to provide a more efficient mesh (i.e. fewer nodes) for solving the equation. However, we have yet to examine the control function resulting from this process.

We now examine the control function generated on an adaptive mesh for the modified Bush problem. It will be shown that the controller obtained by interpolation on an adaptive mesh may be of poor quality. We shall propose and assess a number of measures to improve the quality of controllers obtained with adaptive meshing.

5.2.1 Interpolated Controller

The modified Bush problem was solved using adaptive meshing, with the parameters already given in table 5.1. The performance of the resulting controller was then measured using Euler integration and the same 901 starting points that were used in table 5.2. The performance score was found to be 329.5. This compares to the figure of 323.0 obtained with the regular mesh (lower is better). However, the adaptive mesh used only 1283 free nodes whereas the regular mesh used 4225 free nodes.

Note that the performance score of 323.0 may be regarded as the best score possible since we have chosen to limit the available resolution to 65x65 nodes, the resolution of the original regular mesh. This allows us to compare the performance of one adaptive mesh controller to another, not just in absolute terms, but also in terms of how close they get to the best possible performance that can be obtained on a mesh of given maximum resolution.

Figure 5.2 shows the control function obtained with adaptive meshing. Visually, this appears to be significantly worse than the function obtained with the full regular mesh, figure 5.1. The switching curve is crudely approximated, especially at the ends of the curve. It is apparent that the regions of high mesh density do not exactly correspond with the location of the switching curve.

It is perhaps not surprising that the control function should be more crudely approximated than the value function, since no account is taken of the control function in the mesh refinement process.

To enhance the performance of this adaptive mesh controller we could simply use a lower subdivision threshold, but that will act mainly to improve the quality of the value function and will not necessarily provide a much improved control function. The essence of the problem is that the switching curve lies in a region of the value function that has been approximated by a relatively coarse mesh. Interpolating the control function across a coarse mesh does not provide an accurate control function. Therefore we should either change the subdivision criterion so as to refine the mesh at the switching curve, or we should use a method other than interpolation to get the optimal control at any point within the domain.

This second idea, to get the optimal control at a point without interpolating the optimal control at the nodes, is the idea that we shall look at next.

5.2.2 Recalculated Controller

Given perfect knowledge of the value function, it is possible to get the optimal control at any point within the domain by repeating the search process, i.e. by testing each of the available control inputs to determine which is best. Therefore, if the value function is sufficiently well approximated, it should be possible to get a better control function by searching again for (recalculating) the optimal control at every point where it is required, e.g. for running simulations.

The obvious disadvantage of recalculating the optimal control is that it requires more com-

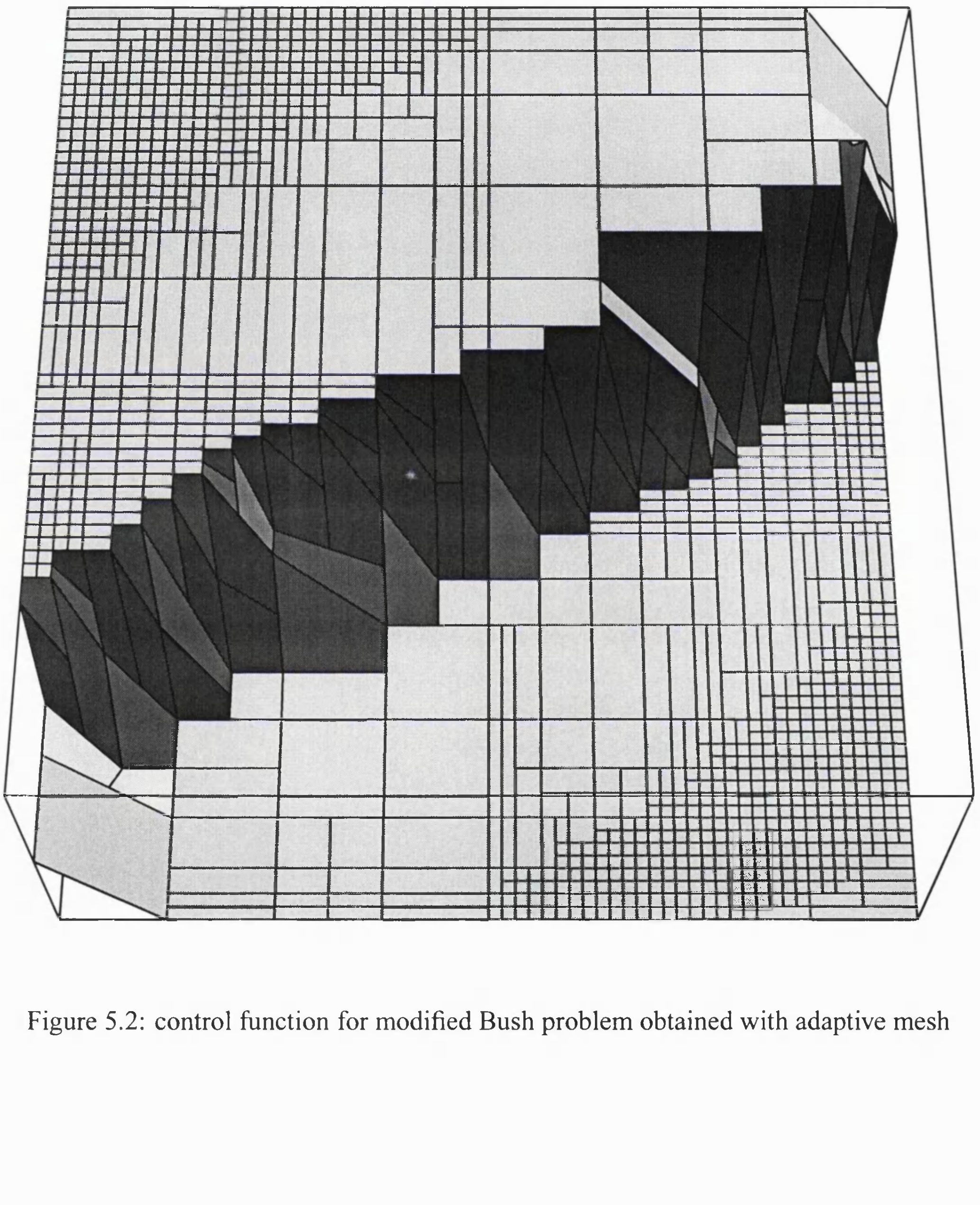


Figure 5.2: control function for modified Bush problem obtained with adaptive mesh

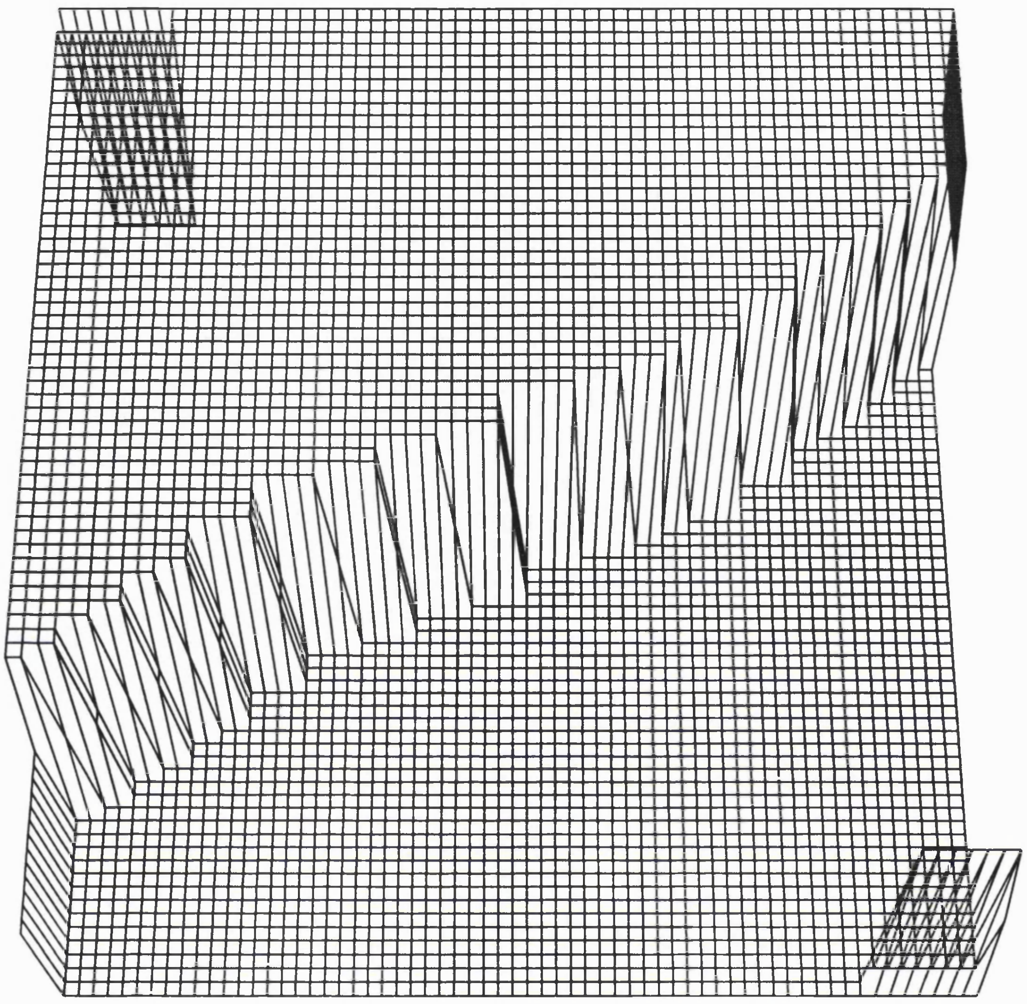


Figure 5.3: Control function found by recalculating the control on an adaptive mesh

putation than does interpolation. How much additional computation is required depends upon the complexity of the search. If the search is complicated the additional cost may be prohibitive. For the Bush problem, however, only three controls are tested $\{-1, 0, 1\}$ and the additional cost is small.

Figure 5.3 shows the control function obtained by recalculating the optimal control at every point on a regular mesh, using the adaptive mesh value function shown in figure 5.2.

The performance of this, 'recalculated' controller was measured at 348.5, using the same 901 starting points as in table 5.2.

The switching curve in figure 5.3 appears slightly cleaner than the one obtained by interpolation (figure 5.2) especially near the ends of the curve where the mesh is less dense. However, some problems remain where the function does not quite follow the desired curve. Moreover, the performance of this control function is considerably worse than the interpolated controller (which was 329.5).

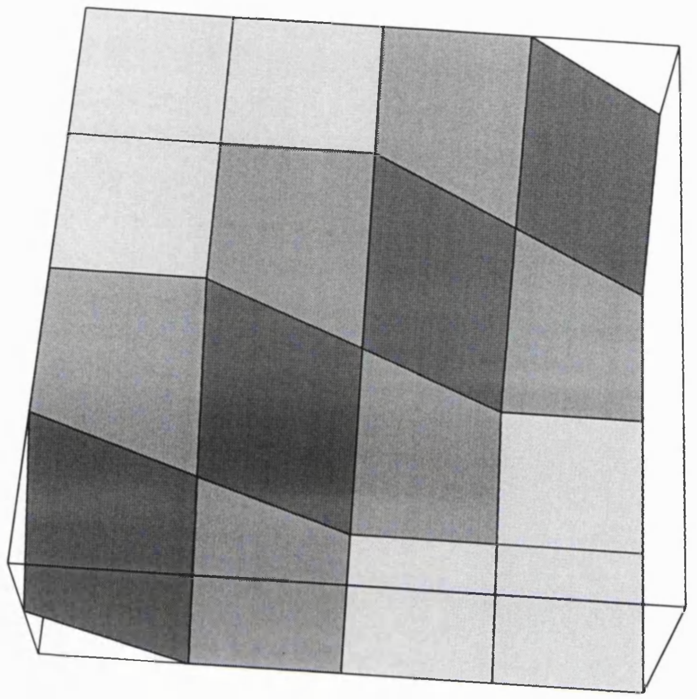
As a further comparison, the value function generated on the full regular mesh (in section 5.1.1) was also used for a performance assessment with recalculated control. The performance score this time was 333.8, also significantly worse than the performance obtained with interpolated control on the same mesh (323.0).

An explanation for these results can be found by comparing the control functions obtained by interpolation and recalculation on a very coarse mesh. Figure 5.4 shows these control functions for the Bush problem solved on a 5x5 mesh. The interpolated controller approximates a proportional-derivative (PD) controller in a region around the origin. The recalculated controller, however, exhibits an artificial switching curve that runs along the cell boundaries and is caused by the discontinuity in the gradient of the value function at those cell boundaries. The interpolated controller will control the system to its target even at scales much less than the cell size. By contrast, the recalculated controller may fail to bring the system closer than a single cell width of the target.

Recalculated control has the potential to provide better performance, if the value function is sufficiently well approximated. However, it can fail badly when there is significant gradient discontinuity between adjacent cells or where proportional control at sub-cell scales is required. It seems unlikely that we will get a sufficiently good approximation of the value function in practice to benefit from recalculated control. Therefore we conclude that control interpolation is the preferred method.

5.3 Subdivision Criteria

Another way to go about improving the quality of the control function is to try different criteria for subdividing the mesh. So far, an estimate of the pointwise error in the value function has been used. This produces a good approximation to the value function, but not necessarily a good approximation to the control function. Perhaps if the error in the control function were considered as well then it would be possible also to obtain a good approximation to the control function. An estimate of the control error can be obtained in exactly the same way as for the value function - by searching for the optimal control at a point and comparing that to the control obtained by interpolation on the mesh. Since control may be a vector, we use the magnitude of the difference of the two vectors as the criterion. Figure 5.5 shows the result obtained by applying this criterion. The control function in figure 5.5 is less 'blocky' than the one in figure 5.2 but the switching curve is obviously not in the right place (e.g compared with figure 5.1). This can be explained by the fact that the value function has been poorly



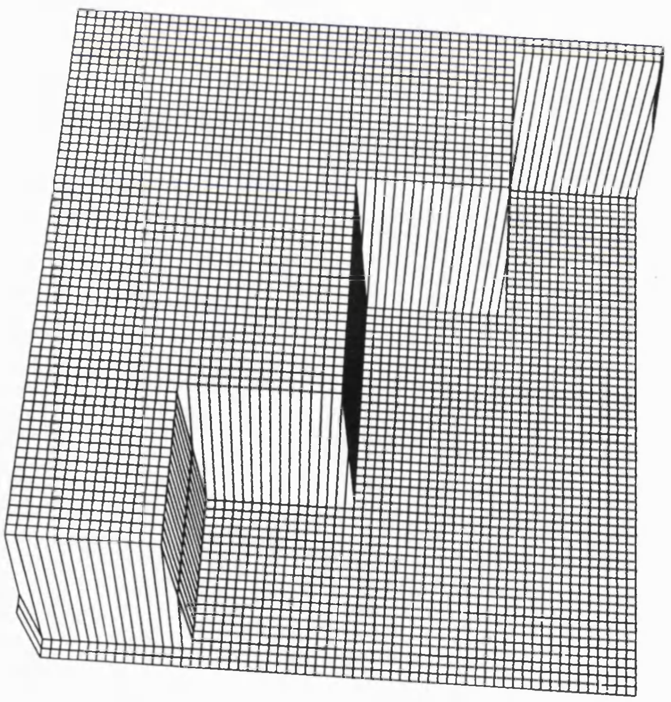


Figure 5.4: interpolated versus recalculated control with a very coarse mesh

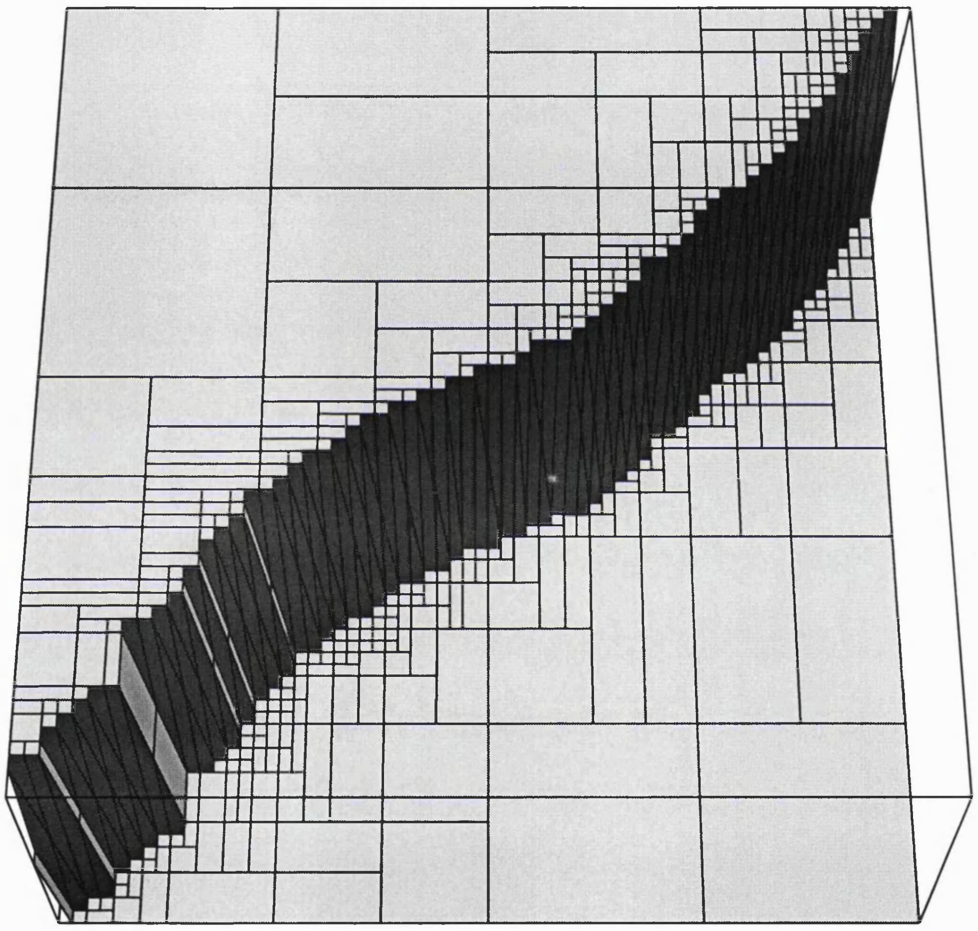


Figure 5.5: control function for the Bush problem obtained by subdividing on control error

approximated with this mesh, even though there is high mesh density around the switching curve. Without a reasonable approximation of the value function it is not possible for the numerical procedure to locate the switching curve accurately.

The performance measurement obtained with this controller was 355.9 (with 468 free nodes), considerably worse than the performance measurement of 329.5 obtained by subdividing on value function error (figure 5.2). These measurements emphasise that control error alone is not a good subdivision criterion.

The obvious improvement is to subdivide the mesh where either the error in the value function exceeds a threshold or the error in the control function exceeds a threshold. This subdivision criterion can be expressed as

$$S = \max\{e_v + ke_c\} \tag{5.3}$$

where e_v is the estimate of the error in the value function, e_c is the control error estimate and k is some constant. In fact for these bang-bang problems the value of k is not critical, since e_c is generally either 0 or some significant value, k can be chosen such that any non-zero e_c will trigger subdivision.

When the problem does not exhibit bang-bang control we would like to choose k so as to provide a good balance between value function accuracy and control function accuracy. Further work may be required to find a useful method for choosing k in these cases.

Figure 5.6 shows the result obtained with the combined subdivision criterion. The switching curve is now much closer to that shown in figure 5.1, but at the cost of using more free nodes. The performance measured with this controller was 328.6 (with 1935 free nodes). Just slightly better than the figure of 329.5 obtained by subdividing on the value function error alone.

Observation of the solution progressing shows that the position of the switching curve moves as the mesh is refined. Early subdivision triggered by the switching curve can therefore be wasted as it is often in the wrong place. Therefore it should be possible to reduce the number of nodes required by delaying subdivision based upon control function error until a good approximation of the value function is obtained.

Figure 5.7 shows the result of subdividing on value function error, until the limit of subdivision is reached, and then using the combined subdivision criterion, equation 5.3.

This control function appears similar to the previous one, but with fewer free nodes. The performance measurement this time was 329.5 with 1571 free nodes. A slight reduction in performance for a useful reduction in the number of nodes.

Surprisingly, the performance of this last controller is identical to that obtained by sub-

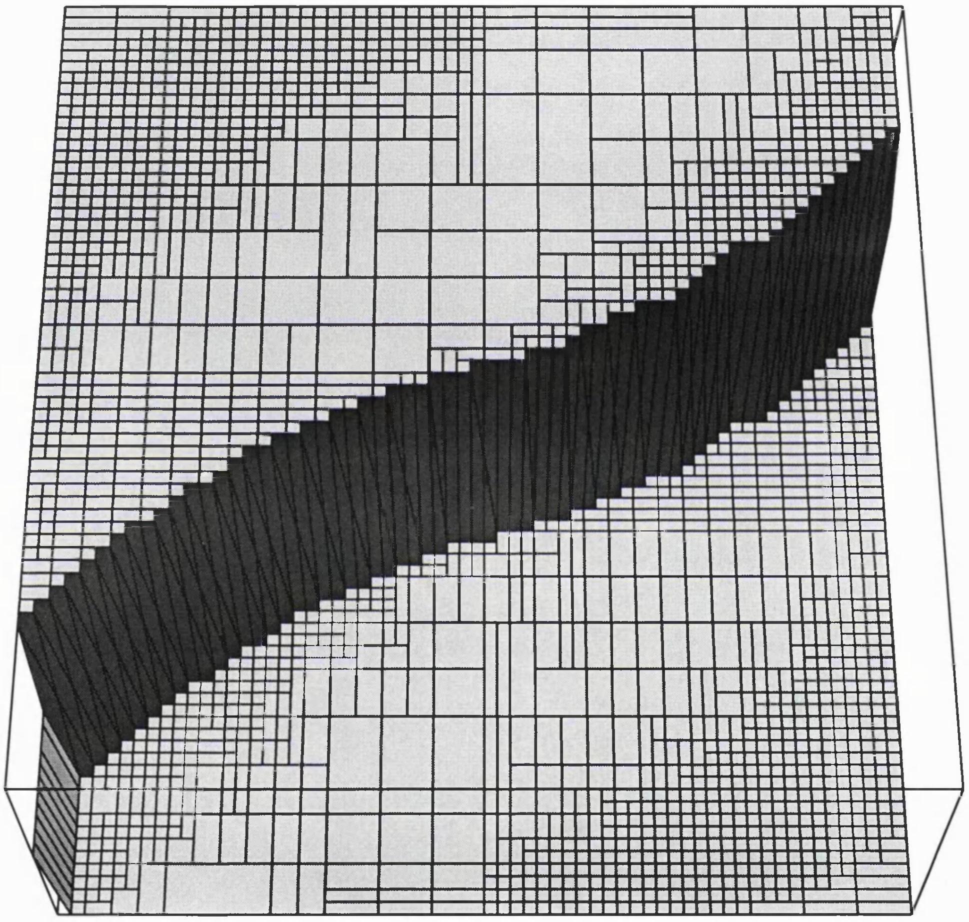


Figure 5.6: subdivision based on value and control function

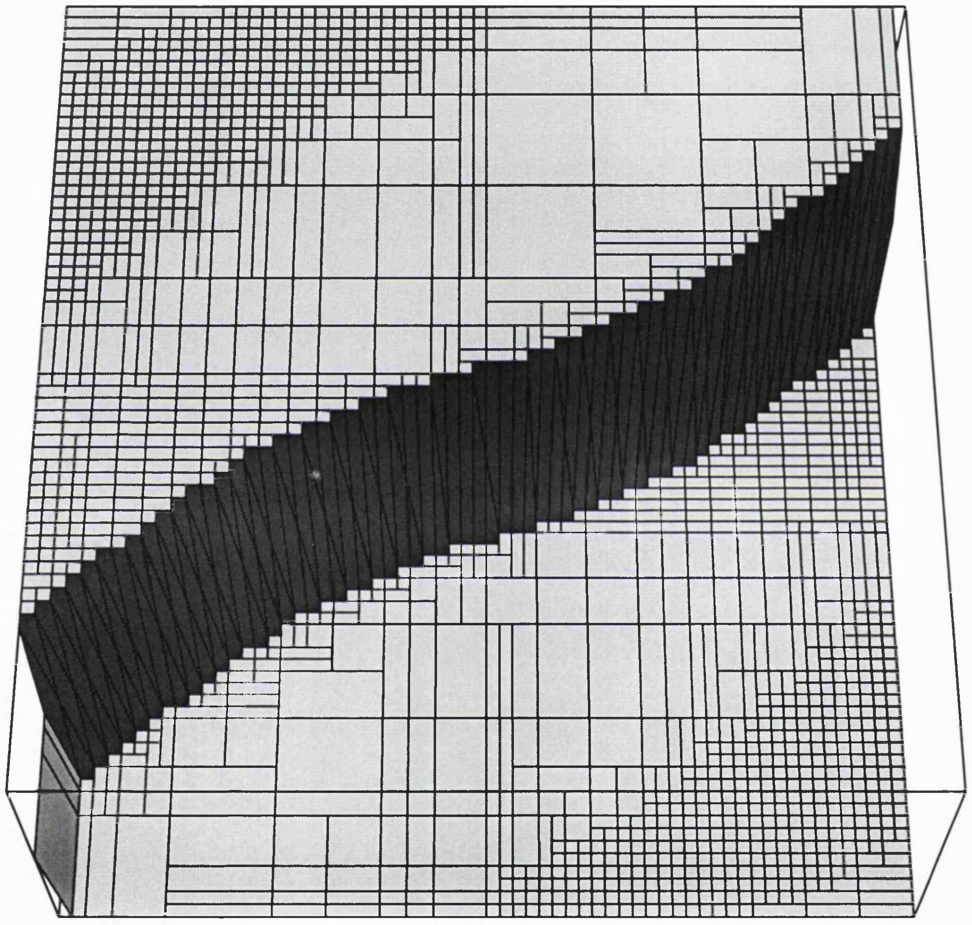


Figure 5.7: subdivision based on value - then on value and control

	performance	free nodes
65x65 full resolution	323.0	4225
subdivide on value (65x65 max)	329.5	1283
subdivide on value and control (65x65max)	328.6	1935
subdivide on value *then* control (65x65 max)	329.5	1571
129x129 full resolution	318.6	16641
subdivide on value (129x129 max)	325.7	2618
subdivide on value *then* control (129x129 max)	329.8	3321

Table 5.3: performance results

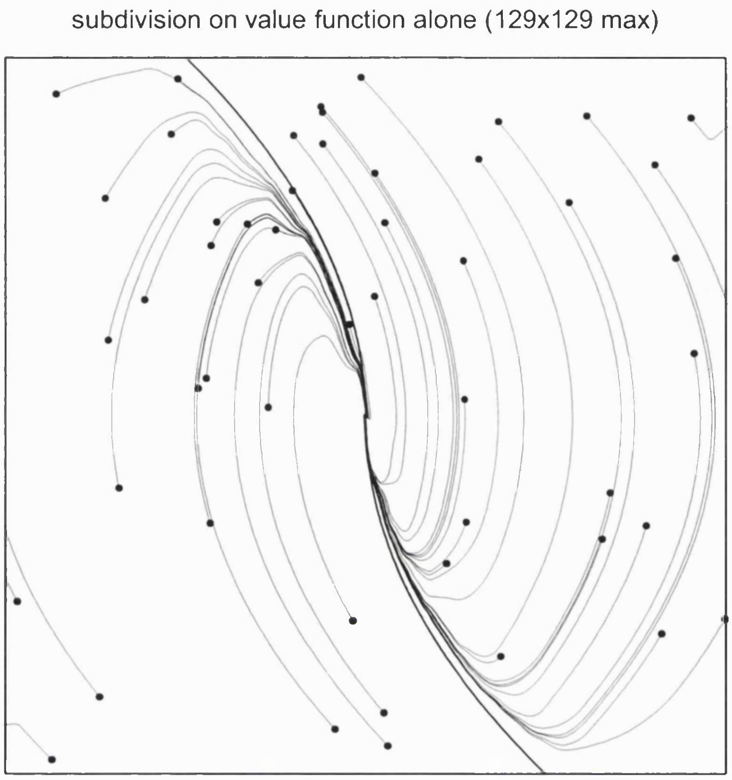
dividing on value function error alone, despite the fact that more free nodes have been used, and that those free nodes have been placed in the critical region around the switching curve. Apparently nothing has been gained by refining the mesh around the switching curve.

To investigate this phenomenon further some more performance measurements were taken using solutions generated on a mesh with twice the maximum resolution (129x129 nodes). Table 5.3 summarises these results.

With the increased resolution, the process of refining the mesh around the switching curve (subdividing on value function error then on control function error) actually *reduced* controller performance when compared to the controller obtained by subdividing on value function error alone. This is the opposite of the result we would expect to see; adding nodes to the mesh should improve performance.

This surprising result can be understood better by comparing a set of trajectories generated with subdivision on control function error to a set generated without such subdivision. Without subdivision on control function error, control switches gradually from maximum acceleration to maximum deceleration. The result is a curved trajectory in phase space (upper diagram in figure 5.8). However, with subdivision on control function error, control switches quickly from maximum acceleration to maximum deceleration, the trajectories change direction abruptly (lower diagram in figure 5.8) and maximum deceleration is applied slightly earlier. The process of subdividing on control function error has defined an effective switching curve that is slightly conservative, due to inevitable numerical error, and the consequence is a reduced performance score.

It is likely that the unexpected reduction in controller performance is, at least partly, a coincidence. The extra time required for control to switch to maximum deceleration, without subdivision on control function error, just happens to be enough to compensate for the fact that



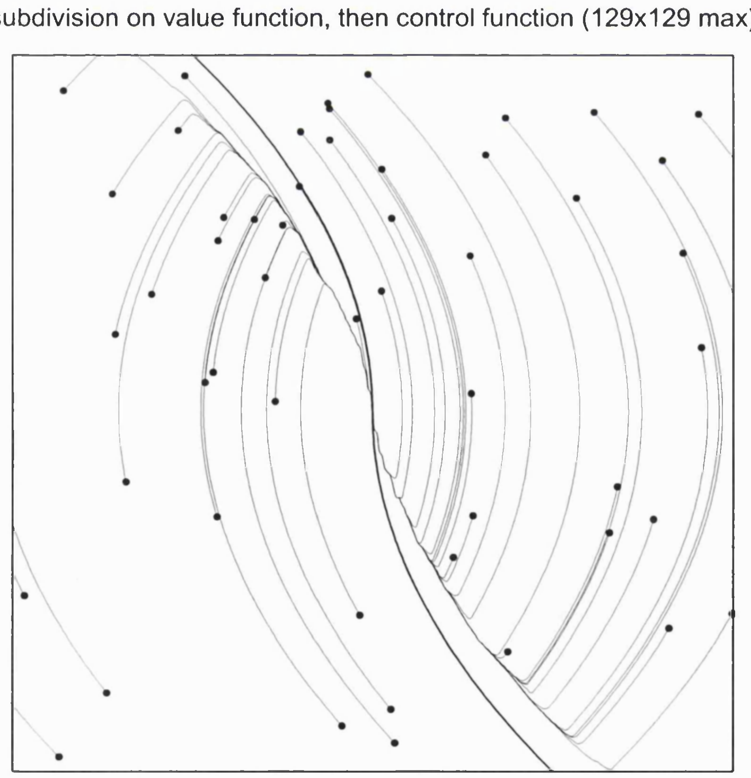


Figure 5.8: trajectories obtained from solutions to the modified Bush problem

The optimum switching curve is shown as a thicker line than the numerically generated trajectories.

maximum deceleration would otherwise have been applied too soon. In tests conducted with another problem (the cart and pole example described in section 6.1.5), the expected result was obtained, i.e. subdivision on control function error did improve performance.

5.3.1 Influence

One outstanding problem with the subdivision criteria described so far is the massive subdivision that occurs at the boundary of the reachable set. In fact we do not require such a detailed solution in this region since the control function is unchanging here and because the accuracy of the solution here has little effect on the location of the switching curve.

Ultimately, it is the control function that is important. The value function is not an end in itself, but is only useful in so far as it affects the control function. It is a fact that the accuracy of the value function has a more significant impact upon the control function in some regions than in others. Therefore it is possible that a more efficient mesh could be achieved by concentrating the mesh in the most important regions of the value function.

Munos and Moore introduced a method that achieves this, by identifying regions of changing control and calculating the *influence* of every other part of the domain upon these regions [60][63].

Let us define the *direct influence*, $\tilde{I}_n(x_j)$, as the extent to which a change in value of node m will alter the value of another node, n, i.e.

$$\tilde{I}_n(x_m) = \frac{\partial u(x_n)}{\partial u(x_m)}. \tag{5.4}$$

If we let y_i be the end-point of the integration step taken from node i then the approximate value function at node i, $u(x_i)$, is obtained from the update rule

$$T(u(x_i)) = hl(x_i, a_i) + (1 - \lambda h) \sum_{j=1}^{L} w_j(y_i) u(x_j), \tag{5.5}$$

where L is the total number of nodes in the mesh and $w_j(x)$ are the interpolation weights associated with some point, x, in the domain (most are 0) and $\sum_{j=1}^{L} w_j(x) = 1 \ \forall x \in \Omega$. Hence, from 5.4 and 5.5 we get

$$\tilde{I}_n(x_m) = (1 - \lambda h) w_m(y_n). \tag{5.6}$$

However, this quantity is not particularly useful because it does not take account of the way that a change in the value of one node may propagate throughout the mesh as the solution evolves. We therefore define the *influence* of node m upon node n, $I_n(x_m)$ as the sum of its *direct influence* on other nodes multiplied by their *influence* on node n, i.e.

$$I_n(x_m) = (1 - \lambda h) \sum_{j=1}^{L} w_m(y_j) I_n(x_j). \tag{5.7}$$

Proper calculation of $I_n(x_m)$ must now be performed in an iterative fashion, much as the value or membership functions are calculated, until it has converged at all nodes within the mesh. After convergence, the value of $I_n()$ measures the influence of every other node upon node n.

Computation of the influence presents some new problems that we have not previously encountered. Firstly, the summation is different from that used to update the value function. It does not necessarily sum to one, and requires that the influence variable at each node be treated as an accumulator; as each node is processed some of its influence is assigned (accumulated) to neighbouring nodes. Thus the influence of any one node is not necessarily known until all other nodes have been processed. Consequently, it is necessary to store two influence functions, one from the previous iteration, and another to act as an accumulator.

Secondly, something special must be done to handle constrained nodes. Influence cannot be assigned directly to constrained nodes but must instead be divided up between the parents of each constrained node if we are to preserve continuity within the mesh.

The third problem relates to step size. The formulation we have described can only work with integration step sizes that are sufficiently large so that $w_i(y_i) = 0$. The 'step to edge of cell' policy used by Munos et al ensures that this is always true. Our implementation, however, may use smaller step sizes, which would mean some of the influence for node i must be assigned to itself and therefore less to the other nodes. Step size therefore directly affects the determination of influence in a way in which it does not affect, for example, the value function. Clearly this is not correct.

This last problem can be resolved by exchanging the simple node update rule, equation 5.5, for the node update rule obtained as the limit of a series of updates, equation 4.7. If we use this update rule to derive the equation for influence again we obtain

$$I_n(x_m) = (1 - \lambda h) \sum_{j=1, j \neq m}^{L} \frac{w_m(y_j)}{1 - (1 - \lambda h) w_m(y_m)} I_n(x_j). \tag{5.8}$$

With this equation, no influence from a node will be assigned back to itself, and hence influence is not dependent upon step size (apart from the usual accuracy issues).

However, the first two problems remain. Therefore we propose an alternative formulation of influence that avoids these problems by calculating an approximation to the influence as defined by equation 5.8. Let us define y_i^- as the end-point of a backwards integration step from x_i which, for an Euler step, is given by $y_i^- = x_i - hf(x_i, a_i)$. Then we can obtain an

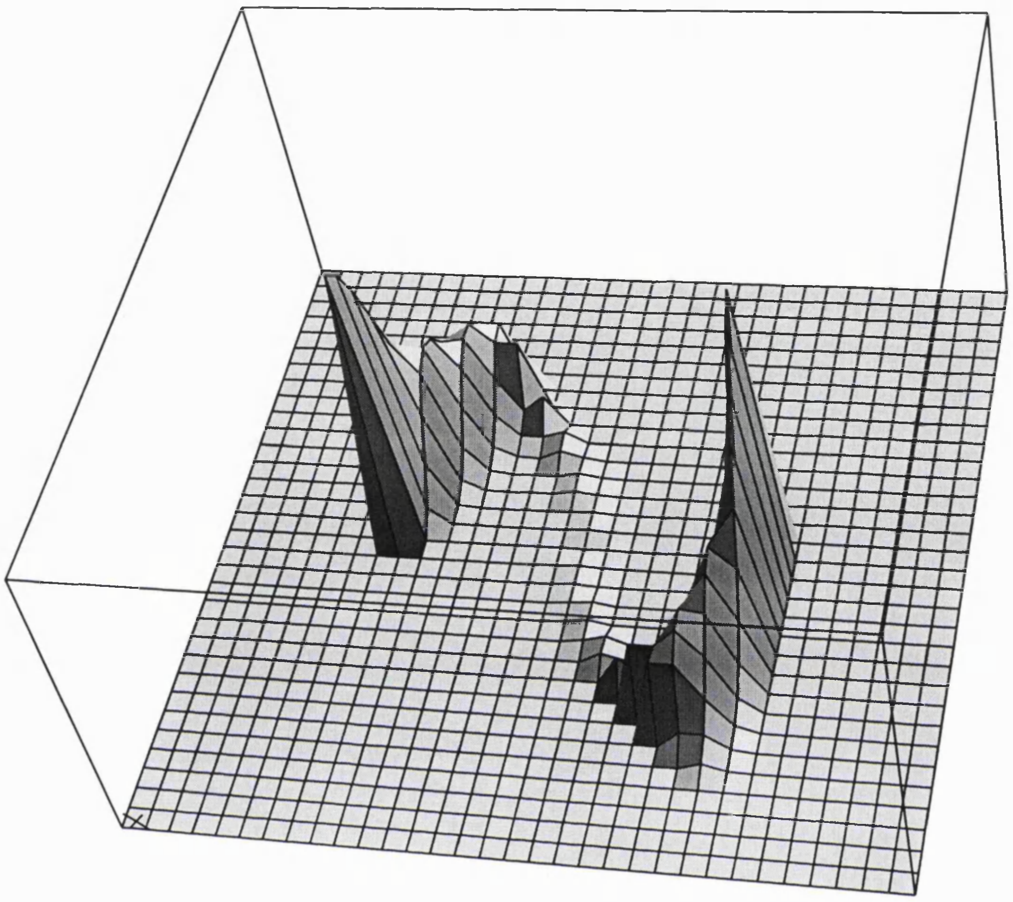


Figure 5.9: influence upon the points $(-0.5, 0)$ and $(0, 0.5)$ for the Bush problem

approximation, $\hat{I}()$, to the influence from

$$\hat{I}(x_m) = (1 - \lambda h) \sum_{j=0}^{L} w_j(y_m^-)\hat{I}(x_j) \tag{5.9}$$

This has the advantage that it can be calculated in much the same way as the membership or value function, with the sole difference being that a backwards (in time) integration step is used instead of a forwards one. We need only store a single influence function and constrained nodes can be handled in the usual way.

Figure 5.9 shows the result of calculating the influence upon the points $(-0.5, 0)$ and $(0.5, 0)$ in the Bush problem. Influence is spread out along the optimal trajectory from each point, almost as if it were via a process of diffusion.

To apply our method for calculating influence to the problem of generating bang-bang controllers, we followed the example of Munos and Moore [60, 63]. Their implementation worked

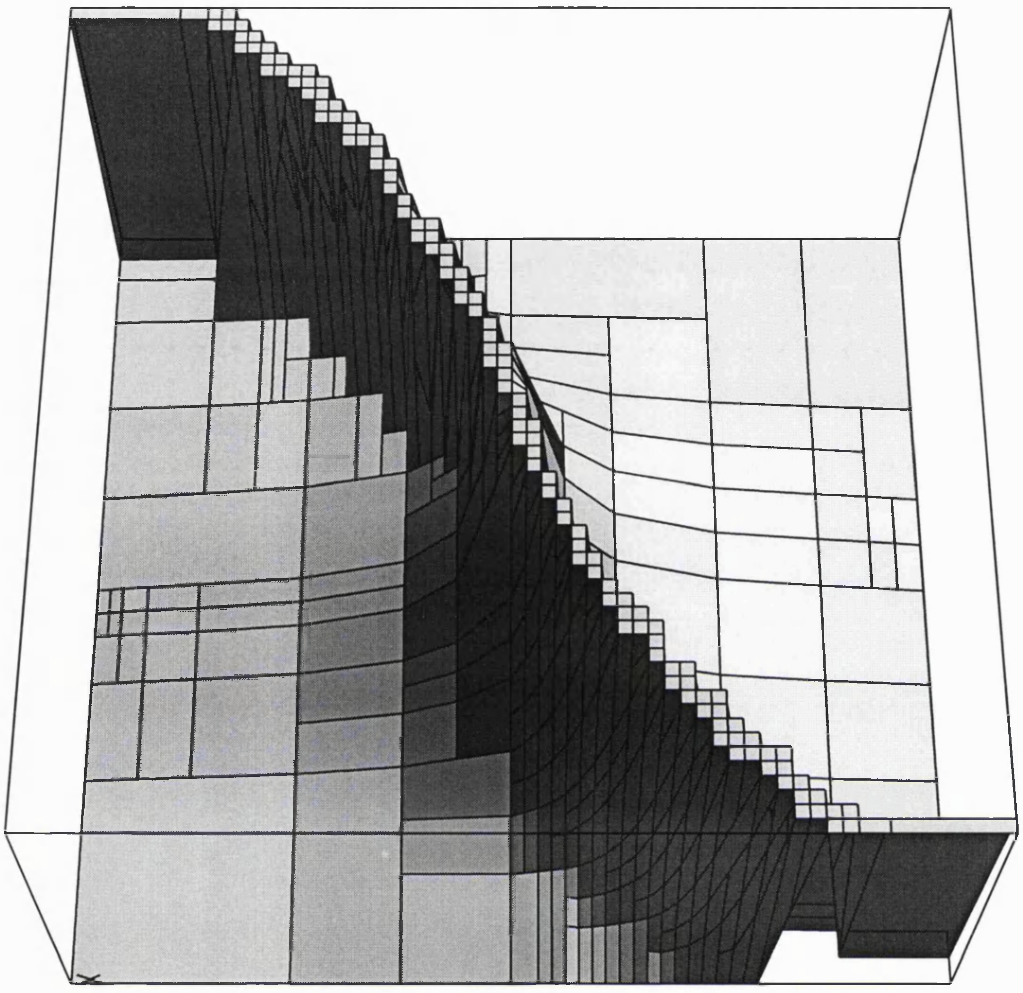


Figure 5.10: influence function for Bush problem on adaptive mesh

by identifying every node adjacent to a cell with non-constant control and labelled these as 'hot' nodes. The influence of each hot node is set to one and the total influence function calculated accordingly. The existing subdivision criterion is then multiplied by the total influence function to obtain a new criterion. Figure 5.10 shows the influence function obtained in this way for the modified Bush problem, using the policy of subdividing on value function error first, and then including control function error (as in figure 5.7).

As might be expected it places very high emphasis on the immediate neighbourhood of the switching curve with influence falling off very rapidly away from that curve. Observation of the solution in progress shows that the 'width' of the influence function ridge depends upon the mesh density; as mesh density increases the ridge becomes narrower. This is the expected behaviour since a perfect solution, if such a thing were attainable, would have an infinitely narrow influence function located precisely on the switching curve.

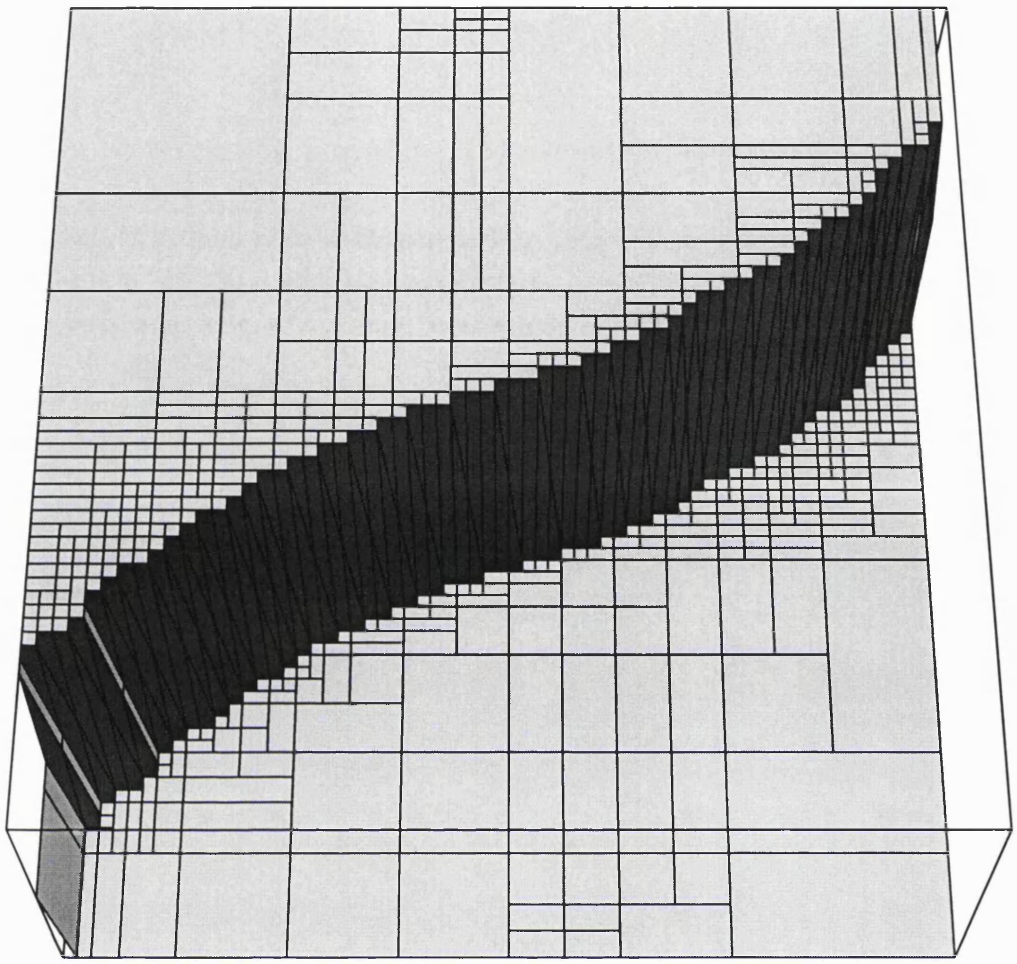


Figure 5.11: control function obtained using influence metric on adaptive mesh

Figure 5.11 shows the control function obtained from the same solution used to construct figure 5.10. The influence metric has removed all of the unnecessary subdivision that was occurring at the boundary of the reachable set and has provided a much smaller mesh. The performance score obtained with this mesh was 331.1 with 563 free nodes. When the modified Bush problem was solved again by subdividing on value function error alone, the performance measurement was better at 326.9 with 340 free nodes.

These performance results are close to those given in table 5.3 (65x65 max resolution) which were obtained without the influence metric. However, when the influence metric was employed far fewer free nodes were required and hence the performance scores were obtained at a lower computational cost.

In the next chapter we shall describe an experiment in which controller performance is measured at several stages as the mesh is refined and the number of free nodes is allowed to increase. In this way, it is possible to compare the performance obtained both with and without the influence metric using meshes with similar numbers of free nodes.

5.4 Summary

In this chapter, the following points have been made:

- Controller performance is a good objective means of discriminating between algorithm design choices.
- Performance can be measured by integrating the cost function along a trajectory until the discount factor makes futures costs negligible.
- The performance of the whole controller can be fairly assessed by running simulations from a set of starting points distributed randomly and uniformly inside the domain.
- Control function extrapolation can be used to maximise the reachable set achieved by a controller and to eliminate undesirable and arbitrary changes in the control function.
- A controller can be constructed either using interpolation between mesh nodes or by recalculation, i.e. searching again for the optimal control at a point.
- Recalculation can provide superior performance, if the value function is sufficiently well approximated. However, it can fail to provide sensible control at sub-cell scales, which makes interpolation a safer choice.
- The influence metric can be used to focus subdivision on those regions that most directly affect the control function.

The result that subdivision on control function error did not provide an improvement in controller performance, and can even be counterproductive, may be an indication that the balance between accuracy of the value function and accuracy of the control function is wrong. Our method actually defines that balance implicitly even though bang-bang control means that the value of the parameter, k, in equation 5.3 is not important, provided it is large enough to trigger subdivision. The value function, for the modified Bush problem, is continuous near the switching curve and hence the accuracy of the value function is limited by the subdivision threshold. The control function, however, is discontinuous at the switching curve and hence subdivision continues up to the maximum resolution of the mesh. What is required is a better means of quantifying the relative importance of value function and control function accuracy so that we may ensure that computational resources are allocated in the most effective way.

Chapter 6

Applying the HJB Method

The problems tackled with the HJB method in this thesis have so far all been simple, two dimensional, examples. This chapter describes the application of the HJB method to some more complex and interesting problems and assesses the utility of the method for these problems.

We begin by discussing the decisions that must be made when applying the HJB method to a new problem. The 'cart and pole' problem is then solved using a number of variations on the adaptive meshing algorithm. The performance of each variation is measured to determine which is the most promising. The HJB method is next applied to the problem of balancing the acrobot. A combined swing-up and balance controller is obtained.

6.1 Solving New Problems

Several decisions must be taken when tackling new problems with the HJB method. Many of these are about explicit parameters of the algorithm, whilst others are about the way that the problem is presented to the algorithm. An informed set of choices can be critical to obtaining a useful solution. For example, if the computational domain is too small then some useful trajectories may not be found. Likewise, if the domain is too large much computation may be wasted or there may be insufficient mesh resolution, even with adaptive meshing. It is worthwhile to describe the relevant decisions, so that we are aware of the issues and to ensure that nothing is overlooked.

What follows is a description of those decisions. We have divided them into three categories: first the problem specific parameters, that must be addressed in each case, second, the parameters related to adaptive meshing and, lastly, algorithm choices that are not necessarily specific to an individual problem, but may nonetheless affect the outcome.

6.1.1 Problem Specific Parameters

Cost function The cost function shapes the solution and determines what kind of behaviour the controlled system will exhibit. It is designed to reward desired behaviours and penalise undesirable behaviours. For example, a minimum time cost function penalises only the time taken to reach some target. The cost function can also be used to provide a balance between minimum-time and minimum-effort solutions.

Size of the domain This controls the set of trajectories that may be included within the solution. If the domain is too large then computation may be wasted or there may be insufficient available mesh resolution. If the domain is too small then some useful trajectories may be excluded. Often the size of the domain can be determined from a knowledge of the system. For example, if it is known that the system has a natural maximum velocity, due to dissipative forces, then that could provide a natural extent for the velocity axis.

Set of controls A larger set of control input vectors may provide greater accuracy, but will incur correspondingly greater computational cost as they must be repeatedly searched for an optimum. If a bang-bang controller is sought then it is sufficient to search the 'corners' of the control space.

Discount factor If future costs are not expected to drop to zero then it is necessary to use a non-zero discount factor to prevent the value function becoming infinite. The discount factor should be large enough to provide a sensible upper bound for the value function. If the discount factor is too large then the controller may place too much emphasis on minimising short term costs and thereby fail to satisfy the overall goal. As an example, consider the problem of swinging a pendulum up whilst using a minimum amount of control. If the discount factor is too large then it may be optimal to do nothing (hence minimising control), rather then complete the swing-up manoeuvre. In some cases the problem will have a natural timescale to determine the discount factor. One such example is a walking controller, where the timescale would be 'a few' strides.

Integration step size This should be the largest timestep that may be used to integrate the equations of motion to the required level of accuracy and stability.

Convergence threshold This is the maximum change in value of any single node during the first value pass after a control pass. If it is too low computation is wasted. If it is too high subdivision may take place prematurely leading to poor subdivision decisions.

Boundary value The value assigned to points outside the domain should be chosen such that every part of the value function inside the reachable set has a value lower than the boundary value. For instance, if there is a known maximum cost and a non-zero discount factor then they provide an upper bound for the value function.

System parameterisation Most systems can be parameterised in more than one way. An example is the acrobot, a double pendulum with a single actuator, which we will discuss in section 6.3. An obvious parameterisation for this system is formed by the angles of the two rigid links to the vertical and the corresponding angular velocities. However, we have found that an alternative parameterisation, with the angle of the centre of mass to the vertical as one parameter, yields better controller performance.

6.1.2 Adaptive Meshing Parameters

Initial subdivision This defines the depth of the initial spatial subdivision tree before adaptive subdivision is applied. An initial subdivision serves to place nodes in important parts of the domain, such as a target set. It also limits the extent to which the subdivision tree can become unbalanced and limits the extent of any poor decisions made in the early stages of adaptive subdivision.

Subdivision limit A limit is imposed upon the maximum number of subdivisions along any one dimension. The subdivision limit is set by choosing the initial integer range of each dimension (which is always a power of 2). When the integer width of a cell is 1 it can no longer be subdivided along that dimension. If the subdivision limit is too low then there may be insufficient available mesh resolution. If the subdivision limit is too high then excessive subdivision may occur.

Subdivision threshold New free nodes are added where the subdivision criterion is above the subdivision threshold. If it is too high then there will insufficient accuracy in the mesh. If the threshold is too low then there will be excessive subdivision, particularly where control based subdivision is applied after value based subdivision has terminated, or nearly terminated.

Subdivision balance The accuracy of both the value function and the control function affect the performance of the resulting controller. Subdivision balance controls the relative accuracy of the value function and control function. The optimal balance will provide the greatest improvement in overall controller performance each time the mesh is refined.

Subdivision rate The subdivision rate is the ratio of new free nodes to old free nodes permitted during each subdivision phase. It is used to ensure that mesh refinement is progressive. Clearly, if the subdivision rate is too low then the mesh will grow very slowly. If, however, it is too large then many poor decisions might be made in the subdivision process as a consequence of adding too many nodes before allowing the solution to progress.

Error multiplier A function may be used used to modify the error metric (subdivision criterion), and hence to modify subdivision decisions. An example is the influence metric, described in section 5.3.1. Other possible error multipliers include the membership function (to focus subdivision in regions of high membership) and the value function (to focus subdivision in regions of high or low value).

6.1.3 Algorithm Choices

Integration method We have employed the Euler method, though a higher order Runge-Kutta method could be used.

Interpolation method Simplicial or multilinear interpolation may be used. Multilinear is better behaved than simplicial but significantly more costly in more than three dimensions. Simplicial interpolation leads to discontinuities in the mesh, which do not occur with multilinear interpolation (see section 4.3.4).

This is a long list of parameters, and unfortunately there is not always an obvious way to determine an appropriate value for each parameter. We must use our knowledge of the problem and the solution method coupled with a bit of trial and error to obtain a useful set of parameters.

Sometimes, information revealed by one solution, however bad that solution may be, can improve our understanding of how to tackle the problem. For example, if some trajectories pass along the boundary of the domain then it is likely that the performance of those particular trajectories could be improved by increasing the size of the domain. So, the process of tackling a new problem is, in part, a process of exploring the problem by building solutions and using those to refine the approach.

The next section introduces the cart and pole problem to illustrate the process of tackling a new problem. The problem will subsequently be useful as a test case for evaluating the performance of variations on the adaptive meshing technique (in section 6.2).

6.1.4 The Cart and Pole

Balancing the cart and pole is a classic control problem that is familiar to anyone who has ever tried to balance a ruler on the end of their finger. Figure 6.1 illustrates the system. It is composed to two point masses, m_1 and m_2, joined by a rigid massless rod of length l. For simplicity we assume that the system exists in a 2 dimensional space with the base constrained to run along the x-axis. Control is applied as a horizontal force to m_1, and the aim is to keep the system balanced with m_2 remaining above m_1. The equations of motion in terms of the

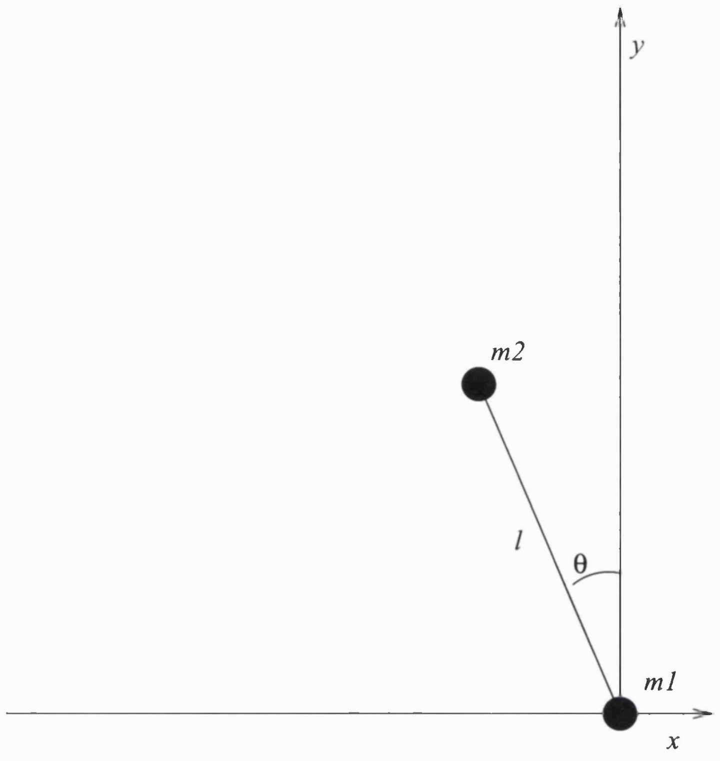


Figure 6.1: The cart and pole

generalised coordinates of this system, $\mathbf{q} = (x, \theta)^T$, are

$$\begin{pmatrix} m_2 l \dot{\theta}^2 \sin\theta \\ 0 \end{pmatrix} + \begin{pmatrix} m_1 + m_2 & -m_2 l \cos\theta \\ -m_2 l \cos\theta & m_2 l^2 \end{pmatrix} \ddot{\mathbf{q}} - \mathbf{Q} = 0 \tag{6.1}$$

where $\mathbf{Q}$ is the vector of generalised forces and is given, in this case, by

$$\mathbf{Q} = \begin{pmatrix} a \\ m_2 l g \sin\theta \end{pmatrix} \tag{6.2}$$

a is the horizontal force applied to m_2 and g is the acceleration due to gravity. Note that this model doesn't include any friction or damping.

6.1.5 Solving the Cart and Pole

The first step in generating a controller for the cart and pole is to choose the physical parameters of the system and controller. For this example the most obvious set of physical parameters will do, i.e. $m_1 = 1$, $m_2 = 1$ and $l = 1$. The controller needs to be strong enough to recover the pole from a reasonable angle, say $\frac{\pi}{4}$ radians. This in turn defines a convenient range for the controller, $a \in [-2g, 2g]$, so that it is possible to recover balance from any static configuration with $\theta < \frac{\pi}{4}$.

Next, a domain and a finite set of controls are required. It is a convenient property of the system dynamics, given by equation 6.1, that they are independent of x. This allows us to analyse and control the system, as if it only had three variables, i.e with a domain in $\mathbf{x} = (\theta, \dot{x}, \dot{\theta})$. Such a domain can give us direct control over $\dot{x}$ but not over x. Of course, it is possible to control position, given control over velocity. However position control will not be optimal, as it would be with a four dimensional domain. The advantage is that a much smaller mesh can be used, making the problem easier to solve.

A bang-bang solution is sufficient for our purposes, hence we only need the extreme controls, plus the control required for static balance, i.e. $a = \{-2g, 0, 2g\}$. With this set of controls, the size of the domain in θ is obvious, i.e. $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$. The size of the domain in the other two dimensions was determined empirically, such that balance could be recovered from static configurations with angles near to $\frac{\pi}{4}$. In practice the domain $[-\frac{\pi}{4}, \frac{\pi}{4}] \times [-5, 5] \times [-\pi\, \pi]$ worked well.

The next important decision is the cost function. For a minimum time controller the cost function would be

$$l(x, a) = \begin{cases} 0 & ||x|| \leq \varepsilon \\ 1 & ||x|| > \varepsilon \end{cases} \tag{6.3}$$

However, in practice we have obtained better controller performance with the following, continuous, cost function

$$l(x, a) = |\dot{x}| \tag{6.4}$$

This produces a similar controller to that produced by the minimum time cost function. It works because the system can only maintain zero horizontal velocity in the balanced position.

The advantages of continuous cost functions, such as equation 6.4, over discontinuous cost function, such as equation 6.3, were discussed in section 5.1.1. In summary, they provide a smoother value function that may be approximated more accurately, with fewer nodes, on the numerical mesh. Also, they provide a better basis for comparing the performance of different controllers because the penalty for degraded performance is gradual, rather than a step change.

The integration step size was chosen as, $h = 0.001$, which is small enough to integrate the equations of motion stably, using the Euler method, but not so small that it precludes real-time animation.

The boundary value must be chosen such that it is larger than the value function at any point within the reachable set. In practice this can be checked by finding the maximum of the value function at every node with membership less than a threshold close to one (e.g. 0.9). A boundary value of 20 was found to satisfy this requirement and was used for this example.

The discount factor was chosen to provide a reasonable, natural limit to the duration of trajectories for the comparison in the next section. In this case a discount factor of 0.5 was

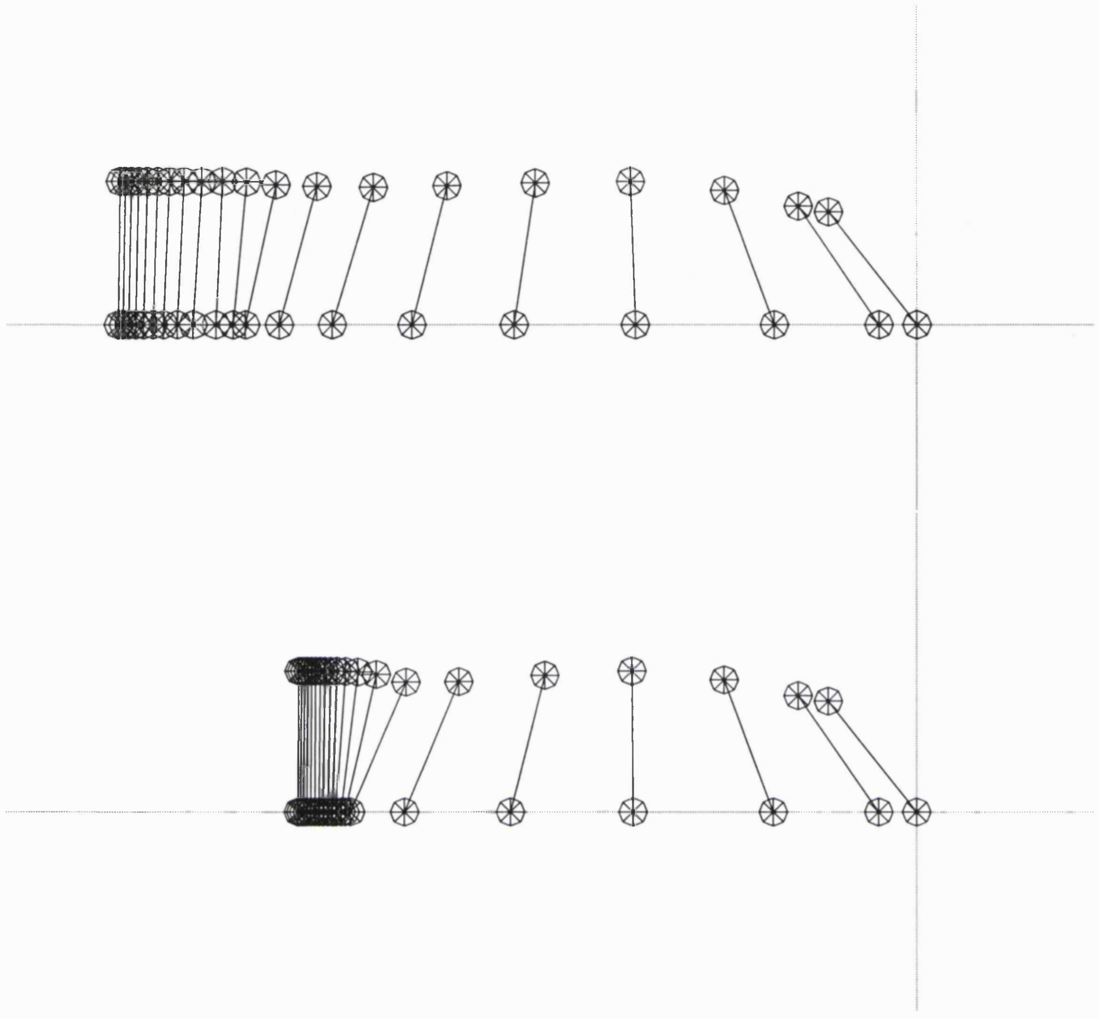


Figure 6.2: cart and pole animation sequences with 4913 and 449401 free nodes

used.

With the parameters outlined we were able to obtain an effective controller using a small regular (non-adaptive) mesh of 17x17x17 nodes.

Figure 6.2 shows two animation sequences of the cart and pole recovering balance from a static unbalanced configuration. The cart and pole is drawn every 0.2 seconds over a four second interval. The upper sequence was generated with the small regular mesh (4913 free nodes). The lower sequence was generated after refining that mesh using anisotropic adaptive meshing and the influence criterion. The mesh used for the lower sequence had 449401 free nodes.

Table 6.1 summarises the parameters used to solve the cart and pole problem. These are also the parameters that will be used in the next section in order to compare the performance of various meshing strategies on this problem.

parameter	value
domain	$[-\frac{\pi}{4}, \frac{\pi}{4}] \times [-5, 5] \times [-\pi, \pi]$
control set	$\{-2g, 0, 2g\}$
subdivision limit	$129 \times 129 \times 129$
initial subdivision	$17 \times 17 \times 17$
subdivision threshold	$\frac{h}{5}$
timestep h	0.001
subdivision rate	2
convegence threshold	0.01
boundary value	20
discount factor	0.5
interpolation method	simplicial
integration method	Euler
cost function	$l(\mathbf{x}, \mathbf{a}) = \dot{x}$
l	1
m_1	1
m_2	1
simulation duration	$20s$
control subdivision factor	$h/2g$

Table 6.1: solution parameters for cart and pole test

The adaptive meshing parameters were chosen to support this performance comparison. For example, the initial subdivision can serve as a baseline for comparing the performance of subsequent refined meshes because it provides an effective controller.

6.2 Adaptive Mesh Evaluation

The adaptive meshing technique has been described in detail, but we do not yet know how much advantage, if any, may be gained. The question therefore remains: is the adaptive meshing technique significantly better than the regular meshes with which we started? This section attempts to address that question by comparing the performance of the following meshing strategies:

- regular,
- isotropic adaptive,
- anisotropic adaptive,
- isotropic adaptive with influence criterion,
- anisotropic adaptive with influence criterion,

The adaptive meshing scheme employed is the edge centred scheme described in section 4.4.3. To form an isotropic meshing scheme the edge centred scheme was modified such that subdivisions are always introduced in sequence as the kd-tree is descended. Hence, if a two dimensional cell must be split in the y dimension, in order to enforce mesh conformance around a new free node, and the next dimension in sequence is x, then the cell is first split in the x dimension and one of two resulting sub-cells is then split in the y dimension.

The problem of balancing the cart and pole is used as the test case, since it is more challenging than the Bush problem, but can be solved relatively swiftly in a three dimensional domain.

6.2.1 Performance Measurements

Performance measurements were taken using the same method employed for the modified Bush problem, described in section 5.1.1. A random set of 1000 starting points, evenly distributed throughout the computational domain, was used. A simulation was run for 20 seconds from each starting point to determine whether the controller succeeded (the pole balanced) or failed (the pole fell over) and to calculate a performance score if it did balance. To provide a fair comparison, a subset of the starting points was found such that every controller successfully balanced the pole from every point in the subset. A total performance score was then calculated

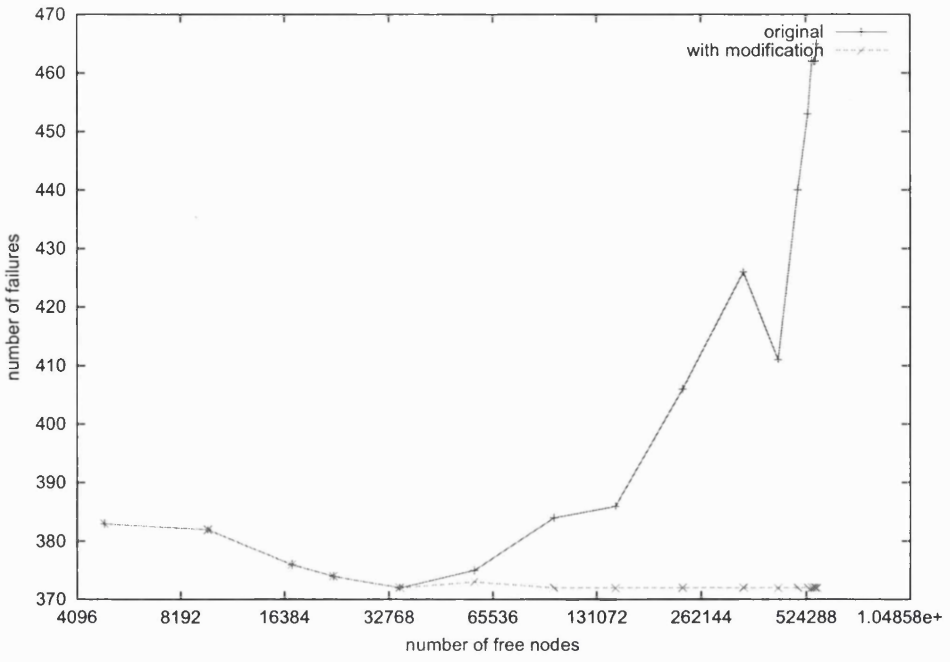


Figure 6.3: effect of controller modification on cart and pole failure rate

for each controller by summing the score obtained with every starting point in the subset. In this way it was possible to compare the performance of all the controllers on a common scale.

Figure 6.3 shows the number of failures for one controller (isotropic adaptive, with influence) as the number of free nodes increases. Note that the number of nodes in the regular mesh was increased by doubling the resolution of the mesh in one dimension, and then doubling the resolution in the next dimension in sequence, etc.

Unexpectedly, the number of failures rose as the number of free nodes increased. We would hope to see the number of failures decrease and approach some lower bound, according to what is physically possible, as the number of free nodes increases. A little investigation revealed that all the new failures occurred when the horizontal velocity of the lower mass ($\dot{x}$) exceeded the bounds of the computational domain by a small amount. When this happens the simulation simply sets the control to zero and the pole falls over. A small modification was made to the simulation algorithm, such that the domain coordinates were clipped to the domain for the purposes of determining a control value. In effect the control function was extrapolated by using its value at the domain boundary. The results of this modification on the number of failures is also shown in figure 6.3. Clearly this modification substantially improved the reliability of the controller.

The increase in failures occurred because, for some starting points, the theoretical optimal trajectory passes along the boundary of the domain. As the mesh was refined, and controller performance improved, the trajectories calculated from those starting points passed closer to the boundary of the domain. Occasionally, integration error will cause such a trajectory, that should remain with the domain, to step just outside it. In these cases control function extrapolation is generally sufficient to maintain control of the system.

Figure 6.4 shows the number of failures (top) and the performance (bottom) for all the test cases (with control function extrapolation). Overall, the anisotropic adaptive meshing scheme with the influence criterion performed best, just slightly better than the "isotropic with influence" case. The influence criterion has clearly provided a significant benefit, whilst the anisotropic meshing has provided only a small, and rather disappointing, benefit. The regular mesh performed significantly better than the isotropic mesh, without the influence criterion.

These results indicate that adaptive meshing performs significantly better than the regular mesh, if the influence criterion is employed.

6.3 The Acrobot

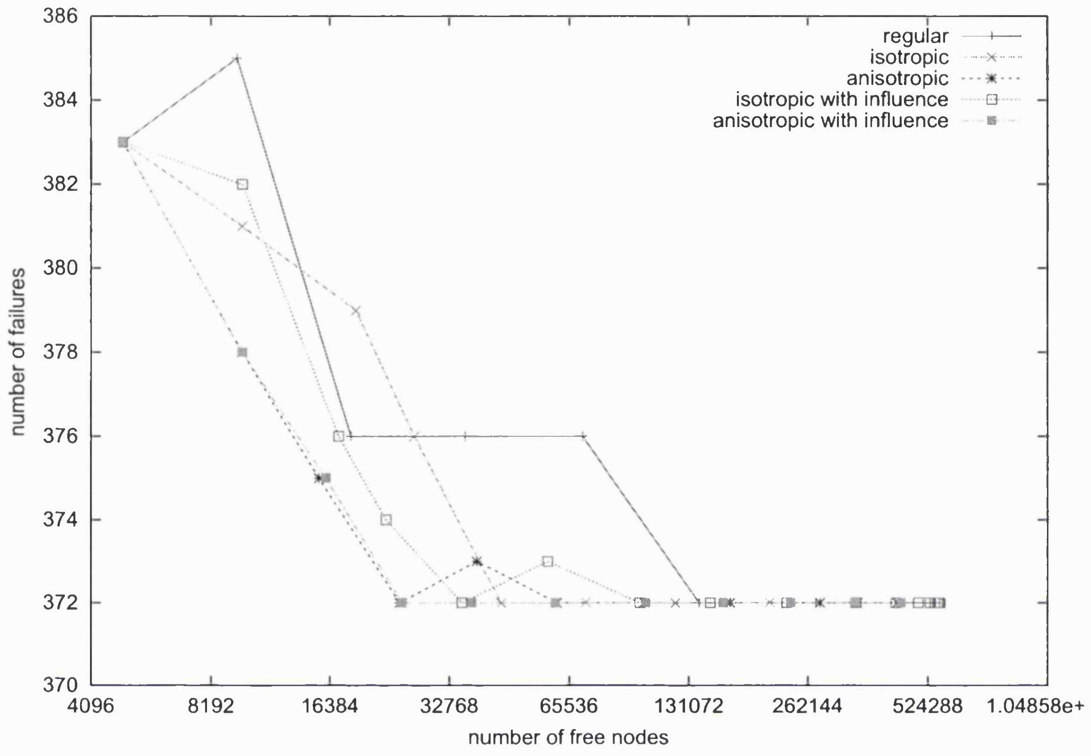
One of the main difficulties in generating effective walking controllers for humanoids is the weak control that the foot and ankle can exert (ankle torque) to help maintain balance. In fact, the balance that we humans achieve whilst upright is a combination of swaying, stepping and using ankle torque. Swaying and stepping can provide dynamic balance, whereas ankle torque is mostly associated with static balance. Many attempts at biped locomotion for robots have suffered from an over-reliance upon static balance and control exerted at the ankle, e.g. Honda's humanoid robots [42].

In fact, it is possible to walk when there is no ankle joint and the contact area between foot and floor is effectively a point (point feet). This is the case when people walk on stilts. It is also possible to build a robot that can stand and walk as if it were on stilts, i.e. with point feet [86]. This is a useful exercise because it forces us to tackle the problems of dynamic balance for humanoids.

The acrobot is a classic control problem that was originally envisaged as an approximate model of a gymnast on the high bar. It can also serve as a model of the swaying component of standing balance. Like a human it is nonlinear and underactuated, meaning that it has fewer actuators than degrees of freedom.

6.3.1 Description of the Acrobot

Figure 6.5 illustrates the acrobot. The base is pinned to the origin. The system is parameterised by two angles $\theta_1$and θ_2, both of which are measured relative to the vertical (y-axis). There are two point masses, as shown, with the joining rods considered to be rigid and massless. The rods



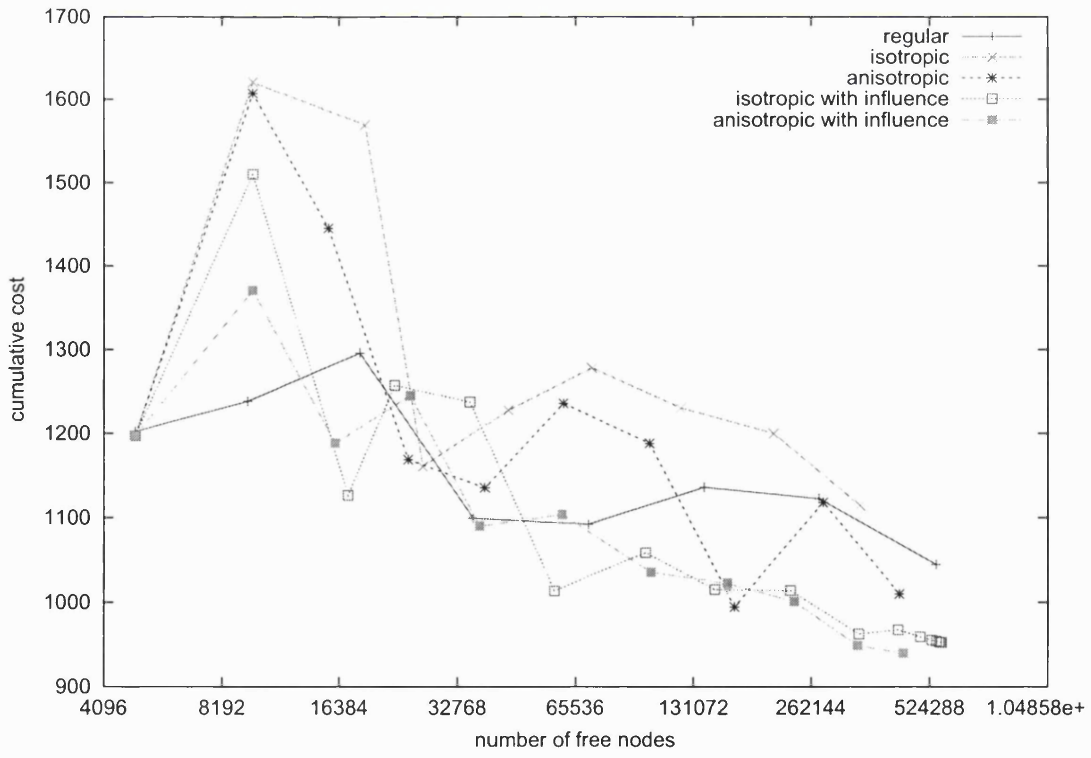


Figure 6.4: cart and pole: number of failures and performance for all test cases

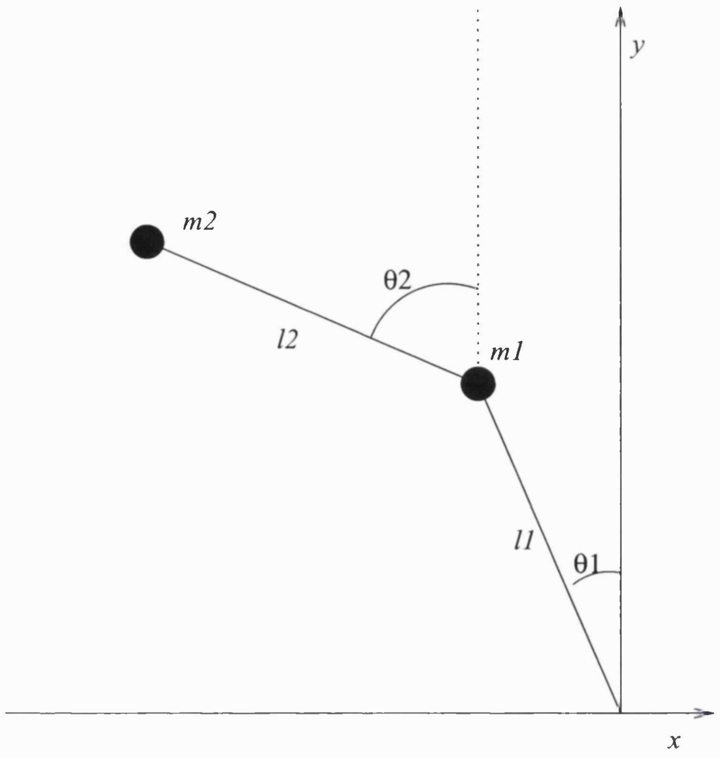


Figure 6.5: The Acrobot

have length $l_1$and l_2 and the masses are m_1 and m_2.

The equations of motion for the acrobot can be formulated as

$$\begin{pmatrix} (m_1+m_2)l_1^2 & m_2 l_1 l_2 \cos(\phi) \\ m_2 l_1 l_2 \cos(\phi) & m_2 l_2^2 \end{pmatrix} \ddot{\mathbf{q}} - m_2 l_1 l_2 \sin(\phi) \begin{pmatrix} -\dot{\theta}_2^2 \\ \dot{\theta}_1^2 \end{pmatrix} - \mathbf{Q} = 0 \tag{6.5}$$

where

$$\phi = \theta_1 - \theta_2 \tag{6.6}$$

and the vector of generalised coordinates is $\mathbf{q} = (\theta_1, \theta_2)$. The generalised force vector, $\mathbf{Q}$, contains a component, $\mathbf{Q}^\tau$, due to the torque applied as a control input at the middle joint, and a component, $\mathbf{Q}^\mathrm{g}$ due to gravity. These are given by

$$\mathbf{Q}^\tau = \begin{pmatrix} \tau \\ -\tau \end{pmatrix} \tag{6.7}$$

and

$$\mathbf{Q}^\mathrm{g} = \begin{pmatrix} (m_1+m_2) g l_1 \sin\theta_1 \\ m_2 g l_2 \sin\theta_2 \end{pmatrix}. \tag{6.8}$$

6.3.2 Acrobot control tasks

There are two 'classic' problems that have been tackled with the acrobot. The first is to swing the acrobot up. It begins with the acrobot stationary and hanging vertically down below its base. The acrobot must swing to and fro in order to raise itself into an upright position with its centre of mass above its base, just as a gymnast on the high bar must swing-up. The second problem is to balance the acrobot with its centre of mass vertically above its base.

The balance can either be in the gymnasts handstand position, i.e. vertically up with $q = (0,0)$, or in any of the set of balanced positions that exist where the centre of mass is vertically above the base. These balanced positions form a one dimensional manifold, the *equilibrium manifold*, within the two dimensional configuration space. Viewed as a gymnast, the base of the acrobot corresponds to the gymnasts hands. However, if we turn the gymnast the other way up, so that the base of the acrobot is now the gymnast's feet, then it becomes a model for standing balance.

This standing balance problem is similar to the task of walking the high wire. Usually, high wire walkers employ a long pole to improve the control they have over their own balance. Balancing the acrobot is like balancing on the high wire, facing perpendicular to the wire, without the pole and without even using the arms. Balance must be maintained by bending only at the hip. To get some idea of how this works, try the following simple experiment: place hands in pockets, rock onto the heels of both feet and attempt to balance by bending at the hip, without moving the feet[1]. It's not easy.

One feature that makes the balance problem difficult is that it is necessary to lean into a fall in order to regain balance. Figure 6.6 illustrates the acrobot recovering balance from an initial unbalanced configuration.

This can be explained by considering the motion of the acrobot's centre of mass in a small region close to the upright balanced position. The only external forces acting upon the acrobot are due to gravity and the reaction force at the base joint. When the acrobot is in the upright balanced position these forces are aligned in the y-axis and they cancel. However, if a torque is applied at the middle joint, so as to make the top of the acrobot move to the left then, by symmetry of action, the bottom of the acrobot would also move to the left, were it not pinned. Since the base is pinned, a reaction force is induced at the pin to preventing the base from moving. Hence there is an x component of the base reaction force, to the right. The sum of external forces acting upon the acrobot is no longer zero, and the centre of gravity must

[1]If possible, it is better to try balancing whilst standing on a narrow plank with the arch of the foot (wear stiff shoes).

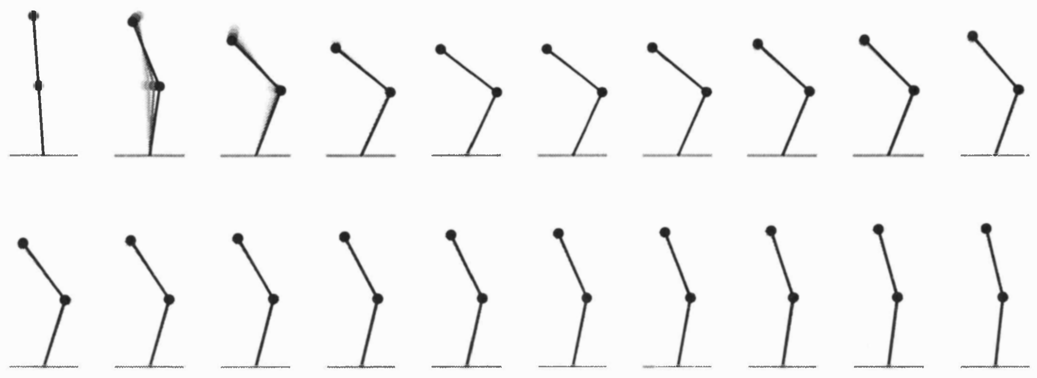


Figure 6.6: acrobot regaining balance

therefore move to the right. So, if the acrobot is out of balance and the centre of mass is to the left of the base, then it is necessary to force the upper link of the acrobot further to the left in order to drive the centre of mass back to the right.

There are two versions of the swing-up problem for the acrobot. The simpler version of this task is to raise the acrobot to some set of upright target configurations without regard to its velocity upon reaching the target. A more difficult task is to also bring the acrobot approximately to rest on the equilibrium manifold, such that it enters the reachable set of a separate balance controller. Swing-up controllers generally work by pumping a bit of energy into the system with each swing until sufficient energy is present to complete the swing up.

The problems of swing-up and balance are a good test of the HJB method. In principle, given a sufficiently large mesh, the HJB method should be capable of generating a single unified solution to the combined problem of swing up followed by balance. The acrobot has two degrees of freedom and a four dimensional phase space. A manageable number of dimensions for the HJB method.

In addition to swing up and balance the acrobot is capable of a whole range of acrobatic manoeuvres [44, 43]. If we imagine that the base of the acrobot is resting on the floor, and we permit it to become detached from the floor, then it is capable of locomotion via hopping [13, 15, 51] or via a cartwheel gait. If it can be persuaded to jump high enough then, like a gymnast on the floor, it should be capable of spectacular acrobatic manoeuvres. We have not attempted to implement such behaviours with the HJB method.

6.4 Acrobot Literature Review

There have been several published controllers for the acrobot. We now review those publications.

6.4.1 Control Theoretic Approach

One of the earliest papers on the subject was by Murray and Hauser [64]. They begin by proving the following:

1. The acrobot is controllable within a suitably small region of the manifold of equilibrium positions.

2. The acrobot is not *input/state linearizable.* This means that there does not exist any change of coordinates and control law such that the resulting dynamics are linear.

Their approach is to construct an approximation to the system, about the manifold of equilibrium positions, that is fully input/state linearizable. Controllers constructed for this approximation proved capable of balancing the acrobot and tracking slow trajectories along the equilibrium manifold.

Bortoff and Spong [19] used a technique called *pseudolinearization* to control the acrobot about the equilibrium manifold. Spline functions were used to construct a nonlinear control law and a nonlinear change of coordinates such that the resulting dynamics are approximately linear. They demonstrated the effectiveness of their control technique experimentally.

Spong studied the problem of "swinging up" the acrobot to connect with a balancing controller [91, 92]. He applied a nonlinear feedback law such that the combined dynamics of the acrobot plus feedback law were partially linear. This is called *partial feedback linearization.* In this case the feedback linearization was partial in the sense that it linearized the response of the second link, not the whole system. The linearization was used to swing the second link between fixed angles $\pm\alpha$ relative to the first link. The desired joint angle, ϕ^d, is given by

$$\phi^d = \alpha sgn(\dot{\phi}) \tag{6.9}$$

where ϕ is the angle of link 2 relative to link 1. This works to pump the second link in phase with the motion of the first, much like a child on a swing, or a gymnast on the high bar, increasing the energy of the system at each pump.

6.4.2 Fuzzy Logic Approach

Lee and Smith applied fuzzy logic to acrobot swing up and balance, with the vertically upright position as goal [49]. They describe automatic methods for the design and tuning of fuzzy systems. Specifically, they employed genetic algorithms, dynamic switching fuzzy systems and meta-rule techniques. Smith et al later compared this controller to one based upon linearization techniques [89]. They concluded that, although their controller showed better performance when there is no external disturbance, it was not robust to external disturbances. Subsequently they 'fixed' their fuzzy controller to make it more robust to disturbances by limiting the angular velocities used for balance, and by tuning the system [88].

6.4.3 Reinforcement Learning Approach

Sutton applied reinforcement learning (Sarsa algorithm) to the swing-up problem [94]. He used an acrobot with dimensions $l1 = 1m$ and $l2 = 2m$ and the goal of raising the tip of the acrobot to a height of $1m$ above the base. This is actually an easy task to achieve since the target set is such a large part of the configuration space that, once the acrobot has been given sufficient energy, it's uncontrolled motion will inevitably reach this goal quite quickly. Although the experiments were all done in simulation, a model-free algorithm was used in the sense that the training only took place as if a real acrobot were present. Each episode began with the acrobot hanging vertically down and stationary and continued until the goal was reached, which always happened eventually. Sutton chose a low limit for the available torque of $\pm 1Nm$ which meant that the acrobot had to swing back and forth several times before it could reach the goal.

Boone examined the use of heuristics and approximate models to speed up reinforcement learning for the acrobot [17]. Using the same model and task as Sutton, he compared learning with model-based, model-free and learned approximate models. He concluded that the availability of an accurate model significantly improved learning speed.

Munos and Moore solved the infinite horizon Hamilton-Jacobi-Bellman equation for swing-up control of the Acrobot, using adaptive meshing techniques [61, 60]. They used the following cost function

$$l(x,a) = \begin{array}{c|c} 1 & \theta_1 \in [-\frac{\pi}{16}, \frac{\pi}{16}], \phi \in [-\frac{\pi}{16}, \frac{\pi}{16}] \\ 0 & elsewhere \end{array} \tag{6.10}$$

which defines the goal as a small region around the vertically upright position. This task is more difficult than that attempted by Sutton and Boone simply because the goal is narrower. Also, owing to the infinite horizon formulation, an optimal policy would be a combined swing-up and balance controller. Regrettably Munos and Moore provide no data about the absolute performance of their solutions, they only provide data about the relative performance of different meshes. It seems likely, from supplied diagrams, that successful swing-up was obtained, but we can only guess as to how effective their controllers were for the balancing task.

Yoshimoto et al applied reinforcement learning to the problem of balancing in the vertically upright position [115]. They employed an actor-critic algorithm (the actor is the control function and the critic is the value function) with normalised Gaussian networks for the actor and critic models. An on-line expectation maximisation algorithm was used for the training. A dynamic allocation scheme was employed to find an efficient set of Gaussian units for the actor and critic nets. Successful balancing, from starting configurations close to balanced, was reported after a small number of learning episodes (37 episodes of 7 seconds each) and using a small number of Gaussian units (38 for the actor, 54 for the critic).

6.4.4 Hopping and Other Behaviours

Early work on making the acrobot hop was conducted by Berkemeier and Fearing [13, 15]. Rather than employing a linear approximation, they found a set of exact trajectories of the nonlinear equations lying on a one dimensional manifold defined by the equation

$$2\theta_1 - \phi = \kappa \tag{6.11}$$

where κ is some constant. This results in periodic motion about the intersection of this manifold and the inverted equilibrium manifold. Given a judicious choice of κ and the physical parameters of the robot this controller can generate motions suitable for hopping. The main difficulty with the resulting hopping motions is that the acrobot leaves the ground with a significant angular momentum. To circumvent the problem, Berkemeier and Fearing implemented a flight phase controller that rotates the internal angle an integral number of times (e.g. once) so that the acrobot lands in a configuration suitable for subsequent balancing. In addition to hopping they also generated a sliding gait that shuffles the base of the acrobot along the ground.

Subsequently, Berkemeier and Fearing implemented their acrobot in hardware[14], using a clever design that rests the acrobot on an inclined table supported by a cushion of air. The effect of inclining the table is to scale down the force of gravity and hence slow down the system's dynamics. In this way they were able to build the acrobot from cheap, lightweight components. They reported positive results for the sliding gait, but the hopping behaviour was not implemented because of technical difficulties.

Berkemeier and Fearing also compared their controller to a pseudolinearizing controller for tracking fast inverted trajectories [16]. Given suitable physical parameters for the acrobot they found that their controller produced significantly less error when tracking periodic trajectories with a frequency around 1Hz.

Miyazaki et al used a similar approach to create a hopping controller for the acrobot [51]. They were able to control the angular momentum of the system at take off.

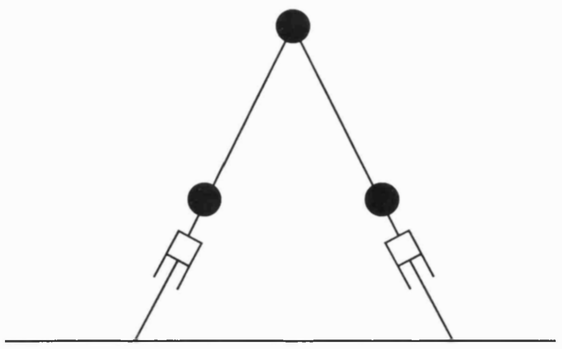


Figure 6.7: compass gait acrobot with massless telescopically retractable lower legs

Another approach to controlling the acrobot is the decision tree method, that was described in section 2.3, of Huang and van de Panne [44, 43]. This method was successful in producing balancing, hopping, cartwheeling and flipping behaviours, but is not well suited to real-time animation or control because of the computational cost.

Boone applied a similar decision tree method to swing-up control of the acrobot, to a near stationary, vertically upright position [18]. He reports good results using an evaluation function that first seeks to increase the energy of the acrobot and then tries to move closer to the goal.

Popa and Wen describe a variation upon a trajectory optimisation technique that iteratively updates the planned trajectory as control is executed [69]. The algorithm employs a Newton-Raphson method to warp an initial trajectory to an acceptable one. It is fast enough to be executed in real time, since it is not necessary to run the optimisation to convergence at each stage. They apply this method to balancing the acrobot in the vertically upright position.

A completely different behaviour that can be achieved with a planar acrobot is a simple walking motion. For this, the two links of the acrobot behave as a pair of legs and the middle joint becomes the hip. Such a walking motion is called a compass gait. Taking inspiration from research on passive dynamic walkers [28], Goswami et al studied the passive dynamics of the acrobot in a compass gait [3]. One problem with the compass gait is that the swinging foot will not clear the ground. Goswami et al sidestepped this problem by postulating a variation of the the acrobot with each leg having a telescopically retractable knee-joint with massless lower leg (shank). Figure 6.7 illustrates the configuration of their acrobot

They were able to show that this system can walk steadily down an inclined plane. Also, they derived a control law to regulate the energy in the system and thereby increase the overall stability of the walker somewhat.

6.5 Balancing the Acrobot with the HJB method

Before addressing the combined swing-up and balance problem we shall first address the construction of a balance only controller. In our experience there are a number of difficulties in generating a useful balance controller that need to be addressed before it is possible to generate the combined controller.

The first step is the choice of cost function. We have seen that continuous cost functions provide some advantages (section 5.1.1) over discontinuous ones. Therefore an obvious choice for the cost function would be

$$l(x,a) = \max_{\theta_1,\theta_2}\{h_{com}\} - h_{com} \tag{6.12}$$

where h_{com} is the height (y coordinate) of the centre of mass of the acrobot. An optimal policy for this cost function would maintain the acrobot in the vertically upright position. However, this is just a subset of the possible balanced configurations. We thus consider the following cost function

$$l(x,a) = |\theta_{com}| \tag{6.13}$$

where θ_{com} is the angle of the centre of mass from the vertical. An optimal policy for this cost function will maintain the centre of mass above the base of the acrobot. Note that an optimal policy need not keep the acrobot stationary, so long as any motion is along the equilibrium manifold. Therefore the set of points which maintain a minimum of equation 6.13 forms a two dimensional manifold in a four dimensional phase space. By contrast, for equation 6.12 that set is a single point, which happens to be a subset of that same manifold. Therefore it should be no more difficult to generate a controller using equation 6.13 than using equation 6.12, and will almost certainly form an easier problem. For that reason we shall first explore the problem of balancing with equation 6.13, before attempting to balance the acrobot upright.

6.5.1 Balancing on the Equilibrium Manifold

A controller was generated using a regular mesh, and the algorithm parameters shown in table 6.2, designed to permit balancing anywhere on the equilibrium manifold.

The solution converged after 48 iterations. The integral of the membership function across the domain was approximately 39% of maximum (where the maximum is defined by membership equal to one throughout the domain).

The controller was able to maintain balance from stationary starting points on the equilibrium manifold where $\theta_2 \geq \frac{\pi}{2}$. Balance was always achieved with a trajectory that stopped

Table 6.2: algorithm parameters

parameter	value
cost function	$l(x,a) = \lvert\theta_{com}\rvert$
physical parameters	m1,m2,l1,l2=1
domain coordinates	$x = (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$
domain	$[-\frac{\pi}{5}, \frac{\pi}{5}] \times [-\pi, \pi] \times [-\pi, \pi] \times [-2\pi, 2\pi]$
control set	$a \in -40, -30, -20, \ldots, 40$
mesh dimensions	$17 \times 17 \times 17 \times 17$
timestep	0.001
discount factor	0.01
boundary value	10
convergence threshold	0.01

moving near the upright, but folded over, configuration. In this folded over configuration the controller proved to be robust to external disturbances (small impulses applied to the acrobot) and always returned to approximately the same configuration. However for more upright starting points the acrobot always fell over. Figure 6.8 shows a typical sequence resulting in a fall. Note that the acrobot makes a stab for the vertically upright position before falling.

The membership function for this controller, figure 6.9, provides some insight into what is going wrong. Notice that membership is unity only at the upright and folded over balanced configurations. This explains why the system always makes for one of these two configurations. They are the only stationary points in the solution i.e. they are the only nodes in the domain where a trajectory exists that goes nowhere. Since they also have zero cost, their value (value function) is exactly zero, and their membership is exactly one. Notice also that the membership function forms a sharper spike at the origin (upright configuration) than at the folded over configuration. This spike indicates that the solution process is failing to discover trajectories in the neighbourhood of the spike that also stay within the domain (apart from the delicately balanced one on the top of the spike).

One possible explanation is that there is insufficient mesh resolution. However, simply using a regular mesh with double the resolution in each dimension (about as much as is currently practical) fails to solve the problem, and the acrobot still falls over. Perhaps if the additional nodes were targeted more carefully then that would solve the problem. Unfortunately, applying adaptive meshing to the problem, with anisotropic meshing and the influence metric, also failed to solve the problem. It took longer for the acrobot to fall over, but it still fell over.

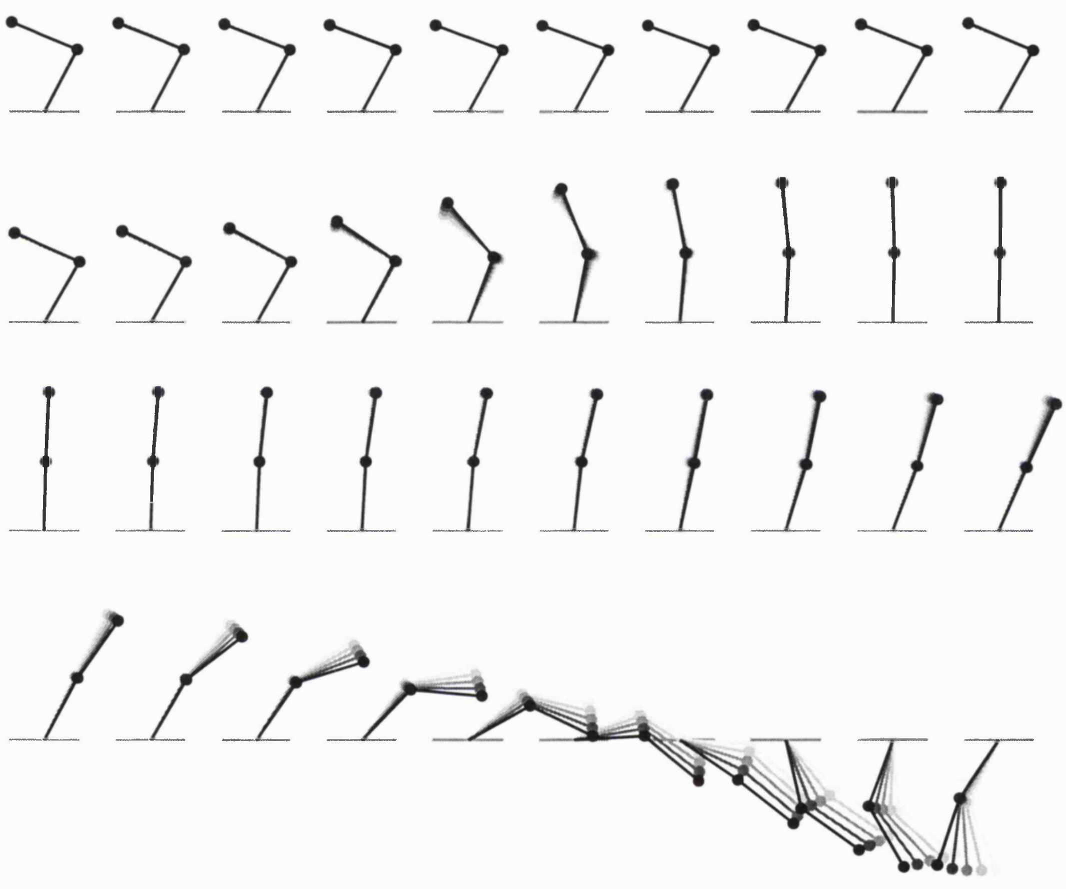


Figure 6.8: acrobot falling over

Frames are drawn 0.1s apart with motion blur effect achieved by drawing preceding frames at 0.02s intervals and fading out. The starting point was on the equilibrium manifold with $\theta_2 = \frac{3\pi}{8}$ and both joint velocities zero.

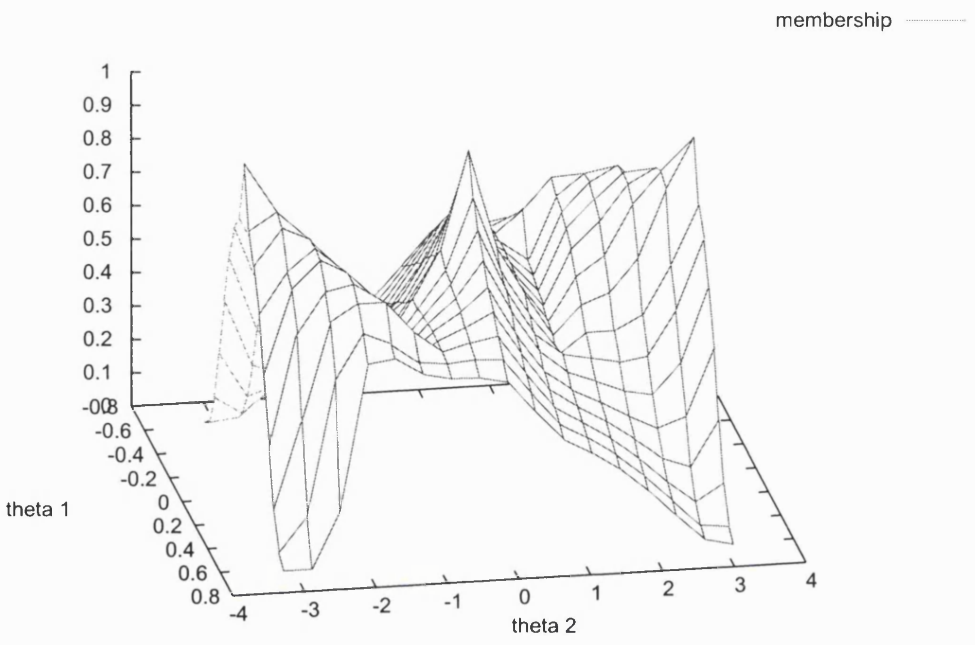


Figure 6.9: membership function for acrobot at $\dot{\theta}_1 = 0, \dot{\theta}_2 = 0$

So, what can be done to solve the problem? One obvious problem with the solution is that it is failing to discover static balance at any point on the equilibrium manifold. There are two reasons for this. Firstly, the mesh nodes do not fall exactly on the equilibrium manifold, except at two points. Secondly, the available controls do not provide exactly the torque required to maintain static balance on the equilibrium manifold, except at the same two points. We can address these problems by using a change of variables, and by introducing an additional torque to the system.

The state of the acrobot can be uniquely defined by the following alternative set of coordinates

$$x = (\theta_{com}, \theta_2, \dot{\theta}_1, \dot{\theta}_2). \tag{6.14}$$

where θ_{com} is the angle of the centre of mass to the vertical. Using this set of coordinates, there will be a line of nodes along the equilibrium manifold (17 such nodes with the 16^4 initial subdivision). If an additional torque is applied to the mid joint, given by

$$\tau_g = -l_2 m_2 \sin \theta_2 \tag{6.15}$$

such that the total torque applied to the mid joint is given by $\tau = \tau_g + a$ then, with zero control applied, the acrobot will maintain stationary balance at any point on the equilibrium manifold.

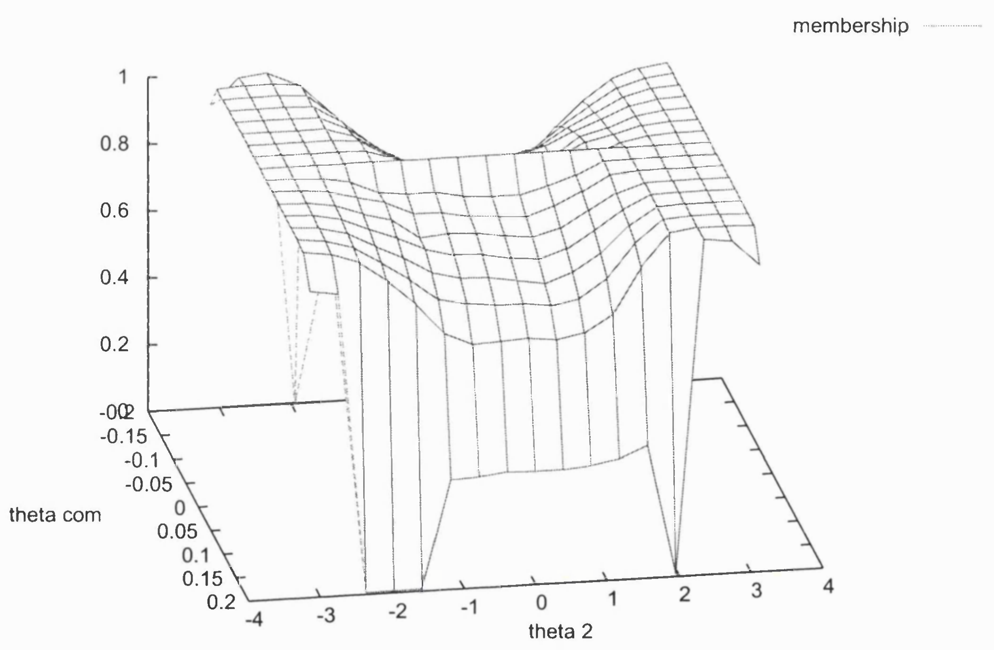


Figure 6.10: acrobot membership function at $\dot{\theta}_1 = 0, \dot{\theta}_2 = 0$ after first change of variables

These two changes have the effect of altering the problem, as it is presented to the HJB method, in such a way that certain desirable properties of the solution are easier to obtain.

Employing these two modifications and the following domain $[-\frac{\pi}{16}, \frac{\pi}{16}] \times [-\pi\, \pi] \times [-\pi, \pi] \times [-2\pi, 2\pi]$, the membership function (the static part of it) shown in figure 6.10 was obtained. The solution converged in 11 iterations and the integral of the membership function across the domain was approximately 65% of maximum. A substantial improvement upon the 39% membership obtained previously.

This is a better membership function. It now has a membership of one all along the equilibrium manifold (the horizontal ridge along $\theta_{com} = 0$) and there is a large area of unit membership around higher values of θ_2, i.e. the folded over configurations. However, the membership function does fall off quickly away from the equilibrium manifold at low values of θ_2.

The performance of the controller bears out these observations. The acrobot appears to be very stable near the folded over configuration. Unfortunately there are some stationary positions, with lower values of θ_2 and very near to the equilibrium manifold from which balance is not maintained.

A further change of variables, given by

$$x = (\theta_{com}, \theta_2, L, \dot{\theta}_2) \tag{6.16}$$

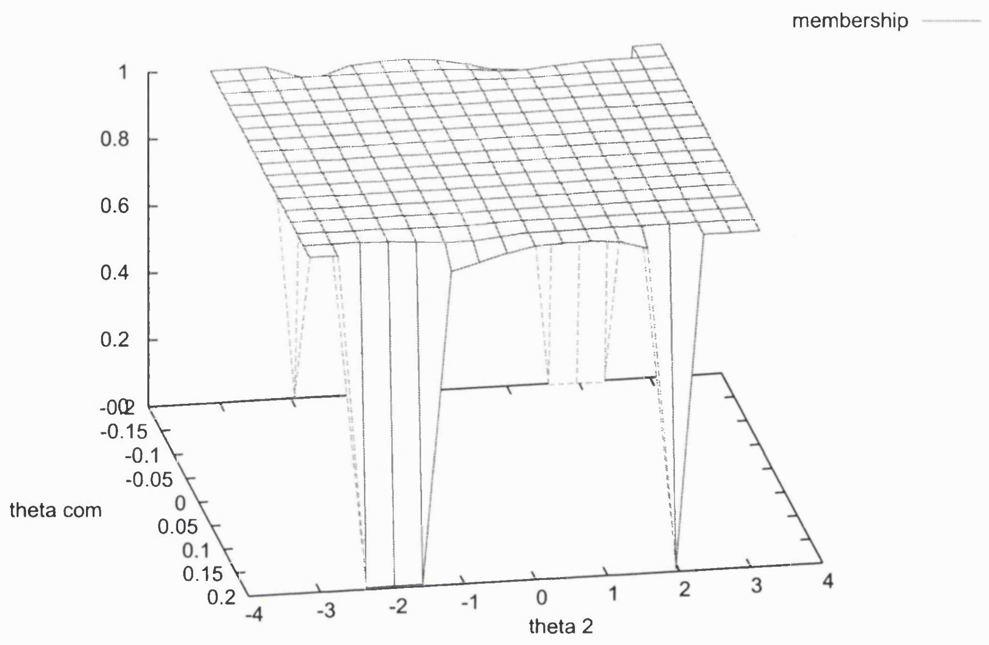


Figure 6.11: acrobot membership function after second change of variables at $\dot{\theta}_1 = 0, \dot{\theta}_2 = 0$

where L is the total angular momentum of the acrobot, brought a solution that converged in 14 iterations (the size of the domain was unchanged). This time the integral of the membership function across the domain was approximately 72%. The static part of the membership function is shown in 6.11.

This membership function looks better than the previous two because the majority of the static part of the membership function is one. Once again, the performance of the controller also improved. With this controller, the acrobot did not fall over from any point on the equilibrium manifold and remained balanced when small disturbances (impulses) were applied to it. Subjectively, the acrobot appeared to be very stable.

An interesting feature of this last controller is that the acrobot never came to rest on the equilibrium manifold but instead cycled steadily through the balanced configurations. This is consistent with the cost function, equation 6.13, which is zero on any part of the equilibrium manifold, whether moving or not.

It is not immediately obvious why this last change of variables, equation 6.16, improves acrobot controller performance. A part of the explanation for this may be that the choice of θ_{com} and L help to define a more focused domain for the solution. The acrobot can regain balance when its configuration is close to being balanced, which is true for low values of θ_{com}, but not

parameter	value
parameterisation	$x = (\theta_{com}, \theta_2, L, \dot{\theta}_2)$
domain	$[-\pi\, \pi] \times [-\pi, \pi] \times [-10, 10] \times [-13, 13]$
control set	$\{-40, 0, 40\} - l_2 m_2 \sin\theta_2$
initial subdivision	$17 \times 17 \times 17 \times 17$
subdivision limit	$16385 \times 16385 \times 16385 \times 16385$
subdivision method	anisotropic with influence
boundary value	20

Figure 6.12: table of parameters used for swing-up and balance

necessarily true for low values of θ_2. Likewise, the acrobot can avoid a fall when L is small, not necessarily when $\dot{\theta}_2$ is small. Hence the change of variables defines a domain in which balance is possible from a greater fraction of the points within it. However, despite many attempts, we have been unable to obtain a controller that was stable and robust to small disturbances from all static balanced configurations except by using the final parameterisation (6.16). Therefore this explanation, that the domain is more focused, may be insufficient.

Another possible explanation, may be that the final parameterisation (6.16) is close to being *differentially flat* [93].

However, the most likely explanation is related to the behaviour of the Acrobot on the manifold defined by $\theta_{com} = 0$ and $L = 0$. On this manifold the acrobot is in dynamic equilibrium because, although it may be moving, it has no component of motion outside of the manifold of static equilibrium positions. When this *dynamic equilibrium manifold* is aligned with the axes of the domain, we can be sure that a set of nodes will be placed directly on it. Furthermore, these nodes will all have the value zero because zero control keeps the acrobot on the manifold. With this parameterisation, the mesh is able to precisely capture the value function on the dynamic equilibrium manifold. Consequently, it is easier to find trajectories that take the acrobot to the dynamic equilibrium manifold because the solution procedure has perfect knowledge of the position of this manifold.

6.5.2 Swing-up and Balance

After having obtained a successful balance controller it was straightforward to then generate a swing-up and balance controller. Table 6.12 shows the algorithm parameters used for this purpose. All other parameters were the same as before, given in table 6.2.

In this example, the controller from the initial mesh failed to perform the swing-up, but after refining the mesh once (for a total of 110977 free nodes), an efficient swing-up manoeuvre

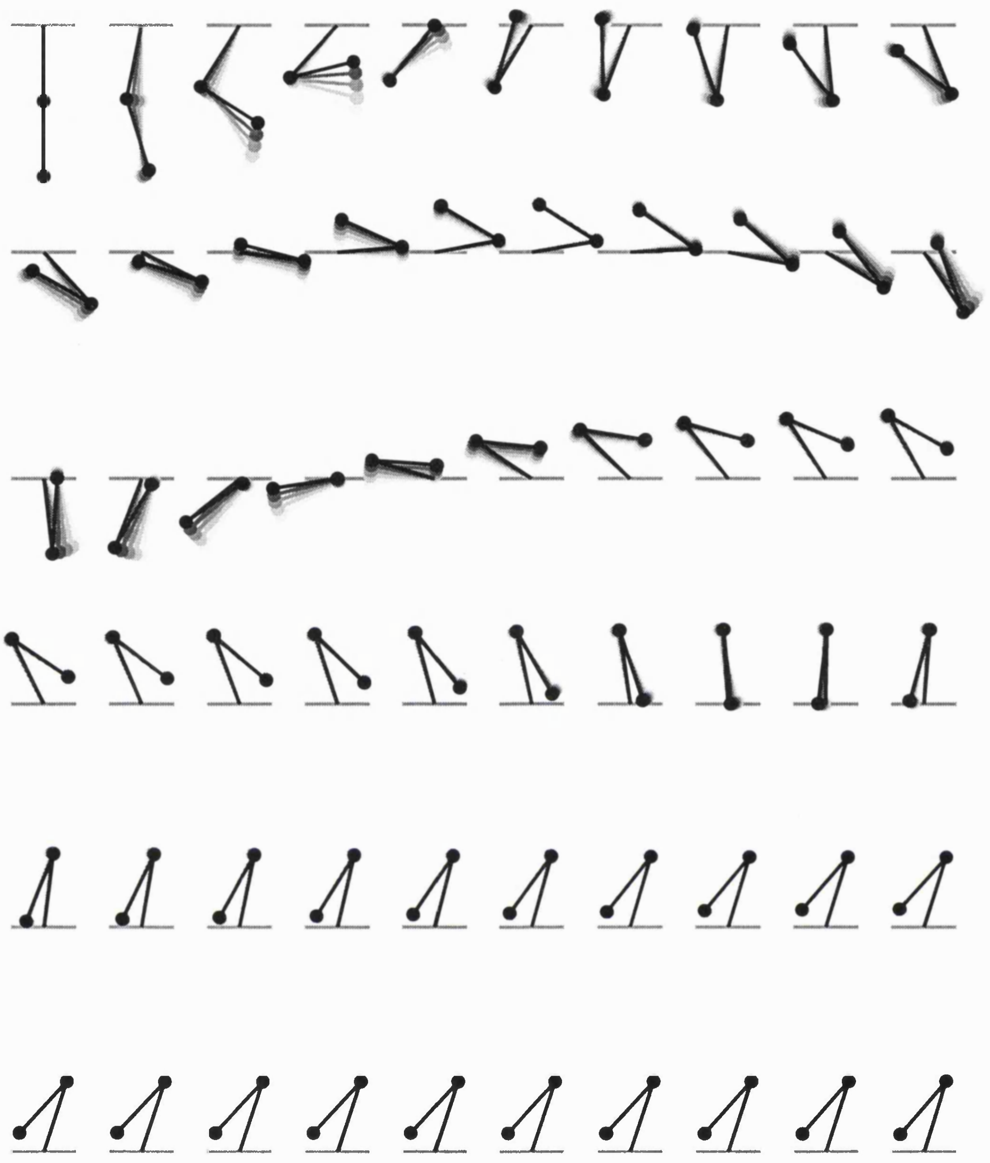


Figure 6.13: acrobot swing up and balance

was obtained. That manoeuvre is shown in figure 6.13. Once balanced, the acrobot settled into a stable oscillation around the inverted, but folded up, configuration, as can be seen in figure 6.13. When small disturbances (impulses) were applied to the acrobot, the disturbances always died away and the acrobot returned to the same balanced oscillation.

6.5.3 Balancing Upright

The task of generating a combined swing-up and balance controller, with balance in the upright, unfolded configuration ($\theta_1 = 0, \theta_2 = 0$), has proven to be more challenging than when the acrobot is allowed to balance anywhere on the equilibrium manifold. The balance is more delicate and requires high mesh density around the desired balanced configuration.

To reward upright balance, the following cost function was chosen

$$l(x,a) = 1.5 - z_{com} \tag{6.17}$$

where z_{com} is the height of the centre of mass of the acrobot. This cost function was chosen such that $l(x,a) = 0$ in the unfolded upright configuration. Figure 6.14 shows a sequence that was obtained using this cost function with all other parameters the same as in section 6.5.2. To achieve this sequence, the mesh was refined several times until it had nearly two million free nodes (1918348 free nodes). The swing-up manoeuvre was very efficient and required one less swing than the manoeuvre shown in figure 6.13. The balance of the acrobot was robust in the sense that its centre of mass remained above its base. However, it did not quite manage to remain in, or near, the unfolded upright configuration and settled into a limit cycle with the upper link swinging through one (nearly) complete revolution, alternately clockwise and anticlockwise, between positions on either side of the unfolded upright configuration. This was not the desired behaviour.

A better upright balancing behaviour, without the swing-up, was obtained by using the smaller domain from section 6.5.1 ($[-\frac{\pi}{16}, \frac{\pi}{16}] \times [-\pi\ \pi] \times [-\pi, \pi] \times [-2\pi, 2\pi]$). To further focus mesh refinement near the target, the error metric (subdivision criterion) was multiplied by a factor, $\hat{e}(u(x))$, given by

$$\hat{e}(u(x)) = \begin{array}{cc} 0 & u(x) > 1 \\ (1 - u(x)) & u(x) \leq 1 \end{array}. \tag{6.18}$$

This has the effect of focusing subdivision upon the region where the value function is less than 1, and of giving greater emphasis to accuracy near the target, where $u = 0$. Once the mesh had been refined to a stage where it had 267427 free nodes, a balancing behaviour was obtained in which the acrobot settled into a small oscillation very close to the unfolded upright configuration.

Finally, a satisfactory swing-up and upright balance controller was obtained by combining attributes of the previous two solutions. The larger domain was used, but with a finer initial mesh inside the region corresponding to the smaller domain. This special initial mesh was constructed by building a regular mesh of size 17^4 and then refining the mesh inside the region corresponding to the smaller domain by subdividing each cell four times in the θ_{com} dimension, twice in the L dimension, and once in the $\dot{\theta}_2$ dimension. This mesh had 151872 free nodes. The error metric multiplier given in equation 6.18 was also employed to focus subdivision close to the desired balanced configuration.

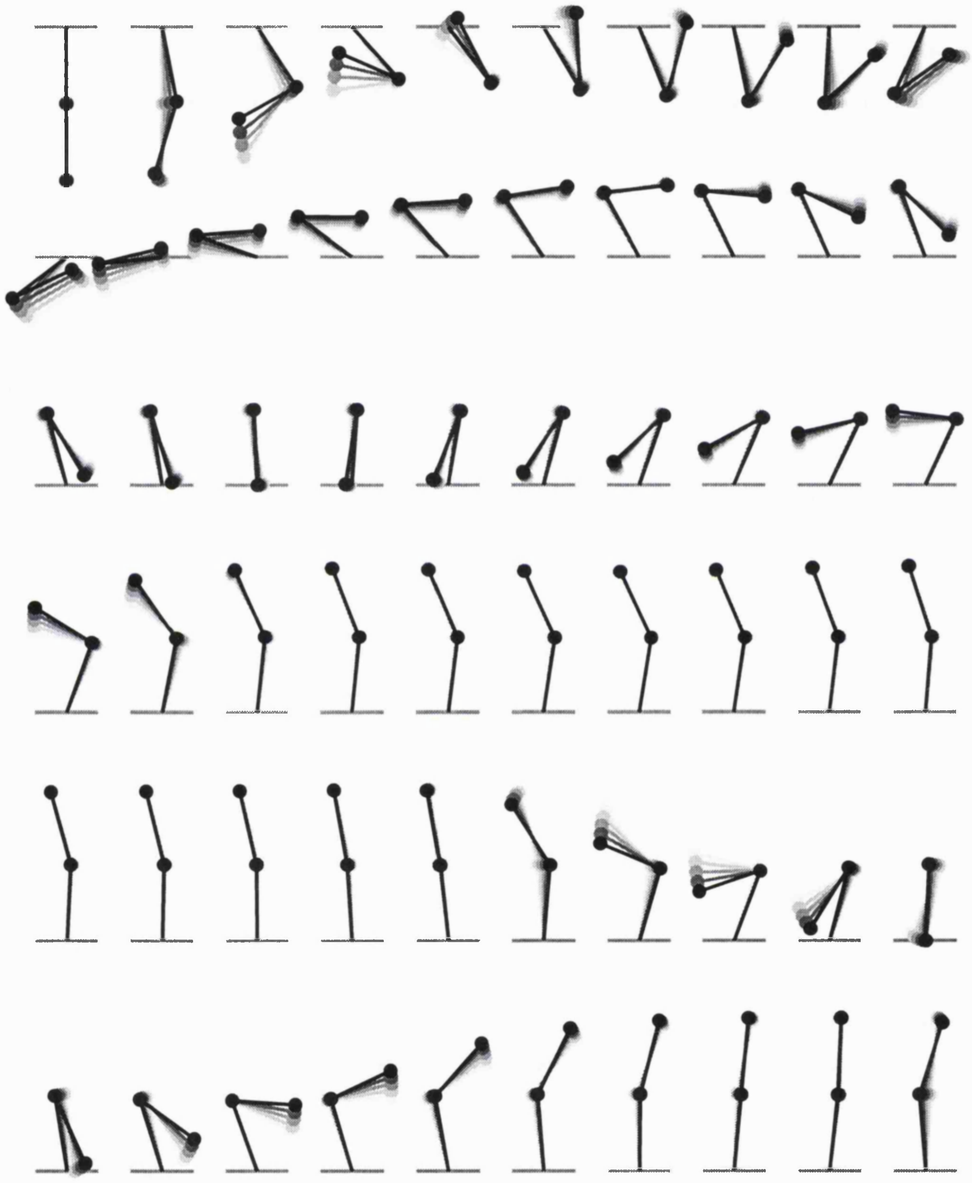


Figure 6.14: acrobot swing-up and upright balance

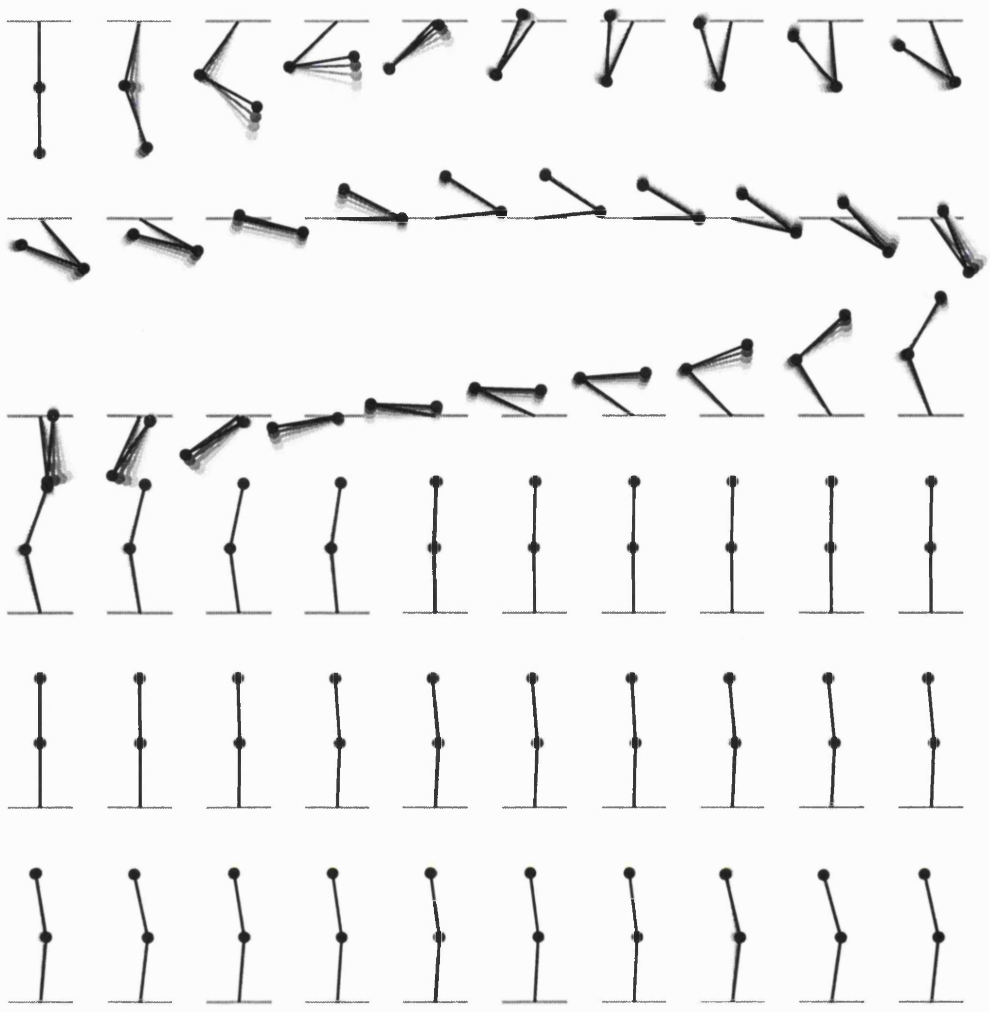


Figure 6.15: final acrobot swing-up and upright balance

Figure 6.15 shows a swing-up and balance sequence obtained after refining this special initial mesh to a stage where it had 512997 free nodes. The swing-up manoeuvre is less efficient than that shown in figure 6.14 (similar to the swing-up in figure 6.13) but now the acrobot settles into a small oscillation close to the desired upright balanced configuration.

6.5.4 Discussion

The major difficulty in obtaining a satisfactory controller for swing-up and upright balance for the acrobot lies in arranging for an effective distribution of mesh nodes within the domain. Unfortunately, the pointwise error metric that we have been using, even with the influence criterion, is not adequate to ensure that mesh nodes are distributed in the most effective way. One reason why the error metric is inadequate is that the value function is discontinuous. The discontinuity is present because, beyond a certain point, the acrobot cannot maintain balance and must swing around (with its centre of mass passing below its base) before regaining balance. The set of such points, beyond which the acrobot must swing around, form a three-dimensional manifold within the phase space of the acrobot. Optimal trajectories on one side of this manifold are not connected to optimal trajectories on the other side, hence the value function is discontinuous at this manifold[2]. The pointwise error is always high adjacent to this discontinuity and hence subdivision will proceed to the limit, unless suppressed by some means. The influence metric will not suppress subdivision at the manifold, unless the control function is unchanging at all points near the manifold, which is very unlikely. Consequently, the mesh refinement process will tend to place more and more free nodes near the discontinuity as the discontinuity becomes more well defined with the mesh.

Our solution, of providing a special initial mesh with higher mesh density around the balanced configurations, and of suppressing mesh refinement outside the region where $u(x) < 1$, serves to avoid these problems because the discontinuity lies outside the region where mesh refinement is permitted. However, this solution is not entirely satisfactory because it is not a general solution and because it fails to distribute new free nodes in the most effective way. What is required is a better method to determine where new free nodes should be placed. The method we have used, for sampling the pointwise error in the solution, is useful but is not perfect. For example, a high pointwise error associated with a small cell may be less significant than a low pointwise error associated with a large cell. The aim should be to distribute free nodes in such a way that we optimise the performance of the controller over the whole domain.

Despite these limitations, the controllers that we have obtained for the acrobot are robust and very efficient.

6.6 Summary

This chapter began by enumerating the decisions that must be taken when tackling a new problem with the HJB method. It is important to understand these decisions because they can be critical to obtaining a useful solution. Some general remarks were made about the significance of each decision and how it can be taken. The cart and pole example was used to illustrate the process of tackling a new problem.

The performance of the adaptive meshing technique was then compared against regular meshes using the cart and pole example. The comparison was performed with isotropic and anisotropic meshing both with and without the influence metric. The results indicate that adap-

[2]This discontinuity is present despite the fact that the system dynamics and cost function are continuous. It is a consequence of the cylindrical topology of the domain, where one boundary, $\theta_{com} = \pi$, is equivalent to another, $\theta_{com} = -\pi$, and the fact that the acrobot is under-actuated. Note that no such discontinuity arises because of the equivalence of $\theta_2 = \pi$ and $\theta_2 = -\pi$. In this latter case, the mid-joint actuator has sufficient strength to ensure that the acrobot controller is never forced to allow the upper to swing around.

tive meshes only provides significantly better performance than regular meshes when the influence metric is employed. Anisotropic subdivision provides marginally better performance than isotropic subdivision.

The HJB method was applied to balancing the acrobot in any balanced configuration. An effective controller for this task was obtained using an alternative parameterisation of the system. Using this alternative parameterisation, an effective controller was obtained for the task of swinging the acrobot up and balancing it in any balanced configuration. An effective controller was also obtained for the task of swinging the acrobot up and balancing it near the unfolded upright configuration, but this required special measures to ensure a sufficient density of mesh nodes near the desired balanced configuration.

Chapter 7

Conclusion

This work began as a search for a better way to generate animation for humanoid and animal figures. To guide the search, we noted that real animal motion is governed by Newton's laws of motion and the need to make efficient use of limited time and energy. This naturally led us to adopt an approach based upon Newton's laws of motion and optimal control theory. Compared to the established methods in computer animation, this approach has the potential to provide greater automation and flexibility in the process of creating lifelike animation.

Practical methods for solving Newton's laws of motion for articulated figures are well documented. However, practical methods for solving optimal control problems are less well understood and there is little agreement on the 'right' way to proceed. Open loop trajectory optimisation is one approach to optimal control problems. However, such methods are not obviously well suited to applications which require that animation be generated in real-time and in response to unpredictable inputs. In feedback form, the solutions of optimal control problems obey the Hamilton-Jacobi-Bellman (HJB) equation. Therefore we chose to pursue an approach based upon solving the HJB equation to generate optimal feedback controllers which can subsequently be deployed in an interactive animation system. The search for a practical method to solve the HJB equation, for the purpose of animation, then became the central research problem.

The difficulties that we encountered in obtaining an efficient and reliable method for solving the HJB equation were such that this became the major part of the work. The most fundamental of those difficulties is the exponential rise in computational complexity with the dimension of the problem; the so-called curse of dimensionality. This severely restricts the number of degrees of freedom of the dynamic systems that may be tackled. For a Newtonian system, where each degree of freedom has an associated velocity, the dimension of the problem is usually twice the number of degrees of freedom[1]. Consequently, we may be able to solve a problem

[1]This need not always be true, for example the cart-and-pole problem can be solved as a three dimensional problem (see section 6.1.5).

with two degrees of freedom (a four dimensional problem), but four degrees of freedom is likely to prove impossible. Furthermore, the curse of dimensionality can severely restrict the accuracy that may be obtained when solving a problem with even a modest number of dimensions. With one degree of freedom a brute force approach, using a high density regular mesh, will probably be sufficient. However, we have seen that a more sophisticated approach is required to obtain an accurate solution to the problem of swing-up and balance for the acrobot (section 6.5.2), which has just two degrees of freedom.

Our research focused upon designing an *efficient* method for solving the HJB equation. By focusing upon efficiency we were able to produce more accurate solutions to problems in a modest number of dimensions. The improvement was sufficient to provide a satisfactory solution to the swing-up and balance problem for the acrobot, which would not have been possible before. However, such efficiency improvements make little difference to the fact that we cannot tackle problems in higher dimensions. Consequently, we did not reach a stage where our method could be applied to animating figures with many degrees of freedom, such as any recognisably human or animal figure.

We have not succeeded in demonstrating that the HJB method is a practical method for life-like motion synthesis. Whilst this is, perhaps, disappointing it does not mean that the approach is necessarily flawed. We remain optimistic that our approach can be applied successfully to the task of generating lifelike animation. As we mentioned at the end of Chapter 1, there are two reasons to be optimistic. Firstly, it is often possible to decompose a high dimensional problem into a set of weakly interacting sub-problems of lower dimension. This is known as domain decomposition. For example, in the human body, the motion of an arm only interacts weakly with the motion of a leg. Perhaps the arm and leg could be solved as separate sub-problems. The literature on legged robots contains several examples of how such complex control problems can be decomposed into manageable sub-problems. Secondly, we can couple simplified dynamic models with kinematic animation methods, as in [70]. So, for example, a foot might be handled as a kinematic appendage to a dynamic model of a leg composed of only two parts: the upper and lower leg. The combination of domain decomposition and simplified dynamic models could be used to dramatically reduce the computational complexity inherent in applying our approach to the animation of human and animal figures.

The main part of this work, a practical method for solving the HJB equation, is largely independent of the particular problem to which it is applied. Although the systems that we have tried to control were governed by Newton's laws of motion, the method is equally applicable to systems that are governed by different laws. Thus our work is not simply applicable to animation, but could also find application in other fields such as control engineering or finance. One example of a quite different problem that can be solved using very similar methods is the shape-from-shading problem of machine vision [27].

7.1 Discussion of Contributions

As we stated in chapter 1, our main contribution is the design of an efficient method for solving the HJB equation for the purpose of animation. Alternatives are described covering many different aspects of the design of such a method. Those alternatives are assessed and compared and a choice is made based upon the available information. Several specific measures are proposed to improve the efficiency of the method. We now summarise these contributions, five of which were listed in Chapter 1 as the main contributions of the thesis.

1. The question of how to choose the timestep, h, in order to obtain both an accurate solution and rapid convergence is discussed in section 4.1. In section 4.1.3 we show that it is possible to obtain rapid convergence even whilst using a 'short' integration timestep, i.e. a timestep that is much shorter than the time required to traverse a single cell. This was achieved by calculating the limit of a series of updates to the value of a single node (the local iteration). Using this technique it is possible to de-couple timestep and mesh resolution without sacrificing convergence. Short timesteps allow for greater accuracy, but lead to slow convergence without this technique.

2. In section 4.1.4 we propose that the nodes be processed in increasing order of value during an update of the value function. We found that this improves the convergence rate, particularly when the value function has approximately the right shape initially, which is generally the case when adaptive meshing is used.

3. We compare the convergence rate with a single pass algorithm, where the value function and control function are updated simultaneously, and a two-pass algorithm, where they are updated separately, in section 4.1.5. We found that the two-pass algorithm converges slightly faster than the single pass algorithm.

4. A more efficient two-pass algorithm may be obtained by performing a number of consecutive value function updates for every control function update. We evaluate a number of rules to determine how many value function updates should be performed consecutively in section 4.1.5. We find that it was not necessarily efficient to perform the updates until the value function converged. The most efficient rule that we tested performs the updates either until convergence is obtained or the number of updates reaches a fixed limit or the

maximum change in the value function is less than some fraction of the maximum change during the first of the consecutive updates.

5. Section 4.2 addresses the question of whether an arbitrary boundary value corrupts the solution. We propose the membership function to quantify the effect of the boundary value upon the solution. We show an example where the membership function is one over most of the domain and hence we are able to conclude that the boundary value has no effect upon the majority of that solution.

6. A number of adaptive meshing schemes are compared for their performance on known analytic functions in section 4.4.5. The anisotropic, conformant, non 2:1 scheme is the most promising.

7. A method for control function extrapolation is proposed in section 5.1.3. We demonstrate that this method can be used to expand the reachable set obtained where cells straddle the boundary of the reachable set.

8. Interpolated control is compared to recalculated control, using the Bush problem on an adaptive mesh as a test case, in section 5.2. The conclusion we draw from this comparison is that interpolated control is the preferred method.

9. In section 5.3 we propose a subdivision criterion based upon a combination of value function and control function error. This criterion can be used to strike a balance between the accuracy of the two functions. However further work is required to find an effective way to determine that balance.

10. A new method for calculating an approximation to Munos' influence metric is proposed in section 5.3.1. Our method overcomes some difficulties with the original formulation, such as the question of how to assign influence to constrained nodes.

11. We compare various adaptive meshing schemes against a regular meshing scheme in section 6.2. We find that there is only a significant advantage when the influence metric is used. Anisotropic subdivision provides marginally better performance than isotropic subdivision.

12. We apply the HJB method to swinging-up and balancing the acrobot in section 6.5. We find that a useful controller can be obtained using an alternative parameterisation of the acrobot.

7.2 Further Work

The most general outstanding task is to fulfil the original goal of applying our method to the creation of lifelike animation. As has already been stated, this requires that the method be developed to a point where it can be applied, with confidence, to that task. What follows is a list of more specific research directions that, we believe, would help to achieve the general goal.

- Investigate the incorporation of domain decomposition methods, and the particular decompositions that may prove effective for humanoid animation.
- Investigate the use of simplified dynamic models coupled with kinematic animation methods, and the particular simplifications that may prove effective for humanoid animation.
- For animation purposes, it is not necessary to obey Newton's laws of motion to a high level of accuracy. One of the attractive features of the spacetime constraints method [111] is that it treats the laws of motion as inherently elastic. It would be interesting to investigate the question of how far the laws of motion can be 'stretched' before believability of the resulting animation suffers. In other words, how can we 'cheat' and still get away with it. One way to go about this is to invent actuators at joints where none should exist, but penalise their use. For example, with the acrobot model we might invent an actuator at the base joint, but heavily penalise its use (perhaps as the square of applied torque). A little judicious 'assistance' from the base joint actuator might go unnoticed, but could make the problem of balancing the acrobot considerably simpler to solve numerically.
- Conduct experiments to measure the ability of an observer to discriminate between natural and synthetic motion. This sort of 'Turing test for motion' could be a valuable tool to aid in the process of discriminating between alternative implementations of a motion synthesis system. In order to conduct such experiments it would be necessary to capture live motion and display it in an identical way to computer generated (synthetic) motion, so that the only perceptual difference between the two would be the motion itself. Note that it is not absolutely necessary to synthesise full humanoid motion in order to conduct such experiments. For example, one could capture the motion of a human on a pogo stick, and then display only the moving pogo stick. A computer model of a pogo stick rider could be considerably simpler than the real thing.

In addition to the topics above, there are a number of algorithmic improvements that could significantly improve the underlying HJB method.

- It may be possible to obtain a more efficient solution, i.e. with fewer nodes, by using higher order interpolation methods, e.g. quadratics or cubics. It is an open question as to how this may be implemented within a kd-tree based mesh.
- A higher order integration method should be used, as the Euler method is very inefficient for animating systems of articulated rigid bodies.
- A method for coarsening an existing mesh (removing nodes and subdivisions) should be implemented. As the solution evolves and more nodes are added to the mesh, the shape of the value function and control function change. Often this will have the result that the distribution of nodes in the mesh is not optimal and some nodes should be removed in order that more nodes can be added elsewhere. Where anisotropic subdivision is used, an ideal mesh coarsening scheme would also restructure the kd-tree, so that the order in which the dimensions are subdivided remains an efficient way to model the value and control functions.
- The question of how to find a good balance between control function accuracy and value function accuracy remains open (see section 5.3). This is an important question for the efficiency of the solution.

There are also some more interesting behaviours that can be achieved with the acrobot model that would present interesting and useful test cases for some of the items listed above. It would be interesting to develop a hopping behaviour for the acrobot and other behaviours that involve the acrobot leaving the ground. The key to this development may lie in choosing an appropriate decomposition of the computational domain. The dynamics of the acrobot are different when it is airborne than when it is on the ground. In particular, some of the acrobot's degrees of freedom are uncontrollable when it is airborne (motion of the centre of mass and total angular momentum). Therefore a domain might be constructed for the airborne phase that does not directly include those uncontrollable degrees of freedom, but instead uses some other parameter(s), such as time to impact.

It is possible to create a walking motion with the acrobot, without it leaving the ground, in a compass gait, as described in section 6.4.4. An obvious way to extend this into a three dimensional walking model is to decompose the motion into the sagital (forwards-backwards and up-down) and lateral planes, and disregard the interaction between motion in those planes. The three dimensional walking motion would therefore be composed of a compass gait coordinated with a lateral rocking motion.

Appendix A

Supplementary Animations

A CD-ROM containing supplementary animations accompanies this thesis and should be found inside the back cover.

Bibliography

[1] R. McNeill Alexander. *Optimal for Animals.* Princeton University Press, 1997.

[2] R. McNeill Alexander. *Energy for Animal Life.* Oxford University Press, 1999.

[3] Bernard Espiau Ambarish Goswami and Ahmed Keramane. Limit cycles and their stability in a passive bipedal gait. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 246–251, April 1996.

[4] Gideon Amos. Positioning human body models. Master's thesis, Deptartment of Computer Science, University College London, September 1997.

[5] Christian Babski and Daniel Thalmann. A seamless shape for hanim compliant bodies. In *VRML 99 Conference Proceedings*, 1999.

[6] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH 96 Conference Proceedings*, pages 137–146. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[7] David Baraff and Andrew Witkin. Physically based modeling: Principles and practice. `http://www.cs.cmu.edu/~baraff/sigcourse/index.html`, 1997. Siggraph course notes.

[8] David Baraff and Andrew Witkin. Large steps in cloth simulation. In Michael Cohen, editor, *Proceedings of SIGGRAPH 98*, Annual Conference Series, Addison Wesley, pages 43–54, 1998.

[9] M. Bardi, M. G. Crandall, L. C. Evans, H. M. Soner, and P. E. Souganidis. *Viscosity Solutions and Applications.* Springer, Lecture Notes in Mathematics, 1660, 1995.

[10] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal Control and Viscosity-Solutions of Hamilton-Jacobi-Bellman Equations.* Birkhauser, Boston, 1997.

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

[12] Richard Bellman. *Eye of the Hurricane.* World Scientific, 1984.

[13] Matthew D. Berkemeier and Ronald S. Fearing. Control of a two-link robot to achieve sliding and hopping gaits. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 286–291, 1992.

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

[15] Matthew D. Berkemeier and Ronald S. Fearing. Sliding and hopping gaits for the underactuated acrobot. *IEEE Transactions on Robotics and Automation*, 14(4):629–634, August 1998.

[16] Matthew D. Berkemeier and Ronald S. Fearing. Tracking fast inverted trajectories of the underactuated acrobot. *IEEE Transactions on Robotics and Automation*, 15(4):740–750, August 1999.

[17] Gary Boone. Efficient reinforcement learning: Model-based acrobot control. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 2, 1997.

[18] Gary Boone. Minimum-time control of the acrobot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, 1997.

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

[20] R. Boulic, R. Mas, and D. Thalmann. A robust approach for the center of mass position control with inverse kinetics. *Computers and Graphics*, 5(20), 1996.

[21] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. *Computer Graphics*, 23(3):233–242, 1989.

[22] F. Camilli and M. Falcone. Approximation of optimal control problems with state constraints: Estimates and applications. *I.M.A. Volumes in Applied Mathematics*, 78:23–57, 1996.

[23] Michael F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, 1992.

[24] James E. Cutting. Generation of synthetic male and female walkers through manipulation of a biomechanical invariant. *Perception*, 7:393–405, 1978.

[25] J.E. Cutting and L.T. Kozlowski. Recognizing friends by their walk: Gait perception without familiarity cues. *Bulletin of the Psychonomic Society*, 9(5):353–356, 1977.

[26] Scott Dyer, Jeff Martin, and John Zulauf. Motion capture white paper. `http://reality.sgi.com/jam_sb/mocap/MoCapWP_v2.0.html`, 1995.

[27] Maurizio Falcone and Charalampos Makridakis, editors. *Numerical Methods for Viscosity Solutions and Applications*. World Scientific, August 2001.

[28] Mariano Garcia, Andy Ruina, and Michale Coleman. Some results in passive-dynamic walking. In *Proceedings of Euromech Conference on Biology and Technology of Walking*, march 1998.

[29] Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 263–270, 1985.

[30] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, 1997.

[31] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of Siggraph '98*, 1998.

[32] Lars Gruene. An adaptive grid scheme for the discrete hamilton-jacobi-bellman equation. *Numerische Mathematik*, (75):319–337, 1997.

[33] Lars Gruene, Martin Metscher, and Mario Ohlberger. On numerical algorithm and interactive visualization for optimal control problems. *Computing and Visualization in Science*, 1(4):221–229.

[34] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: fast neural network emulation and control of physics-based models. In *Computer Graphics*, page 9=20, 1998.

[35] M. Hardt and K. Kreutz-Delgado. Modelling issues and optimal control of minimal energy biped walking. In *Proceedings of the 2nd International Conference on Climbing and Walking Robots, CLAWAR 99*, 1999.

[36] J. K. Hodgins. Simulation of human running. In Edna Straub and Regina Spencer Sipple, editors, *Proceedings of the International Conference on Robotics and Automation. Volume 2*, pages 1320–1325, Los Alamitos, CA, USA, May 1994. IEEE Computer Society Press.

[37] Jessica Hodgins. Three-dimensional human running. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[38] Jessica K. Hodgins. Animating human motion. *Scientific American*, 278(3):46–??, ???? 1998.

[39] Jessica K. Hodgins, James F. O'Brien, and Jack Tumblin. Do geometric models affect judgments of human motion? In Wayne E. Davis, Marilyn Mantei, and Victor Klassen, editors, *Proceedings of the Graphics Interface*, pages 17–25, Toronto, May21–23 1997. Canadian Information Processing Society.

[40] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 153–162. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[41] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 71–78. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[42] Honda. Honda humanoid robots, 2001. at http://world.honda.com/robot.

[43] Pedro S. Huang. Planning for dynamic motions using a search tree. Master's thesis, Department of Computer Science, University of Toronto, 1996.

[44] Pedro S. Huang and Michiel van de Panne. A planning algorithm for dynamic motions. In R. Boulic and G. Hegron, editors, *Computer Animation and Simulation '96*, pages 169–182. Springer-Verlag/Wien, 1996.

[45] G. Johansson. Visual perception of biological motion and a model for its analysis.

[46] G. Johansson. Visual motion perception. *Scientific American*, 232(6):76–89, 1975.

[47] Hyeongseok Ko and Norman I. Badler. Animating human locomotion with inverse dynamics. *IEEE Computer Graphics and Applications*, March 1996.

[48] J. Laszlo, M. Van de Panne, and E. Fiume. Limit cycle control and its application to the animation of balancing and walking. *Computer Graphics*, 30(Annual Conference Series):155–162, 1996.

[49] Michael A. Lee and Michael H. Smith. Automatic design and tuning of a fuzzy system for controlling the acrobot using genetic algorithms, dsfs, and meta-rule techniques. In *Proceedings of the First IEEE International Conference on Industrial Fuzzy Control and Intelligent Systems*, 1994.

[50] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical spacetime control. In *Proceedings of Siggraph 94*, 1994.

[51] Masanobu Koga Akiko Takahashi Masahiro Miyazaki, Mitsuji Sampei. A control of underactuated hopping gait systems: Acrobot example. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pages 4797–4802, December 2000.

[52] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9:62–82.

[53] Thomas B. Moeslund and Erik Granum. A survey of computer vision based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, March 2001.

[54] Joann M. Montepare, Sabra B. Goldstein, and Annmarie Clausen. The identification of emotions from gait information. *Journal of Nonverbal Behaviour*, 11(1):33–42, 1987.

[55] Joann M. Montepare and Leslie A. Zebrowitz. Impressions of people created by age-related qualities of their gaits. *Journal of Personality and Social Psychology*, 55(4):547–556, 1988.

[56] Joann M. Montepare and Leslie A. Zebrowitz. A cross-cultural comparison of impressions created by age-related variations in gait. *Journal of Nonverbal Behaviour*, 17(1):55–68, 1993.

[57] Remi Munos. A general convergence method for reinforcement learning in the continuous case. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the 10th European Conference on Machine Learning (ECML-98)*, volume 1398 of *LNAI*, pages 394–405, Berlin, April 21–23 1998. Springer.

[58] Remi Munos. A study of reinformcement learning in the continuous case by means of viscosity solutions. *Machine Learning Journal*, 2000.

[59] Remi Munos and Andrew Moore. Barycentric interpolator for continuous space and time reinforcement learning. In *Neural Information Processing Systems*, 1998.

[60] Remi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2):291–323, November 2002.

[61] Remi Munos and Andrew W. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1348–1355, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[62] Rémi Munos and Andrew W. Moore. Rates of convergence for variable resolution schemes in optimal control. In *Proc. 17th International Conf. on Machine Learning*, pages 647–654. Morgan Kaufmann, San Francisco, CA, 2000.

[63] Remi Munos and Andrew Moors. Influence and variance of a markov chain: Application to adaptive discretization in optimal control. In *IEEE Conference on Decision and Control*, 1999.

[64] Richard M. Murray and John Hauser. A case study in approximate linearization: The acrobot example. *UCB/ERL Technical Memo.*, 1991.

[65] Louise A. Obergefell, Thomas R. Gardner, Ints Kaleps, and John T. Fleck. Articulated total body model enhancements, volume 2: User's guide. Technical Report AAMRL-TR-88-043, Harry G. Armstrong Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, January 1988.

[66] Frederic I. Parke and keith Waters. *Computer Facial Animation*. A K Peters, 1996.

[67] Pixar. Luxo jr., 1986. film.

[68] Robert R. Playter and Marc H. Raibert. Control of a biped somersault in 3d. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 582–589, July 1992.

[69] Dan O. Popa and John T. Wen. Feedback stabilization of nonlinear affine systems. In *Proceedings of the 38th Conference on Decision and Control*, pages 1290–1295, December 1999.

[70] Zoran Popovic and Andrew Witkin. Physically based motion transformation. In *Proceedings of Siggraph '99*, 1999.

[71] J. Pratt, P. Dilworth, and P. Pratt. Virtual model control of a bipedal walking robot. In *Proceedings of the IEEE INternational Conference on Robotics and Automation (ICRA '97)*, 1997. Albuquerque, NM.

[72] J. Pratt and G. Pratt. Intuitive control of a planar bipedal walking robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '98)*, 1998. Leuven, Belgium.

[73] J. Pratt, A. Torres, A. Dilworth, P. Pratt, and G. Pratt. Virtual actuator control. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '96)*, 1996. Osaka, Japan.

[74] Jerry Pratt and Gill Pratt. Exploiting natural dynamics in the control of a planar bipedal walking robot. 1998.

[75] Jerry Pratt and Gill Pratt. Exploiting natural dynamics in the control of a 3d bipedal walking simulation. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR99)*, September 1999.

[76] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1988.

[77] Marc H. Raibert, H. Benjamin Brown Jr., and Michael Chepponis. Experiments in balance with a 3d one legged hopping machine. *The International Journal of Robotics Research*, 3(2), 1984.

[78] Leemon Baird Remi Munos and Andrew Moore. Gradient descent approaches to neural-net based solutions of the hamilton-jacobi-bellman equation. In *International Joint Conference on Neural Networks*, 1999.

[79] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioural model. In *Proceedings of SIGGRAPH 87*, pages 25–34, 1987.

[80] Charles Rose and Michael F. Cohen. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, pages 32–40, September/October 1998.

[81] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of Siggraph 96*, 1996.

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1:318–362, 1986.

[83] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[84] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 163–172. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[85] Robert Sedgewick. *Algorithms in C*. Addison Wesley, 1990.

[86] Isao Shimoyama. Dynamical walk of stilts type biped locomotion. *Bulletin of the JSME*, 26(215), 1983.

[87] Peter-Pike Sloan, Charles Rose, and Michael F. Cohen. Shape and animation by example. Technical report, Microsoft, 2000.

[88] Michael H. Smith, Michael A. Lee, and William A. Gruver. Designing a fuzzy controller for the acrobot to compensate for external disturbances. In *Proceedings of the 1997 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2264–2268, 1997.

[89] Michael H. Smith, Michael A. Lee, Mihaela Ulieru, and William A. Gruver. Design limitations of pd versus fuzzy controllers for the acrobot. In *Proceeings of the 1997 IEEE International Conference on Robotics and Automation*, pages 1130–1135, 1997.

[90] Warren D. Smith. A lower bound for the simplexity of the n-cube via hyperbolic volumes. *European Journal of Combinatorics*, 21:131–137.

[91] Mark W. Spong. Swing up control of the acrobot. In *Proceedings of the IEEE International Conference on Robotics and automation*, volume 3, pages 2356–2361, 1994.

[92] Mark W. Spong. The swing up control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1), February 1995.

[93] R. Stojic and C. Chevallereau. On walking with point foot-ground contact. In *Proceedings of the 2nd International Conference on Climing and Walking Robots, CLAWAR 99*, pages 463–471, September 1999.

[94] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* MIT Press, 1998.

[95] Frank Thomas and Ollie Johnston. *The illusion of life : Disney animation.* Hyperion, 1981.

[96] Nick Torkos and Michiel van de Panne. Footprint-based quadruped motion synthesis. 1998.

[97] Dorien van de Belt. *Simulation of walking using Optimal Control.* PhD thesis, University of Twente, Enschede, The Netherlands, April 1997.

[98] M. van de Panne, R. Kim, and E. Fiume. Virtual wind-up toys. In *Proceedings of Graphics Interface '94*, pages 208–215, May 1994.

[99] Michiel van de Panne. Motion synthesis for simulation-based animation. Master's thesis, Department of Electrical Engineering, University of Toronto, 1989.

[100] Michiel van de Panne. *Control Techniques for Physically Based Animation.* PhD thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, 1994.

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

[102] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion. In *6th Eurographics Workshop on Animation and Simulation*, 1995.

[103] Tzvetomir Vassilev. Dressing virtual people. 2000.

[104] Tzvetomir Vassilev and Bernhard Spanlang. Efficient cloth model for dressing animated virtual people. In *Proceedings of the Learning to Behave Workshop*, Enschede, Netherlands, 2000.

[105] Tzvetomir Vassilev, Bernhard Spanlang, and Yiorgos Chrysanthou. Efficient cloth model and collision detection for dressing virtual people. In *Proceedings of GeTech*, pages 89–100, Hong Kong, January 2001.

[106] Lawson Wade. *Automated Generation of Control Skeletons for Use in Animation*. PhD thesis, Ohio State University, 2000.

[107] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques, Theory and Practice*. Addison Wesley, 1992.

[108] Peter Whittle. *Optimal Control : Basics and Beyond*. Wiley, 1996.

[109] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 173–180. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[110] Andrew Witkin. homepage. `http://www.cs.cmu.edu/~aw/gallery.html`, 1995.

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

[112] Andrew Witkin and Zoran Popović. Motion warping. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 105–108. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[113] Wayne L. Wooten and Jessica K. Hodgins. Animation of human diving. *Computer Graphics Forum*, 15(1):3–13, March 1996.

[114] Zhan Xu and Xue Dong Yang. V-hairstudio: An interactive tool for hair design. 2001.

[115] Junichiro Yoshimoto, Shin Ishii, and Masa aki Sato. Application of reinforcement learning to balance of acrobot. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 5, 1999.

[116] Dongliang Zhang, Joao Oliveira, and Bernhard Spanglang. Animating scanned human models. *work in progress*, 2001.

[117] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, October 1994.

[118] Xinmin Zhao, Deepak Tolani, Bond-Jay Ting, and Normal I. Badler. Simulating human movements using optimal control. In *Computer Animation and Simulation '96 –*

Proceedings of the 7th Eurographics Workshop on Simulation and Animation. Springer Verlag, 1996.

[119] C. Zhu, R. H. Byrd, and J. Nocedal. L-bfgs-b: Algorithm 778: L-bfgs-b, fortran routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.