Taming Chaotic Circuits

by

Elizabeth Bradley

S.B. Massachusetts Institute of Technology (1983)
S.M. Massachusetts Institute of Technology (1986)

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

Doctor of Philosophy

in Electrical Engineering and Computer Science

at the
Massachusetts Institute of Technology
September, 1992

Signature of Author _____
Department of Electrical Engineering and Computer Science
September 3, 1992

Certified by _____
Harold Abelson
Professor of Computer Science and Engineering
Thesis Supervisor

and _____
Gerald Jay Sussman
Matsushita Professor of Electrical Engineering
Thesis Supervisor

Accepted by _____
Campbell L. Searle
Chair, Department Committee on Graduate Students

Taming Chaotic Circuits

by

Elizabeth Bradley

Submitted to the
Department of Electrical Engineering and Computer Science
on September 3, 1992 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Electrical Engineering and Computer Science

Abstract

Control algorithms that exploit chaotic behavior and its precursors can vastly improve the performance of many practical and useful systems. Phase-locked loops, for example, are normally designed using linearization. This approximation hides the global dynamics that lead to lock and capture range limits. Design techniques that are equipped to exploit the real nonlinear and chaotic nature of the device can loosen these limitations. The program *Perfect Moment* is built around a collection of such techniques. Given a differential equation, a control parameter, and two state-space points, the program explores the system's behavior, automatically choosing interesting and useful parameter values and constructing state-space portraits at each one. It then chooses a set of trajectory segments from those portraits, uses them to construct a composite path between the objectives, and finally causes the system to follow that path by switching the parameter value at the segment junctions. Rules embodying theorems and definitions from nonlinear dynamics are used to limit computational complexity by identifying areas of interest and directing and focusing the mapping and search on these areas. Even so, these processes are computationally intensive. However, the sensitivity of a chaotic system's state-space topology to the parameters of its equations and the sensitivity of the paths of its trajectories to state perturbations make this approach rewarding in spite of its computational demands.

Reference trajectories found by this design tool exhibit a variety of interesting and useful properties. *Perfect Moment* balances an inverted pendulum by "pumping" the device up, over half a dozen cycles, using roughly one-sixth the torque that a traditional linear controller would exert in this task. In another example, the program uses a detour through a chaotic zone to stabilize a system about 300 times faster than would happen by waiting long enough for the system to reach a regime where traditional control methods obtain. Chaotic zones can also be used to steer trajectories across boundaries of basins of attraction, effectively altering the geometry and the convergence properties of the system's stability regions. An externally-induced chaotic attractor that overlaps the phase-locked loop's original lock range is demonstrated here; the controller designed by the program uses this feature to extend the capture range out to the original lock range limits, allowing the circuit to acquire lock over a wider range of input frequencies than would otherwise be possible.

Thesis Supervisors:

Harold Abelson
Professor of Computer Science and Engineering

Gerald Jay Sussman
Matsushita Professor of Electrical Engineering

Contents

1 Introduction **1**

2 Overview **6**

2.1 The Problem 6

2.2 The Idea 6

2.3 How It Works 9

2.4 A Scenario 19

3 Basic Tools and Previous Research **27**

3.1 Dynamic Systems Theory 27

3.2 Chaotic Dynamics 31

3.2.1 Properties of Chaotic Attractors 33

3.3 Classical Control Theory 35

3.3.1 Traditional Linear Control: An Example 39

3.4 Controlling Chaos 43

3.5 Computational Methods 44

4 State-Space Mapping **46**

4.1 Cell-to-Cell Mapping 46

4.2 Trajectory Generation and Discretization 48

4.2.1 Timestep Selection and Trajectory Length 48

4.2.2 Selection of Starting Points 50

4.2.3 Discretization 51

4.3 Dynamics Classification 52
4.4 Parameter Spacing Between Portraits 57

5 Segment Selection and Path Synthesis **61**
5.1 Structure of the Search 61
5.1.1 Data Structures 64
5.2 Finding the Best Segment in a Stack of Portraits 65
5.2.1 Rough-Scale Filter 66
5.2.2 Fine-Scale Reconstruction 68
5.3 Search Region Selection and Refinement 71
5.3.1 On The First Pass 73
5.3.2 On Successive Passes 75
5.3.3 If the Search Fails 85
5.4 Tolerance and Termination 87

6 On-Line Control **92**
6.1 Navigating Along the Path 92
6.2 Stabilizing the System at the Destination 96

7 Examples **99**
7.1 The Lorenz System 99
7.1.1 Targeting and Stabilizing an Unstable Periodic Orbit 101
7.1.2 Targeting a Fixed Point 110
7.2 The Driven Pendulum 111
7.3 The Phase-Locked Loop 121
7.4 Summary and Applications to Reality 129

8 Caveats, Lessons Learned, and Future Directions **131**
8.1 Hard Problems 131
8.2 Easier Problems 133

9 Conclusion **137**

A How to use *Perfect Moment* **141**

A.1 Setting up the run 141

A.2 Specifying different optimization functions 144

B The Phase-Locked Loop **147**

List of Figures

2.1 Schematic of the path finding process 7
2.2 Segmented path 8
2.3 Top-level block diagram of *Perfect Moment* 10
2.4 Block diagram of the reference trajectory generation process 11
2.5 State-space portrait of the damped pendulum 12
2.6 Finding the core segment 14
2.7 Schematic of a partial path after two passes 15
2.8 Extracting a segment and setting up the next pass 16
2.9 Reference trajectory generation algorithm: synopsis 17
2.10 Segmented path with linearization recovery ranges 18
2.11 Search region refinement between passes 23
2.12 Simulation of controlled trajectory 25
2.13 Magnified view of controlled trajectory 25

3.1 A series of expansions of the Hénon attractor 34
3.2 Linear feedback control of an inverted pendulum 40
3.3 Pole selection for linear feedback control 42

4.1 The mapper's flowchart 47
4.2 Using shorter trajectories for local tasks 50
4.3 Discretizing trajectories 52
4.4 Trajectories that are misclassified 53
4.5 A period-one limit cycle 54

4.6 A stack of state-space portraits produced by the mapper 60

5.1 The path finder's flowchart 62
5.2 A rough-scale path between two points 63
5.3 A path and the calls that created it 63
5.4 Failure to find the best trajectory 67
5.5 Two-pass algorithm for segment selection 69
5.6 Approaches to segment extraction 70
5.7 Initial search region and discretization 71
5.8 Variable-resolution grid 72
5.9 Setting up the first pass 73
5.10 A possibly-problematic cell shape 74
5.11 Refining regions between search passes 76
5.12 Problematic search situations 77
5.13 The geometry of the variational system 79
5.14 Lorenz system derivative and variational system 81
5.15 Refining the grid 84
5.16 Error types at segment junctions 88
5.17 Tolerance regions and computations 90

6.1 Control at the segment junctions 93

7.1 Lorenz attractor for a = 16, r = 45, and b = 4 100
7.2 Poincaré section of Lorenz attractor 100
7.3 An unstable periodic orbit 100
7.4 Stabilizing an unstable periodic orbit 101
7.5 Two stable fixed points 101
7.6 Origin and destination points 102
7.7 Movement of fixed points with changing r 104
7.8 A false fixed point 104
7.9 Lorenz attractor for r = 30 104

7.10 Lorenz attractors for $r = 35$ and $r = 40$ 105
7.11 Lorenz attractors for $r = 56.25$ and $r = 60$ 105
7.12 Core segment 107
7.13 Schematized segmented path between **A** and **B** 109
7.14 Numerical integration of segmented path 110
7.15 Segmented path to a fixed point 111
7.16 Geometry of the simple pendulum 112
7.17 Movement of driven-vertex pendulum 113
7.18 State-space portrait of the simple pendulum 114
7.19 Driven pendulum Poincaré sections 114
7.20 Photographs of the experimental pendulum setup 115
7.21 Parameter-space plot for the driven-vertex pendulum 116
7.22 Controlling the driven-vertex pendulum 116
7.23 Inverted pendulum stabilized via parametric resonance 117
7.24 True dynamics for $\Omega = 15$ 118
7.25 Portraits for $\Omega =$ (a) 5 (b) 7.5 (c) 15 119
7.26 Results of *Perfect Moment*'s first pass 119
7.27 The controlled trajectory 119
7.28 A better controlled trajectory 120
7.29 Phase-locked loop block diagram 121
7.30 Time and frequency response of sample-and-hold 122
7.31 Linearized phase-locked loop block diagram 123
7.32 Lock and capture ranges 123
7.33 Sampled phase modulator 124
7.34 Phase-locked loop in chaotic regime 128

9.1 Photographs of the double pendulum 140

B.1 Schematic of the phase-locked loop circuit 149

List of Tables

5.1 Run time as a function of time step magnification 68

8.1 Run time on different computing platforms 134

B.1 Coefficient correspondences 148

Chapter 1

Introduction

This thesis presents a control system design methodology that actively exploits chaotic behavior. This tack not only broadens the field of nonlinear control to include the class of systems brought into vogue — and focus — by the last few decades of interest in chaos, but also opens a new angle on many old problems in the field. Many of these problems, new and old, are interesting and useful applications; all exhibit intricate and powerful behavior that can be harnessed by suitably-intelligent computer programs.

The algorithms presented here intentionally route systems through chaotic regions, using extensive simulation, qualitative and quantitative reasoning about state-space features and heuristics drawn from nonlinear dynamics theory to navigate through the state space. This mode of approach is a sharp contrast to traditional methods of control theory, most of which avoid chaotic behavior at all costs. As these regions often comprise a large and rich part of a chaotic system's state space, avoiding them constrains a system to a possibly small and comparatively boring part of its range of operation.

The program that embodies these algorithms constructs reference trajectories in advance, using a model of the target system. The controller thus designed is then used for on-line, real-time control of the system. During the generation of the reference trajectory, segments are selected from a collection of automatically-constructed state-space portraits and spliced together into a path that meets the specified control objectives. The on-line controller causes the system to follow this segmented path by monitoring the system state and switching the parameter value at the segment junctions. The effects are similar to those of gain scheduling, but the reasoning behind the "scheduling" is very different. Successful synthesis of such a trajectory is, of course, subject to dynamics-imposed limitations. A repellor[1] that persists for all allowable control parameter values cannot be made

[1] globally asymptotically stable in reverse time

reachable by this control technique.

In constructing and examining the state-space portraits, these algorithms use domain knowledge — rules that capture theorems and definitions from nonlinear dynamics — to choose the trajectory distribution on each portrait and the parameter spacing between portraits so as to make both the parts and the whole are representative of the system's dynamics: locally and globally, on both state-space and parameter-space axes. This synthesis is similar to the process that occurs when a nonlinear dynamicist selects and constructs overheads for a presentation or figures for a journal paper. Space and time are limited and the chosen information must be correct in amount, content and level in order to transmit the essential concepts without inundating the recipient.

This computer program, with very little human guidance or intervention, chooses a representative set of portraits by applying a multiple-scales dynamics classification method and then using the results of the analysis to adaptively determine the inter-portrait parameter spacing. This scheme allows the program to autonomously recognize and zero in on parameter ranges that encompass bifurcations[2] or other interesting behavior (i.e., an attractor expanding to cover the control objective.) The distribution of the trajectories on the individual portraits is guided by the system's sensitivity to parameter and state variations in each area of the space; the boundaries of the search region are adapted according to intermediate results of the search.

Nonlinearity can provide significant leverage to a control algorithm that is designed to exploit it. Since nonlinear systems are often exquisitely sensitive to their control parameters, a small range of parameter variation can give a program a large range of behaviors from which to choose. The distance between neighboring state-space trajectories can grow exponentially with time, so small trajectory perturbations can have serious global repercussions. The end results of this sensitivity resemble the paradigms of Maxwell's Demon and Simon's Ant, wherein environmentally-available energy is exploited via small, well-chosen control actions. The Voyager missions used Jupiter as a slingshot for just these reasons. Near the point of closest approach, a small change in angle, via a short, low-energy rocket burn, drastically changed the spacecraft's overall path in a fashion simply unobtainable in a linear system; kinetic energy was also imparted by the planet to the probe over the course of the interaction. Small *errors* in that course correction can, however, have equally dramatic effects. This leverage is the power of and, paradoxically, the difficulty with nonlinearity.

Chaos adds some advantages to this list. For example, the density with which chaotic trajectories cover a section of state space — the so-called strange attractor — has obvious implications for reachability. The now-classic "sensitive dependence on initial conditions" of trajectories on such attractors endows small control actions

[2] topological changes in an attractor

with large effects; at the same time, the structural stability of these structures in the presence of state noise endows the system with a measure of robustness. Furthermore, strange attractors contain an infinite number of unstable periodic orbits that can be located and stabilized.

The *goal* of this thesis is to identify and characterize some of these useful properties, to work out some computer control algorithms that take advantage of them, and to demonstrate their effectiveness on some practical examples.

The *task* is the classic control problem: to cause a system to travel from one specified state-space point to another in some *optimal* way, where optimal is defined by the user and the application.

The *agent* that performs this task is a Scheme[64] program called *Perfect Moment* that runs on HP series-300 and series-700 workstations. It should be viewed as a design assistant or apprentice: it can find reasonable solutions to most problems on its own and vastly accelerate design tasks for an expert control engineer.

The *domain of application* is the set of dissipative chaotic systems that have a single control parameter, are observable, and operate under well-specified design constraints. These algorithms can be applied to any system, but are designed to exploit specific properties of nonlinearity and chaos, so their specialized machinery is more or less wasted on linear problems. The practical examples demonstrated here are drawn from mechanical and electrical engineering, but chaos appears in virtually every branch of modern science and engineering, so potential applications are by no means sparse.

Striking results have been achieved with these techniques[15, 16, 17]: a very small control action, delivered precisely at the right time and place, can accurately direct the system to a distant point on the state space — hence the program's name. An equally-small change can be used to move from the basin of attraction of one distant fixed point to the basin of another. Control actions can briefly push a system directly away from the goal state in order to reach a globally-superior path to that point; "strange attractor bridges" can open conduits to previously-unreachable regions. All of these effects — leverage, movement between basins, counterintuitive moves, and bridges — are demonstrated in the Lorenz system, the first in-depth example in this thesis and one of the classic problems in the field of chaos. A trajectory that includes a section of a strange attractor steers the system through its chaotic region and stabilizes it about 300 times faster than would happen by waiting long enough for the system to reach a regime where traditional linear control methods obtain. The second example is the classic driven pendulum, balanced inverted; this example is given as a comparison point with traditional nonlinear control. For this specific example — one choice of system coefficients and run-time parameters — *Perfect Moment*'s trajectory balances the pendulum using roughly one-sixth the torque that would be required from a traditional linear controller.

The final example is drawn from another field that is rich in chaotic problems: nonlinear circuits. *Perfect Moment* is used in a point-to-region steering mode to find a single trajectory on a strange attractor that alters a particular type of phase-locked loop's global reachability properties and broadens its capture range out to the original lock range limits (a factor of 1.n to 1.m, depending on the target circuit's design.)

This work draws broadly on several fields for its techniques and examples: computer science, electrical engineering, mathematics, physics and control. None of the constituent pieces are considered leading-edge technology in the field in which they originated, but the cross-disciplinary hybridization of the methods is uniquely powerful. *Perfect Moment* uses AI techniques — multiple scales recognition, search, etc, — but it works on far more than toy problems. Domain knowledge from nonlinear dynamics and scientific computation techniques like mixed symbolic/numeric and algebraic/geometric computing broaden its range of applications and cut down on computational complexity. Powerful, special-purpose computers are used to further speed the process. The program's knowledge of — and lack of prejudice against — chaos extends its domain beyond that of classical nonlinear control techniques. For example, traditional linear control is used here where it is appropriate and useful; when it fails, global, more expensive computations are invoked. The higher-level reasoning made possible by the AI/nonlinear dynamics mixture lets the switch between the two modes be performed automatically. Where specific properties of chaos can be used, they are sought out and exploited by the same AI/nonlinear dynamics mixture. Of course, many combinations — pairs and threes — of these pieces have been used in the past, but never to the extent of the project described here.

This thesis reports on the design decisions that guided the development of the program, the structure and function that result, and the performance, extensions and drawbacks thereof. Chapter 2 begins with a high-level outline of how *Perfect Moment* works, which is followed by a scenario of how a design engineer might interact with the program in the course of solving a real problem. Chapters 4, 5, and 6 expand the brief outline in the first part of chapter 2, covering each stage of the analysis, synthesis and control processes in detail. Background material in mathematics and control theory appears in chapter 3, together with a review of current research in those fields and in computer science. Three examples are presented in chapter 7. Chapter 8 discusses caveats, derives limitations to the "domain of application" item in the list on page 3, suggests possible modifications that would loosen these limitations, and proposes future extensions of this thread of research. The conclusion is followed by a user's manual (appendix A) and schematics, drawings and tables that relate to the examples (appendix B.)

I have made no attempt to prove mathematically-rigorous results about many of the design choices and approximations discussed in this document. As a result, I have little evidence as to the range of applicability of this program. *Perfect Moment*

is a prototype of a working engineer's design tool. It uses heuristics extensively and chooses roughly-optimum solutions by balancing several simultaneous tradeoffs and diminishing-return situations. It does not hold out for truly optimal solutions, but rather concocts a "good enough" one as quickly as possible. As a result, proving controllability, for instance, is much more subtle than simply establishing that the appropriate matrix is of full rank. The results are highly dependent on the inputs; this variability is deemed acceptable because those properties are assumed to be fixed in the design environment under investigation.

The driving concept behind this approach to control of nonlinear and chaotic systems is to combine fast computers with deep knowledge of nonlinear dynamics to improve performance in a class of systems whose performance is rich but whose analysis is mathematically and computationally demanding.

Chapter 2

Overview

2.1 The Problem

Perfect Moment is presented with a nonlinear ODE, some control objectives (an origin, a destination, a tolerance and a specification of optimality cost,) a control parameter range, and a few other less-important inputs that allow the user to manage the program's execution time and to plug in real-world design and implementation constraints.

The program autonomously explores the system's behavior, manipulating the control parameter and the search region during its explorations, identifying and exploiting nonlinear and chaotic features and properties in the course of the process. Using the results of this exploration, it attempts to find a *controlled trajectory* between the origin and the destination. Finally, its real-time section executes the control actions that cause the target system to follow that trajectory.

Facilities for gathering data directly from an experimental system, rather than a possibly-inaccurate ODE model thereof, are under development and have been planned into this design.

2.2 The Idea

Envision a collection of state-space portraits, each constructed at a different control parameter value and plotted on a sheet of transparency film. If the system is richly nonlinear, portraits at slightly different parameter values may be very different; see the discussion in chapter 3 for more detail. Parts (a), (b), and (c) of figure 2.1 show state-space portraits of some two-dimensional nonlinear system at three different control parameter values. Trajectories are plotted in solid, dashed and dotted lines

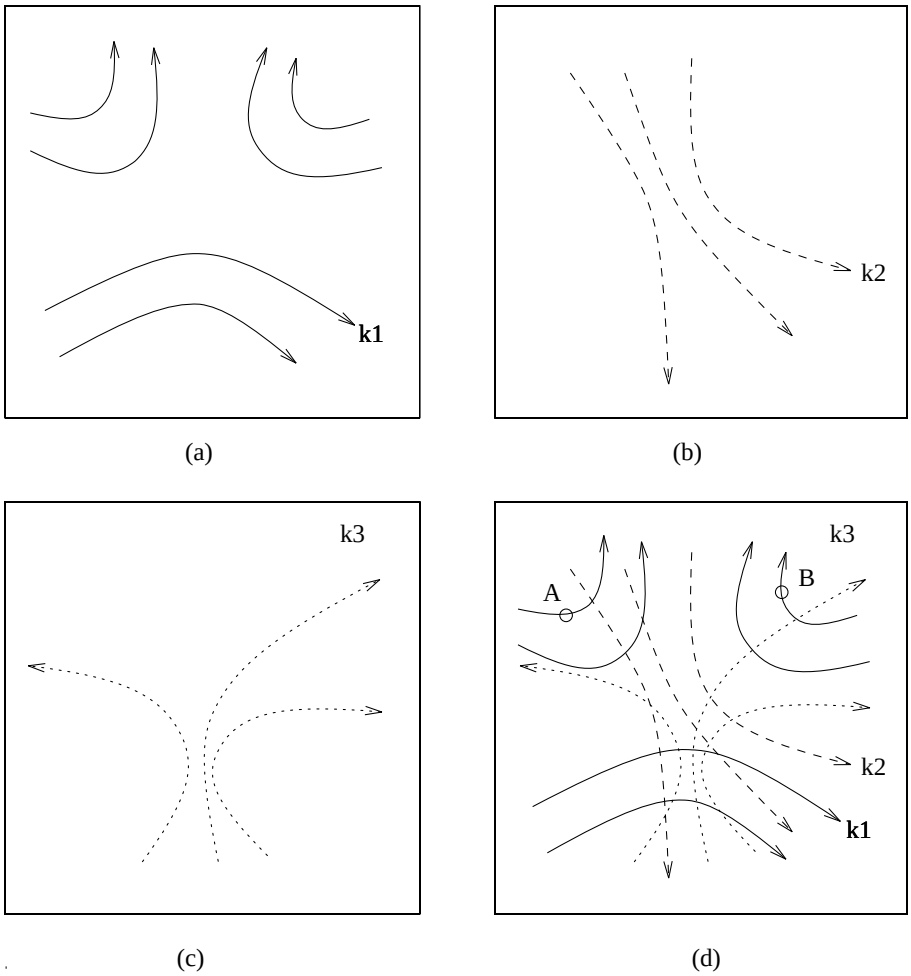


Figure 2.1: Schematic of path finding process. (a), (b), (c): state-space portraits of a nonlinear system at three different control parameter values (d) portraits superimposed for selection of path segments between **A** and **B**.

for the values k_1, k_2 and k_3, respectively.

If one were to stack these sheets on top of one another, as in part (d) of the figure, and look down through the stack, one could choose a set of trajectory segments that together form a path between **A** and **B**. See figure 2.2 for a simplified and expanded view of one possible path.

Note the switchpoint from k_1 to k_3 (solid to dotted) at the center right of figure 2.2 and refer again to parts (c) and (d) of figure 2.1. Because of the nonlinearity, nearby trajectories diverge and, if the timing of this switch is slightly off, the system will not approach point **B** at all. Small parameter changes can have equally profound effects on the overall trajectory if they take place in regions where the variational system grows rapidly with time, such as Voyager's periapse. These effects are unique to *nonlinear* systems; chaotic systems have several addi-

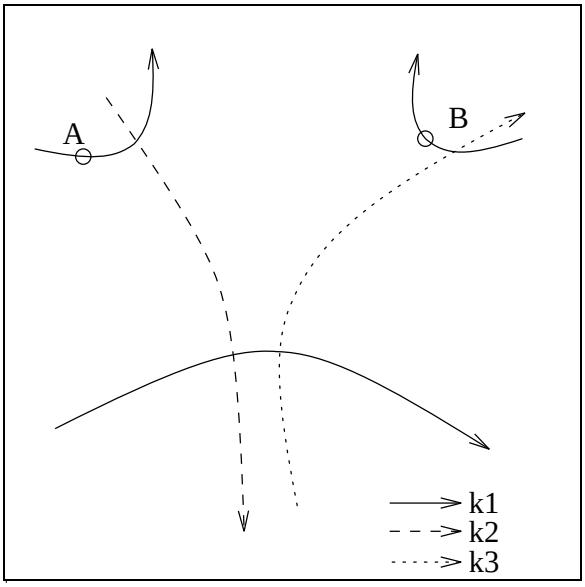


Figure 2.2: Segmented path between **A** and **B**, consisting of five segments selected from previous figure

tional useful properties that will be discussed in later chapters. Besides providing control leverage, this sensitivity — the source of *Perfect Moment*'s name — can also cause errors to blow up exponentially. Much of the intellectual effort involved in this thesis is devoted to walking the line between this leverage and the exponential error growth that it can engender, using nonlinear dynamics, control theory, scientific computation, AI, and many hours of CPU time to exploit the former and avoid the latter. This nonlinear amplification can also exacerbate the effects of the various difficult and fundamental problems that arise in real-world applications of these techniques: parameter and state measurement, system identification and modeling, etc.

The mapping and path finding tasks illustrated in figures 2.1 and 2.2 are relatively easy for a person and very difficult for a machine. Humans are good at reasoning from partial information and know when they reach the information overload threshold, so selecting a "representative" set of trajectories — one that is exactly enough to characterize the dynamics — is a natural task. Human vision is good at pattern recognition and at processing coincident information on different scales, so classification of portraits as "significantly different" is also intuitively easy. Coupled with nonlinear dynamics knowledge that permits interpretation, inference, and prediction of the effects of parameter and state changes, these information acquisition and processing skills cause the selection of a set of maps that are "representative" of the system's dynamics to be a straightforward task for a human expert. Finally, vision's multiple-scale image processing lets one quickly scan the maps for a "good" set of path segments that meet the control objectives

on both local and global scales.

The quotes in the last paragraph highlight the concepts that are difficult to mechanize. The next section gives a brief description of how *Perfect Moment* solves these problems; chapters 4-6 discuss each step in detail. The final section of this overview chapter illustrates the program's use with a (somewhat-optimistic) scenario.

2.3 How It Works

Perfect Moment's first action is to synthesize a reference trajectory. This task is performed by the algorithms lumped into the left-hand block in the diagram of figure 2.3. The information that describes this reference trajectory — the "recipe" — is then used by an on-line controller (the right-hand block in the figure) to cause the system to follow that path.

The double dashed line between the two blocks separates tasks that are performed off line and those that occur in real time: note that information only flows from left to right across this boundary. Improvements in computer technology (speed and accuracy) will allow this boundary to be moved to the right. Ultimately, the on-line controller will pass information back to the reference trajectory generator in real time, allowing the controller to adapt to poor models or systems where a parameter varies over time.

The functions involved in the generation of the reference trajectory are broken down further in figure 2.4. The box marked **A** transforms the user's inputs into the form required by the internal components and performs some initial setup computations. The *mapper* (**B**) constructs state-space portraits. The *path finder* (**C**) searches these portraits for useful path segments. On the first pass, the inputs to the mapper are derived from the user's specifications. On successive passes, this information is based on what the path finder determines about the evolving partial paths.

Perfect Moment's mapping module takes an ordinary differential equation, a control origin and destination and several other inputs that specify the control parameter range, the geometry of the region to be explored, and the length and resolution for generation of individual trajectories. It maps the system's state space within that range and region, with particular attention to chaotic and other features that the path finding module is designed to exploit. The latter then synthesizes a segmented path between the origin and the destination, applying the specified optimality criteria to these portraits to select the component segments of the controlled trajectory. The program iteratively improves the trajectory's accuracy: it first finds a gross path between the regions surrounding the origin and destination, then refines this path until it meets the control tolerance. During

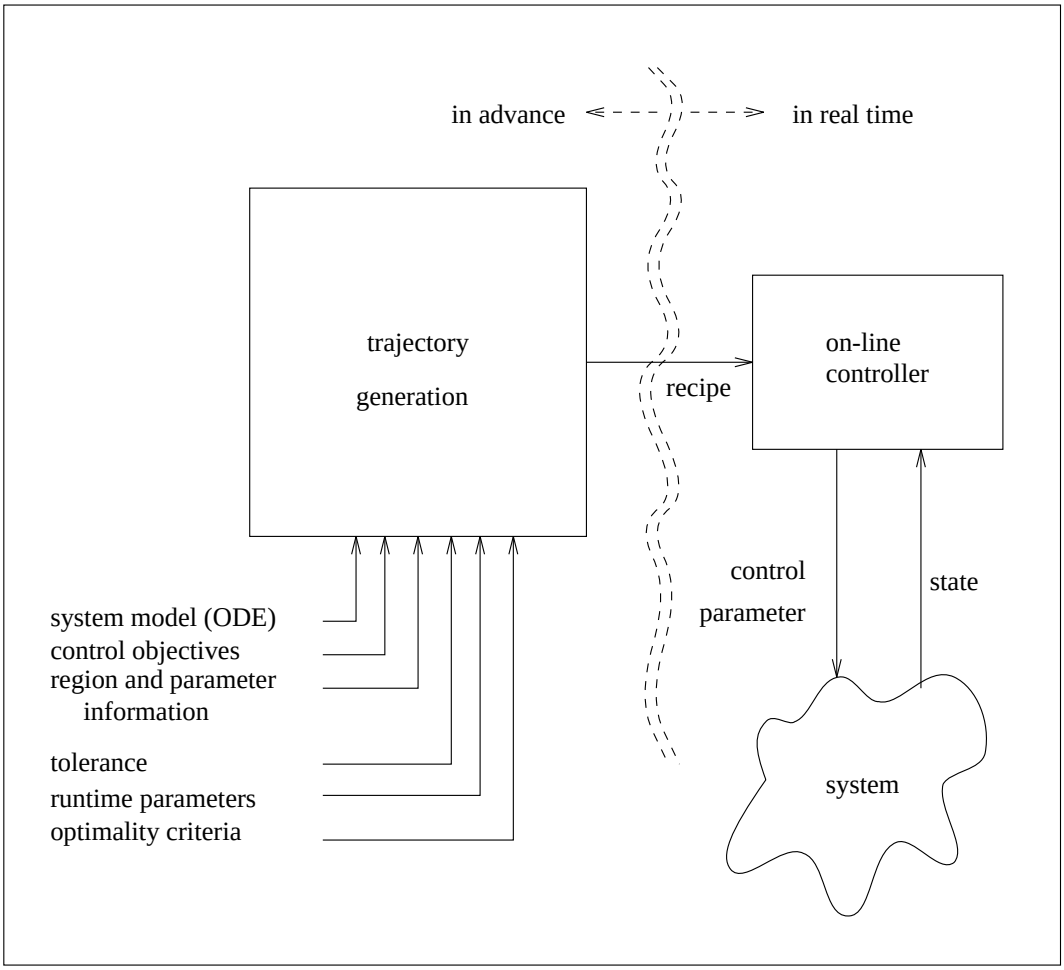


Figure 2.3: Top-level block diagram of *Perfect Moment*: reference trajectory generation is performed in advance and then executed by an on-line controller

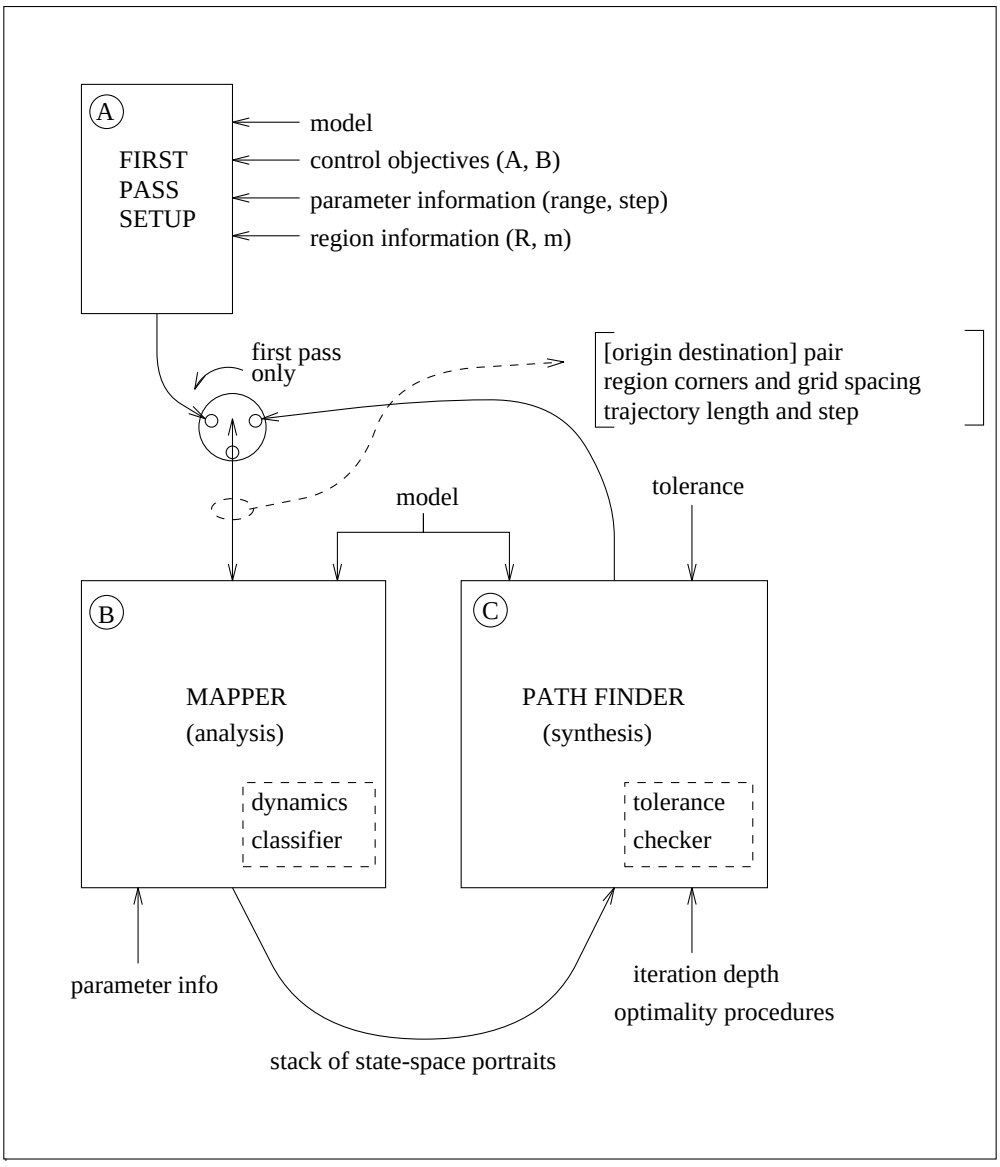


Figure 2.4: Block diagram of the reference trajectory generation process: the mapper produces a stack of state-space portraits and the path finder searches them for "good" path segments, according to the specified optimization criteria

Figure 2.5: A state-space portrait of the damped pendulum with a 3×3 grid. The horizontal axis is the angular position θ and the vertical axis is the angular velocity ω. *Perfect Moment* generates one trajectory from each cell, discretizes them into lists of cells touched, and synthesizes a rough dynamics classification for each

this refinement, the mapping and path finding processes are interleaved: based on information gathered in the latter, the former is repeated on a finer grain in regions of particular interest. This interrelationship gives rise to the cyclical loop between the boxes labeled **B** and **C** in the figure.

The state space is partitioned on a grid and trajectories are represented as lists of the cells that they touch, together with time of entry to and exit from each cell — a scheme derived from cell-to-cell mapping[41] and reminiscent of symbolic dynamics. Cells are represented as n-vectors: for three-space, `#(x y z)` in Scheme notation. A state-space portrait for a particular control parameter value, region of interest, and discretization contains one trajectory from each grid square, generated by numerical integration from the cell's centerpoint — or from the control origin and destination, in the grid squares that contain those points. See figure 2.5 for a portrait of a damped pendulum:

$$\frac{d^2\theta}{dt^2} + 2\frac{\beta}{\beta_{critical}}\frac{d\theta}{dt} + \sin\theta = 0 \tag{2.1}$$

with $\beta = 3 \times \beta_{critical}$.

After being discretized into lists of cells and times of entry, these trajectories are automatically classified according to their dynamics (the dashed box inside block **B** in figure 2.4.) For example, the trajectory starting at the "x" on figure 2.5 would be represented as the list

```
(((#(1 0) .002) (#(1 1) .100)
  (#(1 2) .284) (#(1 1) .424))
 (relaxing to the fixed cell #(1 1)))
```

The third line of this list is the classification of the dynamics. This particular trajectory is relaxing to a fixed cell, the discrete analog of a fixed point. The classification entry in each trajectory's data structure describes the type and location of the attractor to which it is relaxing. Identification of different attractor types is based on patterns in the sequences of cells in which the trajectories' points fall. A long terminal span in one cell identifies it as a fixed cell, a repeating, periodic sequence of cells is a limit cycle, and a nonrepeating, aperiodic, yet bounded set of cells is a chaotic attractor. These choices and definitions are discussed in much greater detail in section 4.3. This judgement is based only on information *at and above the scale of the current cell size*, and its results depend on that size. Because the path finder only recognizes and uses features on that scale, any finer-scale exploration would be moot. Moreover, this rough-cut method will work on exactly

the sort of noisy, experimental data that historically has posed severe problems for formal mathematical methods for dynamics classification (e.g., [1, 84].)

Perfect Moment is given an initial parameter range and step by the user; this step size is reduced automatically by the mapping module if successive portraits are significantly different — for example, if an attractor mutates (or *bifurcates;* see section 3.1) into a different type or approaches one of the control objectives. The dynamics-classification scheme described in the previous paragraph is used to determine when such an event occurs. A user-specified iteration depth limits the extent of this magnification.

The search region is first determined (block **A** in figure 2.4) by expanding the bounding box of the origin and destination by a heuristic overrange parameter that is initially specified by the user and then revised dynamically by the program over the course of its run. For example, one of the revision rules expands the region if no path is found and all trajectories exit the search region. The mapper (**B**) is then invoked, producing one portrait (like figure 2.5) for each automatically-chosen parameter value. The path finder (**C**) searches the discretized trajectories in the returned stack of portraits for the trajectory containing the best segment between the cell containing the origin and the cell containing the destination. Optimization criteria used to select this segment (the inputs at the bottom right of the figure) consist of user-specified Scheme procedures that operate upon time and state variables. The appropriate chunk of this trajectory is then extracted and used as the core of the ultimate segmented path.

The grid and the rules that manipulate it solve a variety of the "hard to mechanize" problems outlined in the previous section. To select a representative set of trajectories from among the uncountably-infinite number of candidates, the cell size is chosen small enough so that each region is explored and yet large enough to restrict the amount of information to the minimum necessary to capture the dynamics. *Perfect Moment* uses rules that stem from nonlinear dynamics theory in this choice; a very simple instance is the rule that forces coarser discretization in regions where nearby trajectories remain close to one another. The combination of multiple-scales dynamics classification and variable-step parameter exploration, both of which also depend intimately on the grid, serves as a reasonable first cut at obtaining a good sampling of the system's dynamics — in both state and parameter space. The grid is also used to control the flow of the interleaved mapping/path finding process. By first identifying a gross path and then refining it on finer scales — all the while only mapping and searching the regions that are necessary to do so, at the maximum possible granularity, as determined from the *local* dynamics — the program's effort is focused where it can be best used. Finally, in tandem with the overrange parameter, the grid allows *Perfect Moment* to find globally-good paths that have locally-bad segments (e.g., driving east to an airport to catch a westward flight) — the sort of path that purely-local control schemes miss. At any level of the search, moves within the current grid square that look locally bad will

Figure 2.6: Finding the controlled trajectory's core segment in a collection of state-space portraits of the damped pendulum; same system and axes as on the previous figure. Parts (a), (b) and (c) show portraits produced by the mapper for three different damping coefficients: (a) β_{crit} (b) 3 β_{crit} (c) 5 β_{crit}. The best trajectory segment from each portrait is extracted; all three are shown together on part (d). The best of the three (indicated with an arrow) is then selected and returned. A 4×4 grid division and a factor of three overrange factor were specified by the user for the region of interest; the parameter range and step were [1, 5] and 2, respectively. **o** = origin; **d** = destination.

be evaluated and then executed if they are judged useful.

Figure 2.6 illustrates one pass of the mapping/path finding process. These pictures were all produced by *Perfect Moment* during the course of a normal run; the system under investigation is the same pendulum model as in figure 2.5, but with variable damping. This simple system is not chaotic. It has stable fixed points at $[\theta, \omega] = [2n\pi, 0]$ and unstable fixed points at $[\theta, \omega] = [(2n-1)n\pi, 0]$ for all strictly-positive damping values. All trajectories on this figure are classified by the program as transients relaxing to one of the former: depending on the region of the initial state, one of the three cells `#(0 3)`, `#(1 3)`, or `#(3 3)`. Since the fixed points do not move or mutate over the parameter range, the mapper never explores the parameter axis more carefully, regardless of the specified iteration depth.

The "best" trajectory segments — for the purposes of this example and for the bulk of this thesis, those with the shortest euclidean state-space pathlength[1] — from the portraits of parts (a), (b) and (c) are isolated in part (d.) The metric used here to evaluate and extract segments actually uses both pathlength *and* distance between segment endpoints and control objectives; this combination and its implications are discussed in chapter 5. The same optimality cost functions are then used to select the best of these three segments — the one from the $\beta = \beta_{crit}$ portrait that is identified by an arrow in the figure.

• • •

After a core segment between the origin and destination cells is found, the entire mapping/path finding process is repeated within those cells — highlighted in figure 2.6(d) — to find two path segments, one on each end, that connect the ends of the core segment to **o** and **d**, shown schematically in figure 2.7. Four more shorter segments would be found on the next pass; in general, 2^n new ones are found at the nth pass. The program continues repeating the process in the gaps between segments until the specified control tolerance is met. At every level, the

[1]This distance "metric" is, of course, affected by the various scaling factors that are involved in the normalization and the region scaling

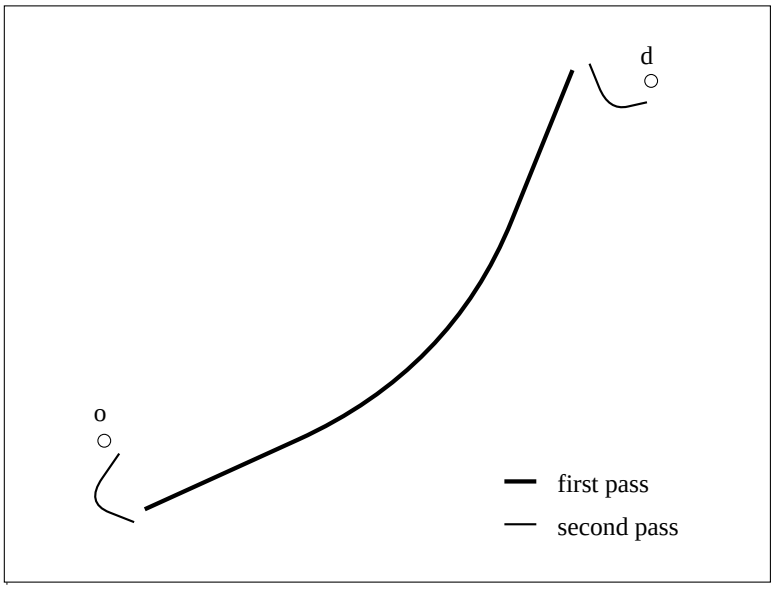


Figure 2.7: Schematic of a partial path after two passes: the core segment — found on the first pass – bridges most of the gap between the state-space points that are to be connected. Successive passes search for shorter segments that span the gaps in the path

mapping and search resemble that in the discussion of figure 2.6: *Perfect Moment* is always trying to connect pairs of state-space points.

The search region and the cell size are revised heuristically between passes. This focuses the exploration on finer scales in smaller regions, as dictated by the control requirements (e.g., tolerance, proximity to evolving partial paths) and guided by the local dynamics. The next-pass search region is simply the cell that surrounded the gap in the path, but shifted so as to be centered around the two points that define the gap that is to be traversed. See figure 2.8(b). Within that region, a new, smaller grid division is used. Factors that influence the cell-size reduction between passes include the fractal dimensions and Lyapunov exponents of chaotic attractors, integrations of the variational system from each segment's starting point, etc. For example, if the vector field at the top of figure 2.8 were much more turbulent than at the bottom, the lower dashed square would be subdivided on a much coarser grain than the top one.

If the search fails, it is repeated with increased accuracy and loosened bounds: finer discretization and larger overrange factor. Should this continue to fail, the program will blindly continue revising accuracy and bounds and repeating the search; at this point, the user should intervene to alter the parameter range and possibly the iteration depth. If all else fails, she should perhaps then change the physical system (different input, parameter, etc) and repeat the entire run.

After all segments are found, the program generates a *script* or *recipe*, which

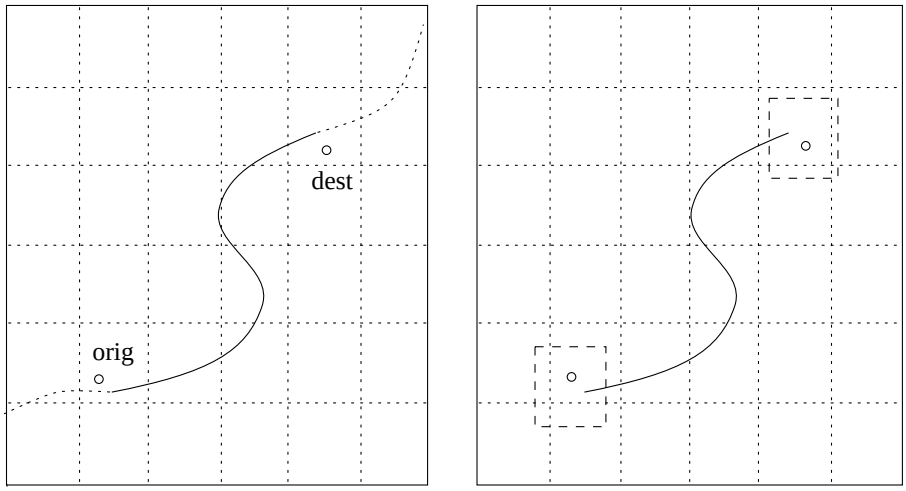


Figure 2.8: The core segment is extracted (a) from the best trajectory on the set of portraits; smaller search regions (b) are then computed and used on the next pass to find the shorter segments that connect this segment and the objectives

consists of the parameter values, starting and ending points of each segment, and the eigenvalues/vectors and sensitivity derivatives at each junction. This recipe is then passed to the on-line controller — the right hand block on figure 2.3 — which monitors the state and routes the system along the reference trajectory by appropriate switches of the parameter value at the segment junctions. Note that the segments are not followed in the order in which the program finds them.

Figure 2.7 is more typical[2] of normal paths constructed by the program than figure 2.2. It also conveys a better feel for the way the controller works as the path is being executed. One segment is used to do most of the work; shorter segments are used to connect to it. This process is reminiscent of inter-city route planning using a highway map: one chooses a segment of interstate and then connects to it on smaller roads, backtracking if appropriate. Once a path is found, the program spends most of the time monitoring the state and allowing the system to ride along on its own dynamics. It only intervenes occasionally, at the segment junctions, to change those dynamics.

The mapping and path finding algorithms described up to this point are summarized in figure 2.9. The next two paragraphs and chapter 6 discuss on-line control.

It is vital that the inter-segment parameter switches take place much faster than the target system's time scales, at exactly the right times, and with extreme accuracy. Speed is a formidable limit here, since control actions moderated by computer I/O must occur *at least* ten times[3] faster than the natural frequencies of

[2]The latter is possible, but only for a somewhat-pathological combination of inputs.

[3]The factor of ten is a rough rule of thumb for general low-Q feedback circuits[65], but may

1. Inputs:
   - System: ODE $f : \Re^n \rightarrow \Re^n$
   - Control objectives: origin **A**, destination **B**, tolerance T, $\{\mathbf{A}, \mathbf{B}, T\} \in \Re^n$; optimization criteria $g : Z^n \rightarrow cost$ and $h : \Re^n \rightarrow cost$.
   - Control parameter information: initial range (Δk) and interval $[k_{low}, k_{high}]$
   - Region information: overrange factor R_0 and number of intervals m_0.
   - Implementation-related parameters: iteration depth D_i
2. Expand the bounding box of **A** and **B** by R_0 to obtain search region.
3. Construct state-space portraits in this region at each $k_{low} + n\Delta k$ for $(k_{low} + n\Delta k) \in [k_{low}, k_{high}]$, using m_0 cells on a side.
   - Generate a trajectory from the center of each cell and from **A**.
   - Generate a trajectory in backwards time from **B**.
   - Discretize trajectories into lists of cells.
   - Classify the dynamics.
4. If the maps for k_i and k_{i+1} exhibit significant differences (e.g., attractors entering the destination cell, bifurcations, etc,) construct a portrait at $\frac{k_i + k_{i+1}}{2}$.
5. Repeat step 4 until neighboring maps are similar, up to $D_i - 1$ times.
6. Applying the optimization criteria to these maps, choose and extract the best trajectory segment between the cells containing the origin and the destination. This segment is designated S^0, starts at $S^0_{init} \epsilon \Re^n$ and ends at $S^0_{final} \epsilon \Re^n$ with $k = k_0$.
7. If no such segment exists, revise R and/or m and repeat from step 2.
8. Revise the cell size and the search region and repeat steps 3-6 for the pairs of points [**A**, S^0_{init}] and [S^0_{final}, **B**] to find the segments S^1 and S^2 that cross those gaps. Iterate this step on successively smaller scales between the segments' endpoints until the tolerance is met or the cell size falls below the machine epsilon.

Figure 2.9: Reference trajectory generation algorithm: synopsis

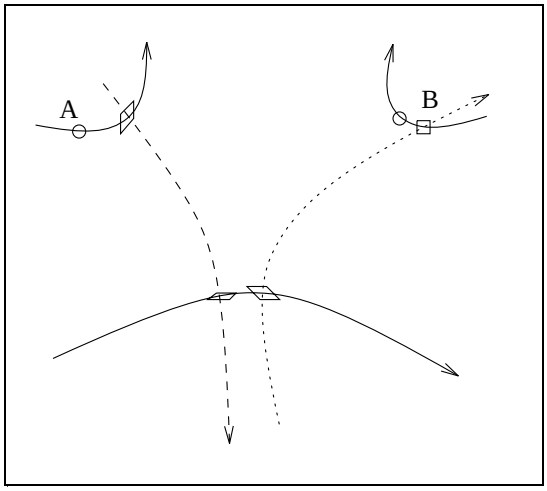


Figure 2.10: Segmented path with linearization recovery ranges: the on-line controller can recover from errors made at the segment junctions, but only within the regions where its linearization holds

the system under control. Quantization error, inescapable because of the discrete nature of computer arithmetic in a continuous world, coupled with the exponential rate of error growth, puts an upper bound on the attainable accuracy; some related issues are discussed in [24]. Modeling error is a more profound problem, as chaotic dynamics depend sensitively on parameter as well as on state. Small control tolerances over long paths are fundamentally unachievable in the face of large positive Lyapunov exponents, inaccurate computer arithmetic, slow I/O or bad models. These issues are the core of chapter 8.

Since *Perfect Moment*'s ultimate goal is the control of real physical systems, it uses an additional, autonomous control device in an attempt to correct for such errors — a simple local-linear controller programmed with the linearization and sensitivity derivative at each segment junction. The rationale here is that these points are the most likely locations for errors because the actuators switch there. The shape and size of the region within which this autonomous controller can perform error recovery is determined by the eigenvectors and eigenvalues of the system's Jacobians at those points. See figure 2.10. Once the destination is reached, stabilization is effected by local linear control, under the aegis of the same backdoor controller. With enough computational effort and a fast local-linear controller, this scheme could be extended across the entire length of each segment instead of just at the junctions. This extension is discussed in chapter 6.

Perfect Moment can produce a running commentary, complete with graphics, on its status, actions and choices. This feature is not simply a development aid. An experienced control engineer or nonlinear dynamicist can monitor this report and

be inadequate in general.

intervene to accelerate the process or push on some particular part of the design that might be fuzzy, ill-posed or otherwise hard to formalize or specify. Equally important, a less-experienced user[4] can learn about nonlinear dynamics, control, search, etc, from observing the program's actions.

Perfect Moment currently handles systems that have a single control parameter. Its domain could easily be broadened by extending the algorithms described here to a higher-dimensional parameter search space, but the run time of the mapping and search processes is exponential in the number of parameters, so this angle is not pursued in this thesis. Though the ideas and methods driven by the investigation of single-parameter variations are *probably* representative of those required in higher-dimensional spaces, some of the math may be different; see chapter 8. The search does not always succeed: though the control parameter adds a dimension to the space that can open conduits between previously-unreachable regions, some destinations may still be unreachable. Finally, since *Perfect Moment* currently depends on presimulation and accurate observations of the system state, it cannot be applied to systems where the state is neither directly nor indirectly observable, nor can it adapt to bad models or react to time-varying systems. None of the algorithms detailed in this thesis run in real time: many hours of CPU time are required for the integrations and searches. The intended use of this design tool is to find paths that will be followed many times or that are particularly important.

2.4 A Scenario

This section presents a scenario of an interaction between *Perfect Moment* and a engineer who is designing a controller for a system modeled by the equations:

$$F\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -y - x \\ x + ay \\ b + z(x - c) \end{bmatrix}$$

These equations were constructed by Otto Rössler[66] in an attempt to find the simplest vector field that recreates the chaos-inducing state-space folding action of the Lorenz equations([52] and section 7.1 of chapter 7.) This development is purely illustrative and is not meant to suggest that the Rössler equations are a practical model for any engineering system.

In the presentation that follows, the user's responses to the program's queries are prefaced by the symbol "==>;" clarifying comments are in plain text and computer interactions are in `typewriter` font.

[4] and occasionally even the program's author

```
Enter the vector of state variable names in the form
#(t name1 name2 ...):
==> #(t x y z)

Enter the name of the Scheme function that performs the system
derivative, using *parameter* to denote the control parameter:
==> rossler
```

which is defined elsewhere as:

```
(define (rossler state t)
  (let ((x (first state))
        (y (second state))
        (z (third state)))
    (list (- (+ y z))
          (+ x (* *parameter* y))
          (+ *rossler-b* (* z (- x *rossler-c*))))))
```

with:

```
(define *rossler-b* 2)
(define *rossler-c* 4)
```

Of the three variable coefficients, **b** and **c** are held constant and **a** is used as a control parameter.

```
Enter the control origin:
==> #(-1 -2 -1)

Enter the control destination:
==> #(2 -4 0.7)

Enter the desired optimization function:
==> shortest-path-between-two-cells

Enter the desired tiebreaker function:
==> minimum-sum-distance-tiebreaker

Enter the control tolerance at the destination
(vector of percentage allowable error on each axis):
==> #(1 1 1)

Enter the initial overrange parameter (default = 2):
==> 7

Enter the initial grid division (default = 5 cells/side):
==> 2
```

```
Enter the parameter range in the form '(low high):
==> '(0.3 0.41)

Enter the parameter step:
==> .025

Enter the iteration depth for the mapping process (default = 5):
==> 2
```

The control origin and destination are randomly-chosen points. The tolerance is $\pm[1\ 1\ 1]\%$ of the destination three-vector, so any value between `#(-1.98 3.96 -.693)` and `#(2.02 -4.04 .707)` is sufficient[5]. The given optimization and tiebreaker functions together minimize euclidean path length; the reasons behind the division of the cost computation into two separate functions are discussed in chapter 5. An overrange factor of seven happens to expand the search region to include the entire chaotic attractor; this value makes the pictures interesting and visually appealing — and causes this scenario to demonstrate the program's identification of and response to bifurcations — but it is certainly not necessary for the run. A two-cell-per-side division is a much coarser division than one would normally use, but the graphics are hard to understand if it is any higher. This coarseness has some interesting effects on the dynamics classification, which will be discussed a few pages hence. A parameter range of $[0.3, 0.41]$ might be selected to reflect a particular actuator's limits; a parameter step of 0.025 and an iteration depth of two require an actuator resolution of 0.0125 normalized units.

```
Do you wish to monitor the run? (g[raphics]/d[escriptive]/n[o]):
==> g

What two state variables shall I use as axes for display
purposes?  (e.g. '(name1 name4)):
==> '(x y)

(finding core segment...)

(beginning mapping process)
(mapping region from #(0 -10. 4. -6.1) to #(0 11. -10. 5.8)
   with 2 cells on a side)
(map: parameter = .3)
(map: parameter = .325)
(map: parameter = .35)
(map: parameter = .375)
(map: parameter = .4)

(zeroing in because of bifurcation between
```

[5]Units in DEQ models in the nonlinear dynamics literature are typically normalized beyond all recognition even if they do have physical significance, so one must be careful about what these numbers mean.

```
  (limit cycle at param = .375) to
  (chaotic attractor at param = .4))
(map: parameter = .3875)

(zeroing in because of bifurcation between
  (limit cycle at param = .375) to
  (chaotic attractor at param = .3875))
(iteration depth reached!)
```

Perfect Moment detects the mutation from limit cycle to strange attractor — a bifurcation — and automatically halves the inter-portrait parameter step in the range between them, up to `iteration-depth` times. The algorithm that classifies attractor types — the dashed box inside the mapper in figure 2.4 — relies on patterns in the cell sequences in the discretized trajectories and in described in detail in section 4.3.

The coarse state-space division causes the mapper to overlook a bifurcation. The system actually has a period-one limit cycle for $\mathbf{a} = 0.3$ and $\mathbf{a} = 0.325$, a period-two limit cycle for $\mathbf{a} = 0.35$ and $\mathbf{a} = 0.375$, and a chaotic attractor for $\mathbf{a} = 0.4$. However, the grid cells are so large that the period-one and period-two limit cycles are effectively indistinguishable. With a finer grid, this error would not occur[6].

The path finder now examines the trajectories on the first portrait:

```
(filtering discretized maps for best segment)

(searching parameter = .3 map)
(full trajectory set plotted in "white" on graphics window)
(origin and destination cells highlighted in magenta)
(metric of best segment from parameter = .3 portrait
   is .6987631193213509)
(best trajectory from this portrait shown in green - hit space to proceed)
```

The shortest trajectory — the one spiraling out from **o** — is highlighted in green on the graphics window. Several of the trajectories relax to the period-one limit cycle at the center of the picture and the others exit the search region, some quite quickly and all but two through the faces of the cube that are projected down onto the page. This picture is somewhat hard to understand because it is a 2D projection of a complicated 3D system. This complexity is at its worst at the stage when the best segments from all portraits are displayed. For brevity and clarity, this scenario will now switch to a terser report, as if "d" were selected instead of "g," but it will occasionally switch back to illustrate a point. In reality, such switches cannot happen midstream.

[6]Incidentally, these bifurcation points are slightly different from the (analytically-computed) ones in [78, fig12.4], perhaps because of numerical damping and other differences in integrator-induced dynamics (see section 3.1.)

Figure 2.11: The revised search region for the second pass: **x** is the destination; **o** is the endpoint of the core segment.

The search continues:

```
(metric of best segment from parameter = .325 portrait
   is .37690004259127996)
(metric of best segment from parameter = .35 portrait
   is .57024201718115)
(metric of best segment from parameter = .375 portrait
   is .9428828034128187)
(metric of best segment from parameter = .4 portrait
   is 1.3182527185509918)
(metric of best segment from parameter = .3875 portrait
   is 1.1290250050193757)

(segment found:
  (#(-.8839252429162614 -2.06407393764175 -.7480393971015914)
   #(1.9364798919639825 -3.9401437064785974 .6703285610222558)
   .325
   (#(-10. 4. -6.1) #(11. -10. 5.8)))
   #[compound-procedure 23]
   #[compound-procedure 24])
```

This text describes the core segment: the coordinates of the starting and ending points, the parameter value from the portrait on which it was found, the corners that were used in the search, and the local derivatives with respect to state and parameter. The corners are necessary for the grid-size management; this requirement is explained in section 5.3. These derivatives are evaluated at the segment's parameter value and endpoints during the tolerance computations. See chapter 5 for examples of their construction and use.

The program checks the path — which consists at this point of a single segment — against the tolerance and determines that the segment's endpoint

```
#(1.9364798919639825 -3.9401437064785974 .6703285610222558)
```

does not lie within the specified bounds. However, no additional work is required to "connect" this segment to the origin, as that is the initial condition from which it emanated[7]. *Perfect Moment* repeats the mapping and path finding in the cell `#(1 1 1)`, which contains the destination and the upper right end of the core segment, first shifting the region originally enclosed by that cell to be centered around the two points, as shown in figure 2.11. Note that the picture on figure 2.11 is *not* normally a part of the program's report.

[7]The segment starting point #(-.8839 ...) reflects a single integration step forwards.

```
(refining segmented path: pass 1)

(beginning mapping process)
(mapping region from
   #(0 -3.281760054018009 -.4700718532392987 -2.2898357194888717) to
   #(0 7.218239945981991 -7.470071853239299 3.6601642805111276) with
   3 cells on a side)

(map: parameter = .3)
(map: parameter = .325)
(map: parameter = .35)
(map: parameter = .375)
(map: parameter = .4)
```

The choice of 3 cells per side for the second pass depends on the turbulence of the vector field in the new search region; discussion of this issue is deferred until the background material is given in chapter 4.

The new search region cuts through the limit cycle, truncating its constituent trajectories, destroying the periodicity and making it no longer recognizable to the classification algorithms as a limit cycle. See the portrait below. The same thing happens to the chaotic attractor at $\mathbf{a} = .4$. In fact, *all* trajectories on *all* portraits simply go to the sink cell, so none of the system's bifurcations are identified and *Perfect Moment* never reduces the Δk. Again, the coarseness and the limited region affect the program's perception of the dynamics, but not to the point of interfering with the search.

```
(filtering discretized portraits for best segment)
(metric of best segment from parameter = .3 map
   is .09287347615094517)
```

Note that the points to be connected are so close that most of the trajectories on the figure above are not useful and all are unnecessarily long. The automatic choices for these starting points and lengths, which follow a global paradigm whose intent is to sample the dynamics and locate and identify attractors, constitute extra work in this case; only the first few points of a few of the trajectories are actually evaluated by the path finder. A solution to this is given in section 4.2.1.

```
(metric of best segment from parameter = .325 map
   is .09137466656489988)
(metric of best segment from parameter = .35 map
  is .08989234841513333)
(metric of best segment from parameter = .375 map
  is .08842693792615496)
(metric of best segment from parameter = .4 map
  is .08645249030362429)
```

Figure 2.12: *Perfect Moment*'s simulation of the controlled trajectory found for the Rössler example: the left-hand plot is an $x - y$ projection and the right-hand plot is an $x - z$ projection

Figure 2.13: A magnified view of the small rectangles in figure 2.12. A smaller timestep is used after the segment junction to make the switch visible.

```
(segment found:
 (#(1.9338060838277273 -4.007308541357479 .6882206397421609)
  #(1.9834875951711908 -4.001956758114882 .6970138534356772)
  .4
  #(-3.281760054018009 -.4700718532392987 -2.2898357194888717)
  #(7.218239945981991 -7.470071853239299 3.6601642805111276)
   #[compound-procedure 25]
   #[compound-procedure 26]))

(tolerance met!  the total path is

((#(-1 -2 -1)
  #(1.9364798919639825 -3.9401437064785974 .6703285610222558)
  .325
  #[compound-procedure 23]
  #[compound-procedure 24])
(#(1.9338060838277273 -4.007308541357479 .6882206397421609)
  #(2 -4 .7)
  .4
  #[compound-procedure 25]
  #[compound-procedure 26])))

do you want to test this controlled trajectory
(h[ardware]/s[imulation]/q[uit])?
==> s
```

See figure 2.12 for the graphics plot that *Perfect Moment* produces. The program would normally only display the left-hand plot in figure 2.12 — the $x - y$ projection that the user selected in the final instructions given to the program on page 21. The segment breaks are not visible on these plots because of the scale. The areas in the small rectangles are magnified on figure 2.13 — with a smaller timestep after the segment junction to make the breaks obvious. Note that the endpoint of the core segment was very close to but not quite within the tolerance region, so the second segment is far smaller. Also note the slight steering change, visible on the right-hand plot, that is due to the parameter switch from 0.325 to 0.4.

```
do you want to test this controlled trajectory
```

```
(h[ardware]/s[imulation]/q[uit])?
==> q
```

This example is somewhat contrived and very simple. The tolerance was loose and *Perfect Moment* was able to construct an adequate path after two passes, so the controlled trajectory has two segments and only requires the controller to perform a single parameter switch. The grid was so coarse that no trajectories besides those emanating from the origin and destination were ever used; thus, the path includes no counterintuitive moves and the entire gridding/overrange machinery is virtually moot. The program never fails to find a path and hence never adjusts the overrange or intervals. Incidentally, if the destination were *outside* the cells touched by the system's limit set for the parameter range [0.3, 0.41], no path segment could have been found to bridge the gap: the vector field simply doesn't go that way for any parameter value inside the range. The examples in chapter 7 put *Perfect Moment* through much more difficult tests.

Chapter 3

Basic Tools and Previous Research

This chapter reviews areas of dynamics, control theory, and computer science that are pertinent to this thesis. The first two sections discuss general and chaotic dynamics. Section 3.3 covers classical control theory and includes a worked-out example; section 3.4 ties together control and chaos. The chapter concludes with a very brief review of the corpus of computational methods upon which this work draws.

Much of this material is basic textbook knowledge. The rest is in the current literature, but spread across several research areas and communities. This chapter should be skimmed first and referred to (if necessary) later on. The primary intent of its inclusion here is to clarify notation, definitions, and derivations for readers who may be unfamiliar with or rusty on one or two of the areas involved. A secondary goal is to allow readers to understand, by way of comparison, how this research relates to their own fields of technical expertise.

3.1 Dynamic Systems Theory

By the implicit function theorem, the equations for an n-dimensional[1] dynamic system can locally be written

$$\frac{d\vec{x}}{dt} = \vec{F}(\vec{x}, k_1, \ldots, k_p, t) \tag{3.1}$$

where $\vec{x}$ is an n-vector of system state variables and $\vec{F}$ is a possibly-nonlinear function of the state $\vec{x}$, the parameters k_i and the time t. The set of points

[1] $n/2$ "degrees of freedom," to stretch the Hamiltonian use of the term somewhat.

$\{\phi_t(\vec{x}_0) : -\infty < t < \infty\}$ is called the *trajectory* through $\vec{x}_0$. If $\vec{F}$ does not depend on t, the system is said to be *autonomous*; a *nonautonomous* system of dimension n whose drive is periodic in T can be converted to an autonomous system of dimension $n+1$ by the addition of the equation

$$\frac{d\tau}{dt} = 2\pi/T$$

to the system above. Drive "frequencies" in the resulting equations are dimensionless ratios of true drive frequencies to natural frequencies. All nonautonomous systems treated in this thesis have sinusoidal drives.

Dynamic systems are often described or classified according to their steady-state behavior: the ω-limit set (or *attractor*) covered by their trajectories as $t \to \infty$, omitting the *transient* sections at the beginning of each. In a dissipative system, all trajectories in an open set — the basin of attraction — surrounding an *attracting* ω-limit set ultimately evolve to and then remain upon that attractor.

If $\vec{F}$ is linear and nonsingular, the system (3.1) has a single fixed point $\vec{x}_{eq}$ such that $\vec{F}(\vec{x}_{eq}) = \vec{0}$. This point is either globally asymptotically stable, in which case its basin is all $\vec{x} \in \Re^n$, or it is unstable. In either case, the steady-state behavior of such a system is identical throughout $\Re^n$ and the parameters k_i only affect specific attributes of the behavior: the position of the fixed point, the sign and magnitude of the eigenvalues and the directions of the eigenvectors.

If $\vec{F}$ is nonlinear, the situation is potentially much richer. The system (3.1) can have any number of ω-limit sets of several different types:

- fixed points
- periodic orbits
- quasiperiodic orbits
- chaotic attractors

Their basins of attraction partition $\Re^n$ and are often intricately interwoven; the system's "steady-state behavior" is unique only within a particular basin.

Fixed points in a *nonlinear* system have a wider variety of stability classes:

- globally asymptotically stable
- unstable
- asymptotically stable
- stable (in the sense of Lyapunov)

- metastable

Entries one and three differ only in the size of their basins (all of $\Re^n$ in the latter.) An unstable fixed point has n positive eigenvalues and is asymptotically stable in negative time[2]; a metastable one has at least one negative eigenvalue and is unstable in negative time. Limit cycles or periodic orbits repeat after k cycles. Quasiperiodic orbits contain a countable number of periodic functions; the simplest is where two incommensurate frequencies cover a torus: one governs oscillation around the axis of the ring (major orbit) and one parametrizes the winding through the center hole (minor orbit[3]). If $n > 3$ and $\vec{F}$ is nonintegrable, the system can exhibit chaotic behavior and so-called *strange attractors* — the topics of section 3.2.

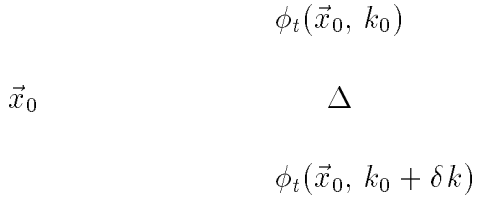
The direction and magnitude of the vector field described by the system (3.1) depend strongly on the equations' parameters. In this thesis, only a single parameter is allowed to vary, so equation (3.1) becomes

$$\frac{d\vec{x}}{dt} = \vec{F}(\vec{x},\, k,\, t) \tag{3.2}$$

The *sensitivity derivative* that measures the effects of small parameter variations on the vector field near a particular state-space point $\vec{x}_0$ for the parameter value k_0 is

$$\left.\frac{\partial \vec{F}(\vec{x}, k, t)}{\partial k}\right|_{\vec{x}_0,\, k_0,\, t_0} = \lim_{t \to 0} \vec{\Delta} \tag{3.3}$$

where $\vec{\Delta} = \phi_t(\vec{x}_0,\, k_0 + \delta k) - \phi_t(\vec{x}_0,\, k_0)$:



Parameter changes affect both local and global features of the state-space plot, causing basins to move and change shape and attractors to mutate from one type to another. Fixed points split and merge, forming new fixed points, limit cycles or chaotic attractors, the latter when a new coefficient value causes the system to become nonintegrable. These topological changes are known as bifurcations. Even between bifurcations, small parameter variations can have dramatic effects on the position, shape and size of existing attractors.

Because most nonlinear equations do not admit closed-form solutions, numerical integrators are typically used to generate system trajectories. The Runge-Kutta algorithm used in this thesis is fairly unsophisticated compared to others in the literature, which range from the simple forward Euler method to the

[2]it is said to be part of the system's *α-limit set*

[3]These terms are not standard terminology.

Bulirsch-Stoer[76] algorithm, which uses Richardson extrapolation, to symplectic methods[83], wherein each iteration is given by a canonical transformation of the state space. See, e.g., [63, chapter 15] or [79] for other examples.

Since a numerical integrator is itself a dynamic system, it can mask or alter a system's true dynamics. A simple instance of integrator-induced dynamics is a forward Euler integration of the simple harmonic oscillator:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -x \end{aligned} \tag{3.4}$$

The derivative extrapolation adds numerical negative damping and the simulated state-space trajectory spirals outwards instead of remaining on the unit circle. This effect worsens as the time step grows; it is also found to a lesser extent in virtually all other algorithms, including Runge-Kutta. The general situation is more complex — and daunting, from the point of view of an engineer — because the integrator-induced changes are virtually indistinguishable from the dynamics of the system itself. This complexity is not surprising; two coupled equations, at least one of which is nonlinear, are operating upon inexact state and approximate coefficients. As a result, one cannot blindly attribute a particular path, basin, bifurcation, etc, to the target system. It may well be a result of the timestep, the order, the family of the method, or some combination thereof. The uncertainty arises primarily because most numerical integrators do not preserve all of the integrals of the motion. An important exception is symplectic integration, which preserves, among other things, the energy integral. Bulirsch-Stoer methods treat the dependence on the step size analytically in order to mitigate its effects. One important implication of this inaccuracy is that simulations should be viewed with suspicion until other time step values and integration methods have been used to double check the conclusions. In the limit as $\Delta t \to 0$, of course, *all* numerical integrators work perfectly — and take infinitely long to run. The nonlinear dynamics of mechanical integrators and numerically-induced chaos have become popular research topics in the last few years[6, 39, 56].

Because of the complexity inherent to nonlinear and chaotic behavior and the high number of dimensions that are often involved, techniques that condense information are common in analysis and presentation. The most popular method is the Poincaré section, which reduces a system's dimension by sampling in time or space. In a nonautonomous system, the trajectory is sampled once per drive period T. In an autonomous system, the trajectory is sampled where it pierces some $n-1$-dimensional transverse hyperplane Σ in a defined direction. On a Poincaré section, a period-k limit cycle appears as a repeating sequence of k points, a quasiperiodic orbit as a filled curve, and a chaotic attractor as a dust of points sprinkled over some portion of the space. Other common information manipulation techniques, not used here, include bifurcation diagrams, correlation graphs, delay coordinate embedding, etc. See [14] for a review.

More details about the concepts in this section may be found in, e.g., [40].

3.2 Chaotic Dynamics

Chaos is complicated, unpredictable and seemingly random behavior in a deterministic physical system. A system need not be complex, noisy or subject to experimental error to exhibit chaotic behavior: it need only be nonlinear, and, if it is continuous-time, have three independent state space dimensions. This definition is at odds with the classic conception of determinism, as expressed by Laplace: "given the exact position and velocity of every particle in the universe, [I can] predict the future for the rest of time"[22]. The strict truth of this statement rests upon a tight interpretation of the word "exact." Although only deterministic forces are present, a chaotic system regime is random — in the sense that the distant future cannot be predicted without an infinite-precision knowledge of the present.

Every nonlinear dynamicist has his or her own favorite definition of chaos; differences between them are subjects of spirited debate. Two are examined at length here and a number of others are summarized.

One of the most widely-accepted definitions is that of Guckenheimer and Holmes [34], which states that a chaotic system has:

- A countable set of periodic orbits of arbitrarily long periods
- An uncountable set of bounded nonperiodic motions
- Local merging and separating constraints that "mix" trajectories

The state space of a chaotic system is filled with a dense web of stable and unstable manifolds; where these intersect transversally, they form heteroclinic and homoclinic orbits or points[4]. One such orbit — a fixed point on a section — implies the existence of an infinite number of others of the same type[5, 34]. These unstable periodic or fixed points, together with a limited number[49] of tamer stable ones, fill the first category of the definition above. The "bounded periodic motions" fill a region of the system's state space, eventually passing arbitrarily close to every point. Trajectories travel around in chaotic zones, being pulled in along the stable manifolds and pushed out along the unstable ones. These pushes and pulls form the layers and folds of the chaotic (or *strange*) attractors typical of such systems and give rise to the symptomatic sensitive dependence upon initial conditions that

[4]In the state space, a homoclinic orbit starts at a saddle point and wraps back around to itself. A heteroclinic orbit connects two saddles. These two species are comparatively rare and very interesting; most trajectories simply start from a repellor and end on an attractor. See [5, Part 3].

is expressed in the third item of the definition. The denseness of trajectories also gives rise to the property elucidated in the Shadowing lemma, informally stated thus: "With high probability, the sample paths of the problem with external noise follow *some* orbit of the deterministic system closely"[29][5].

Devaney's definition of chaos[26] requires the chaotic region of the system's state space to exhibit:

- Sensitive dependence upon initial conditions
- Topological transitivity
- Dense periodic points in the Poincaré section

Topological transitivity means that chaotic regions are connected: chaotic trajectories are confined to one region unless perturbed. On a surface of section through the attractor, the unstable periodic orbits appear as fixed points at the intersections of the stable and unstable manifolds of the system. The apparently-orderly phenomenon of embedded periodic orbits seems out of place within the chaos, let alone as part of its definition; these orbits, however, exert little influence on real trajectories because of their instability. External control has been used to stabilize systems at some of these periodic points[59], as discussed further in section 3.4.

An unscientifically-chosen sampling of other definitions from current texts, popular nonfiction and picture books (in alphabetical order by first author) appears below:

- "A dynamical system ... is chaotic if (i) it is transitive (ii) it is sensitive to initial conditions (iii) the set of periodic orbits ... is dense..."[10]
- "The irregular and unpredictable time evolution of many nonlinear systems has been dubbed 'chaos.'"[9]
- "The motion on strange attractors is chaotic" "...on which nearby orbits, though bounded, diverge exponentially."[51]
- "...a chaotic system is a deterministic system that exhibits random behavior"[60]
- "...the sequence is *determined* by its initial value — and yet, it *cannot be predicted* other than by letting it run."[62]
- "...recurrent behavior that is not an equilibrium, a cycle or even quasiperiodic motion...arises from sensitive dependence on imperfectly known initial conditions, resulting for example in broadband noise..."[78]

[5] A more formal statement may be found in [34, page 251].

Chaos theory is powerful, fascinating, and broadly applicable across science and engineering, but it has a variety of drawbacks and inconsistencies. Nonlinearity is a necessary condition, but sufficient conditions, at least for real physical systems, are not as clear: "the justification for classifying much irregular behavior as chaos depends on the accumulation of numerical evidence and on experience with a few idealized mathematical systems known to [have positive Lyapunov exponents]"[22]. Proving nonintegrability is difficult, even if modeling is not an issue. Conflicts between chaos and thermodynamics also arise. Cream stirred into a cup of coffee is a good example of this — the equations governing interaction of the spoon and the small volume elements of fluid are deterministic and yet the mixing sends the system down the (unclimbable, by the second law of thermodynamics) entropy hill. It is also difficult to reconcile chaos and quantum mechanics; this recently-debated topic[31] is part of the bigger problem that the latter has with the world as it is described by classical mechanics. Schrödinger's equation is linear. Because it describes the fundamental constituents of matter, one should theoretically be able to use quantum mechanics to describe anything at all, but the linearity theoretically precludes any chaotic behavior. For these reasons, and others, many theorists are uncomfortable with the field and its level of mathematical rigor.

3.2.1 Properties of Chaotic Attractors

Many chaotic systems exhibit fractal structure in their state-space portraits; some authors even use that as part of the definition. A fractal[54] is a self-similar structure with non-integer Hausdorff dimension.

Figure 3.1 shows iterations of the system of nonlinear difference equations:

$$\begin{aligned} x_{n+1} &= \alpha - x_n^2 + \beta y_n \\ y_{n+1} &= x_n \end{aligned}$$

with $\alpha = 1.24$ and $\beta = 0.4$. The behavior is chaotic and the attractor — the Hénon attractor — is fractal.

The Hausdorff or fractal dimension of a set A is strictly less than the topological dimension of the manifold on which it lives; it is defined as

$$D_h = \lim_{\epsilon \to 0} \left\{ \frac{\log(N(A,\epsilon))}{\log(1/\epsilon)} \right\} \tag{3.5}$$

where $N(A, \epsilon)$ is the smallest number of closed balls of radius $\epsilon > 0$ needed to cover A. For a Cantor set — a line segment with the middle third removed, *ad infinitum* — $D_h = 0.63$. For a Cantor set with the middle *fifth* removed, $d_h = 0.76$. A chaotic attractor is the cartesian product of a Cantor-like set and a smooth manifold.

The denseness that gives rise to the Shadowing lemma makes a chaotic attractor robust with respect to noise. Variations in state[6] simply disturb a trajectory

[6]Limited, of course, by the distance to the closest basin boundary.

Figure 3.1: A series of expansions of the fractal Hénon attractor, exhibiting self-similar structure. Each plot is a magnification of the small rectangle on the plot to its left.

such that its new path eventually relaxes to an attractor thread that it would ultimately have reached in any event. However, the time taken to reach that point is unpredictable. This robustness does *not* extend to parameter variations, nor is it a universal panacea for control errors. These issues arise at several points in the body of this document.

Lyapunov exponents, which parametrize the growth of perturbations in chaotic systems, are defined as:

$$\lambda = \lim_{t \to \infty} \frac{1}{t} \ln |m_i(t)| \tag{3.6}$$

where the $m_i(t)$ are the eigenvalues of the variational system — the matrix-valued linear differential equation that governs the growth of a small variation in the initial condition, and whose derivation is covered in section 5.3.2. An n-dimensional system has n λs, each measuring the expansion rate, along one basis vector, of the distance between two neighboring trajectories. The largest positive λ dominates the behavior as $t \to \infty$; if all are less than zero, the one with the largest magnitude dominates the state-space contraction. In the linear case, the Lyapunov exponents are equivalent to the real parts of the eigenvalues.

Positive λs play a role in chaos and are sometimes used in its definition. In a three-dimensional system with three positive λs, no attractor exists because all directions are expanding and all trajectories diverge. If all three are negative, trajectories relax to a fixed point. If there is one positive and one negative λ, the system is chaotic[7]. Locally and in cross-section, a three-dimensional chaotic

[7] $\lambda_1 > \lambda_2 = 0 > \lambda_3$.

attractor has one positive and one negative λ (= eigenvalue); if both were positive, the point would not be on an attractor.

Large positive λs present some numerical and computational difficulties — because of the computer's floating point limit, the possible stiffness of the system and the near-zero pivots in the matrices. One approach is to integrate the variational system forwards for T seconds, find its eigenvalues, and let $T \to \infty$. However, if any λ is positive, this method fails after $T_{blowup} = \frac{k \ln 2}{\lambda}$ seconds in a computer with k-bit arithmetic. The encounter between the largest λ and the floating-point limit is not the only problem; the disparity between λs can cause basis vectors along shrinking or slowly-growing directions to get lost in the signal-to-noise ratio of the pivoting. A better approach is to integrate forwards for $T < T_{blowup}$, renormalize the evolved variational vector to $\vec{u}^{(1)} = \frac{\delta \vec{x}^{(1)}}{||\delta \vec{x}^{(1)}||}$, and repeat the operation K times, keeping track of the renormalization factors. Then the largest positive Lyapunov exponent is

$$\lambda_1 = \frac{1}{KT} \sum_{k=1}^{K} \ln ||\delta \vec{x}^{(k)}||$$

3.3 Classical Control Theory

A control system imposes a fixed relationship between a system's[8] outputs and its inputs. Closed-loop control systems "feed back" a measurement of the state or the output to accomplish this goal. State-variable feedback hooks into the equations more directly, but is not possible in systems that are not *observable*. Output feedback is easier to implement because outputs are, by definition, accessible. However, the output may not be an adequate measure of the system's state. A yellow traffic light in some European countries, for example, may be about to turn red or green. Even if the state variables are accessible, a single zone of *controllability* may not encompass both the starting point and the goal, in which case *no* time-invariant controller can move the operating point from one to the other[9]. Controllability and observability have grave implications for limits on both linear and nonlinear techniques; the mathematics involved is much more difficult in the latter case. Feedback control has a variety of useful attributes; in electronic circuits, for example, a closed-loop amplifier can be less sensitive to variations in component values and to particular kinds of noise, or have better input and output impedance, etc.

Analysis and synthesis techniques in feedback control are comparatively easy — and extremely well-studied; see [13], for example — for linear systems, but much more *ad hoc* and complicated in the nonlinear branch of the field. This relative lack of coherency can be traced back to a variety of causes; several of the main

[8] In the control literature, the target system is termed the "plant."

[9] i.e., the goal is not *reachable* from the origin.

culprits are summarized below:

- Very few nonlinear systems admit closed-form solutions
- Many of the classical tools like Laplace transforms and the Nyquist criterion depend critically upon superposition, and hence linearity
- Nonlinear systems can have many complicated, interwoven basins of attraction; the behavior in each can be very different and both the topology of the boundaries and the texture of the behavior can be highly sensitive to parameter and state changes

The final item on this list is perhaps the most crucial. It has far-reaching implications for both analysis and synthesis: local and global decomposition, stability, controllability, etc.

Despite these difficulties, nonlinear control theory has been developing steadily over the past three decades, driven by the fact that the world — and hence most of the systems that one would want to control — is basically nonlinear. Computers are playing a growing role in this development, not only as glorified calculators that allow their users to quickly simulate system trajectories, but also as competent assistants that automate significant chunks of the design process[45] and even possibly as tools that bring some of those chunks into the on-line phase of the design. This expansion of computers' roles in the design process is occurring in other realms of science and engineering as well[3]; some broader implications of this are discussed in section 3.5.

Reference texts for the remainder of this section are: Friedland[32] for general control theory with a particular emphasis on state-space methods, Slotine and Li[73] for general nonlinear control, Åström[7] for adaptive control, and Roberge[65] for electronic applications. Omission of a citation for an individual concept or method means that it is cited in most or all of these texts. This list is only a very small (and personally-biased) sampling of the hundreds of texts available on these topics.

Possibly the simplest and most obvious way to sidestep the mathematical problems posed by nonlinearity in a system is to linearize the equations around an operating point and construct a linear controller, based on that approximation, that is valid in a small state-space patch surrounding that point. A worked-out example of this is given in section 3.3.1. The size and shape of the region where the linearized controller works are dictated by the abruptness of the nonlinearity: a linearization of a slowly-varying system works over a broader range than a linearization of a quickly-varying one. Furthermore, the inherent approximation can hide critical or interesting features like separatrices, fractal basin boundaries, or observations that are essential to proofs about global structural stability. Singularities near an operating point present serious problems for linearization and are

quite common in engineering systems (e.g., an operational amplifier in saturation or backlash in a gear.) Feedback control mitigates these effects by muffling the sharp edges; the high loop gain softens the nonlinearity, narrows its range and thus partially insulates it from the system's terminals (and vice versa.) For example, the width of the crossover distortion region in a complementary emitter-follower output stage can be reduced by the gain of the loop.

Despite its drawbacks, linear control is useful and ubiquitous. Its theory and practice have been case-hardened by years of research and design experience and so are well-thought-out and powerful. In this thesis, for instance, local linear control is used — effectively — once the global-view, nonlinear controller has brought the trajectory into range. It is also used to recover from small errors in well-characterized locations along the large-scale path.

A classic electrical engineering application of linear control is the standard method for operational amplifier design. This recipe, targeting a particular rise time/overshoot or gain/bandwidth combination, specifies construction of Bode, Nyquist and root-locus plots and suggests ways to adjust the values (R, C, etc) of the components that affect the loop compensation such that the specifications are satisfied. An example of a design rule that is used in this process is that a pole close to the imaginary axis causes a lower rise time and a higher overshoot. If these cookbook adjustments are unsuccessful, the circuit's topology can changed via the addition of one of a variety of well-known hacks — like a lead or lag compensation network — and the adjustment can be repeated[65]. One can attempt to cancel out an offending pole by superimposing a zero upon it, at the same time adding another pole in the desired location. This type of cancellation is dangerous in some systems; in the inverted pendulum, for example, it can make the device uncontrollable[32].

Linearization makes the mathematics of global/local composition/decomposition very difficult. *Local* properties of fixed points and *first-order* bounds on a variety of properties can be established with the linear theory, but generalization is very difficult. One classic scheme solves this by piecing together a collection of local-linear controllers into a global control system: essentially a patchwork of locally-applicable feifdoms. This approach has a long and rich tradition; it was pioneered by Kalman[48] in the 1950s and, recently, has even seen some AI applications[67].

Simple adaptive control strategies[7] also piece together appropriate linear algorithms in different regimes of a piecewise-linear problem, but the regimes are not necessarily defined as *state-space* regions. Adaptive control systems have two characteristic time scales — one for the feedback loop and a slower one for the *changes in* the feedback loop. In gain scheduling, the loop gain is adjusted according to a predetermined table or schedule. Model reference control is yet another flavor, wherein the differences between the real system's output and the output of an ideal model are used to modify the real system's inputs so as to drive the difference to

zero. Many other adaptive control algorithms exist as well. The dimension that adaptive control adds to the space — the feedback gain — can make previously unconnected regions reachable.

Many variations and improvements on the linearization theme has evolved over the past 50 years. Some of these are mathematically more intelligent about how the linearization itself is accomplished. For example, better approximations than a straight Taylor series/Jacobian derivative approach exist; Killing systems (of which bilinear functions are an instance) are one example[20]. Other methods extend the idea of "linearization" into the frequency domain. Describing functions[65, 73], a nonlinear analog to the sinusoidal steady state response, can be used to compute steady-state behavior (fixed points, limit cycles, etc) — but not transients. Describing functions, like all other approximations, have a limited domain: they only work when the nonlinearity (1) is odd (2) can be lumped into one component (3) contributes only small harmonics and subharmonics and (4) is time-invariant.

Members of another broad class of nonlinear analysis and synthesis methods reverse-engineer the precise form of the target system's nonlinearity and use that information to transform the closed-loop system so that linear techniques can be applied. Many of these methods factor the system's physics into the controller. The resulting designs are conceptually clean, in direct contrast to the *ad hoc* approaches described in the previous paragraphs, but they are very hard to implement. For example, stability can be established by examining the appropriate Lyapunov function[40, 73] for the system. Little mathematical guidance in finding such a function is available, outside of the requirement that it be positive definite, but they often "look like" energies. In feedback linearization[46], the original state variables are transformed ("nonlinear feedback") so as to make the closed-loop system linear. This technique is obviously powerful, but finding a good transformation, like finding the right Lyapunov function, can be very difficult. A change of coordinates can have the same effect, as can the determination of an immersion of a nonlinear system into a linear one[10][21]. Injecting physics into control design is a potentially rewarding but difficult task[72]; it creates highly-specialized, hand-crafted designs that demand knowledge, design time, and time-invariant applications, but that work very well when all of those conditions are met. Highly-specialized, hand-crafted control laws can break down in the face of modeling inaccuracy; *robust* control techniques, such as sliding mode control[73], combine the "nominal" control with secondary techniques that are specialized to deal with these uncertainties.

This section gives only a brief and incomplete summary of the broad literature on nonlinear control and the even-vaster linear control literature; see [25] or the texts listed on page 36 for a more comprehensive review. The common thread uniting almost all nonlinear control techniques is to use ingenious ways to "get around" the nonlinearity so that the highly developed linear theory can be brought

[10]Feedback linearization is actually a special case of an immersion.

to bear. The detour can be accomplished either by the controller or by the observer that measures the system state.

3.3.1 Traditional Linear Control: An Example

This section, a worked-out example of traditional linearized control, can be omitted without loss of continuity, but is useful background for section 7.2. Theory and references for this development are given in section 3.3.

The two-dimensional nonlinear system:

$$\vec{F}(\vec{x}) = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ \Omega^2\theta - \alpha\omega + \beta u \end{bmatrix} \tag{3.7}$$

is a normalized model for a simple, damped (α) pendulum, driven (βu) by a motor attached to its pivot. With $\beta u = 0$, this system has metastable equilibrium points at $\theta = 0 \pm 2n\pi$ and stable equilibrium points at $\theta = \pi \pm 2n\pi$.

The Jacobian $\frac{\partial f^i}{\partial x_j}$ (or $D_x\vec{F}$) of this vector-valued ordinary differential equation is:

$$\begin{bmatrix} 0 & 1 \\ \Omega^2 & -\alpha \end{bmatrix} \tag{3.8}$$

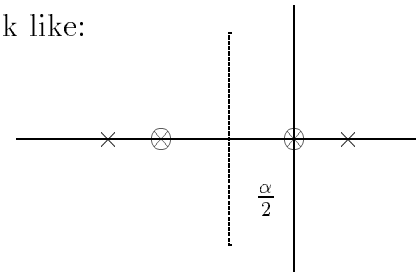
This matrix is computed by the `jacobian` function discussed in section 5.3.2. The entries in (3.8) happen in this case to be independent of state and time; this does *not* hold in general. The characteristic polynomial is

$$|D_x\vec{F} - s\vec{I}| \;=\; s^2 + \alpha s - \Omega^2$$

and the eigenvalues are

$$-\frac{\alpha}{2} \pm \sqrt{(\frac{\alpha}{2})^2 + \Omega^2}$$

On a pole-zero plot with $\alpha > 0$, these look like:



The $\otimes$ points are $\Omega = 0$ poles and the $\times$ poles reflect some nonzero natural frequency. The latter — the true physical situation — is clearly unstable. The $\alpha < 0$ plot is the mirror image of this one about the y-axis.

A traditional linear controller stabilizes a system

$$\dot{x} = Ax + Bu \tag{3.9}$$

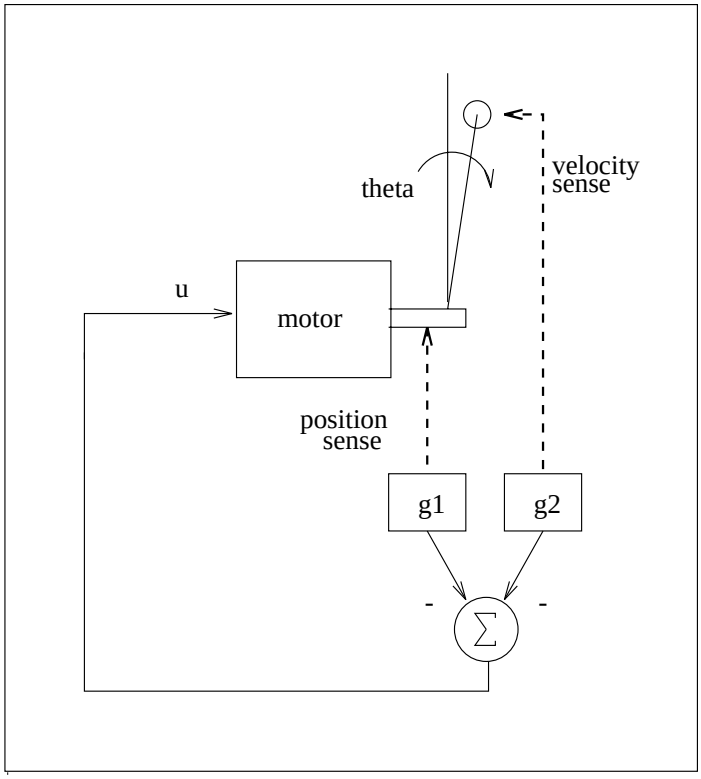


Figure 3.2: Linear feedback control of an inverted pendulum: A traditional linear controller stabilizes a system by wrapping it in a feedback loop and finding values for the feedback gains g_1 and g_2 that place the poles of the closed-loop system in the left-half plane.

by wrapping it in a feedback loop (see figure 3.2) and finding the gain matrix G of the feedback element — g_1 and g_2 in the figure — such that the poles of the closed-loop system are in the desired location, presumably somewhere in the left-half plane. A system is controllable if and only if the rank of the matrix

$$Q = [B\, AB\, \ldots\, A^{m-i}B]$$

is equal to m, the order of the system.

If the control objective is to balance the simple pendulum at the inverted ($\theta = 0$) point and the available torque is unrestricted, A is the Jacobian (3.8), the poles are as shown on the previous page, and (compare (3.9) and (3.7))

$$B = \begin{bmatrix} 0 \\ \beta \end{bmatrix}$$

The matrix

$$Q = \begin{bmatrix} 0 & \beta \\ \beta & -\alpha\beta \end{bmatrix}$$

is of full rank, so the system is controllable.

In general, the gain matrix G required to site the closed-loop system's poles so as to make the characteristic polynomial be

$$s^2 + a_1 s + a_2 \tag{3.10}$$

is

$$G = [(QW)]^{-1}(\hat{a} - a) \tag{3.11}$$

where

$$W = \begin{bmatrix} 1 & a_1 & \dots & a_{k-1} \\ 0 & 1 & \dots & a_{k-2} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

and $(\hat{a} - a)$ is the difference vector between the desired and actual coefficients of the characteristic polynomial.

For the pendulum,

$$W = \begin{bmatrix} 1 & a_1 \\ 0 & 1 \end{bmatrix} \qquad (\hat{a} - a) = \begin{bmatrix} a_1 - \alpha \\ a_2 - (-\Omega^2) \end{bmatrix}$$

and

$$G = \begin{bmatrix} 0 & 1/\beta \\ 1/\beta & 0 \end{bmatrix} \begin{bmatrix} a_1 - \alpha \\ a_2 - (-\Omega^2) \end{bmatrix} = \begin{bmatrix} (a_2 + \Omega^2)/\beta \\ (a_1 - \alpha)/\beta \end{bmatrix}$$

The roots of the closed-loop polynomial (3.10) are:

$$-\frac{a_1}{2} \pm \sqrt{(\frac{a_1}{2})^2 - a_2}$$

The effects of a_2 on the pole positions — assuming a fixed, positive a_1[11] — is shown graphically in part (a) of figure 3.3. The magnitude of a_1 determines the damping constant of the critically-damped case where the two poles meet. The system is marginally stable for $a_2 = 0$ and unstable for $a_2 < 0$. The precise pole placement depends on the specification and the design process resembles the mentioned in conjunction with operational amplifier design recipe on page 37. If the specifications can tolerate some oscillation — part (c) of the figure — one would choose $a_2 > (\frac{a}{2})^2$ to obtain a faster closed-loop response. If overshoot is undesirable, one chooses $(\frac{a}{2})^2 \geq a_2 > 0$ to place the poles on the negative real axis between the two $\otimes$ points on part (a), and lives with the slower transient response shown in state-space form in part (b). The two values a_1 and a_2 can be used to obtain a precise value for the eigenvalues and eigenvectors that guide the trajectory

[11] If a_1 were chosen negative, *no* a_2 value could make the system stable.

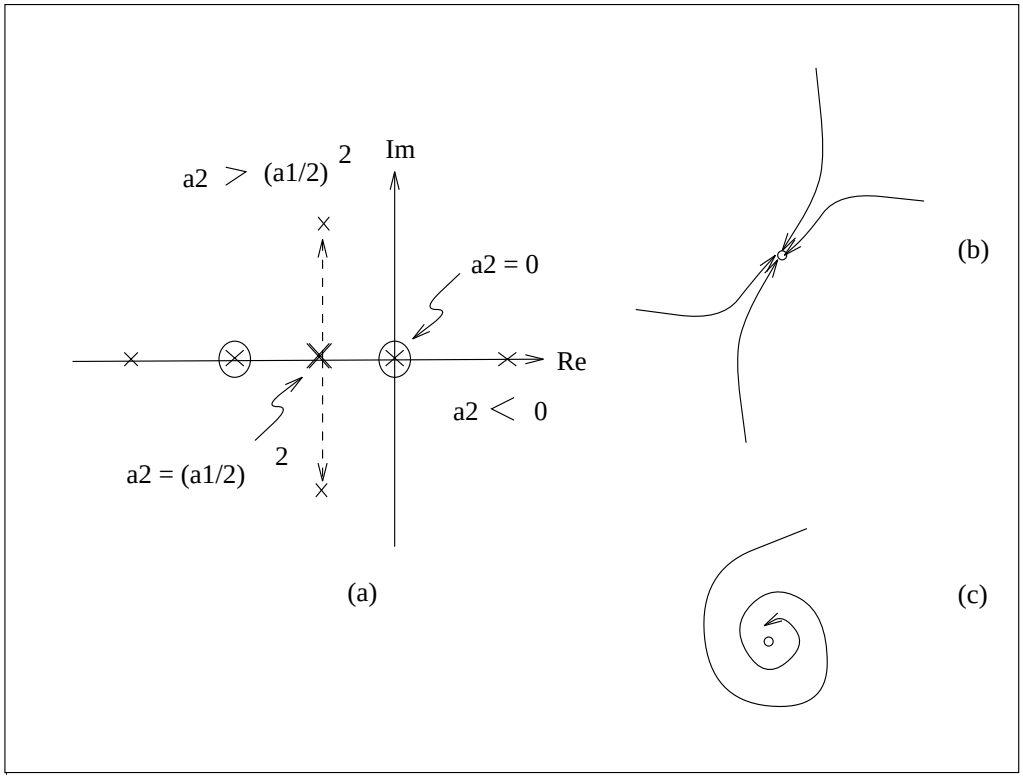


Figure 3.3: Pole selection for linear feedback control: (a) root-locus plot as a function of a_2 (b) overdamped behavior (c) underdamped behavior.

in to the newly-stabilized fixed point. They can also be used to balance the rise time/overshoot tradeoff by moving the complex poles nearer the imaginary axis.

For example, say the pole positions $(-1 \pm 4j)$ were chosen, yielding a closed-loop polynomial of $s^2 + 2s + 17$ and a local transient (step) response $Ae^{-t}\cos(4t + \psi)$. Recall that the gains are tailored to a particular value of the Jacobian matrix A and drive B; if those matrices change, the closed-loop poles move. If they move into the right-half plane, the system becomes unstable. In this particular case, A, B and $(\hat{a} - a)$ — and hence G — do not depend upon the state, so deviations do not affect the poles or the stability. This lack of dependence on state implies a range that covers the $\theta - \omega$ plane, but requires high amplitudes and speeds from the drive, so practical limitations restrict the range below that ideal. In general, A and possibly B are highly position-sensitive, so linearized control is even more tightly restricted around the operating point. The shape of this region can be determined by making variations along each of the basis vectors and identifying the values that cause the poles to cross the axis (or exceed some other design limit like bandwidth.) These limits are, of course, functions of the initial pole placement as well; poles that are deep in the left-half plane are in some sense "safer" because they are further from the $j\omega$ axis.

3.4 Controlling Chaos

Chaos in engineering systems was considered complete anathema until very recently. Observation and analysis of the phenomenon over the last few decades has evolved into synthesis and active use in the last few years.

Chaotic behavior in adaptive control systems has been documented by several groups. Both the system under control and the adaptive algorithm may contribute to the prerequisite nonlinearity; external disturbances can also play a role. In [55], the target system has a singularity at the desired operating point, so linearization is impossible. The parameter being varied is the accuracy of the controller's model of the plant. The feedback gain shows classical chaos: for some parameter values (good plant models) it is periodic when the plant is stabilized. For others (bad models), the gain behaves chaotically but *nonetheless* regulates the output. This observation causes the authors to hint in the direction of controlling chaos, but they fall short of active pursuit of the idea, admitting only that "chaotic parameter estimates are not necessarily a bad thing to have"[55]. Sudden perturbations in plant parameters can set off chaotic bursts on adaptive control system outputs; for a large class of systems, it has been conjectured[42] that there is a quantifiable[12] tradeoff between the maximum perturbation amplitude from which a system can recover monotonically and the time it takes to do so. Note that this statement is

[12] inversely proportional to a linear power of the control stiffness

stronger than the usual linearized/local/Lyapunov stability analyses, in which the perturbations are infinitesimal. These kinds of bursts — unprovoked or in response to perturbations — turn up in many natural and man-made systems. Virtually any of the figures in [53], for example, as well as many of the papers cited here in other contexts (e.g., [12, 42, 55, 59],) show or remark upon these bursts.

The first, most rudimentary use of chaos in the design process was a simple matter of recognition and avoidance: using knowledge of chaotic zones' boundaries to site an operating point as close to the middle of a non-chaotic basin as possible[12], or knowledge of the parameter ranges where it occurs to avoid it altogether[71].

Many of the current and recent papers that have some permutation of the words "control" and "chaos" in their title — like [71] — simply suppress chaos. However, its unique properties make possible a new spectrum of design techniques which derive their power from *not* treating a chaotic system as a linearized and tamed cousin to its true nature. Members of the first generation of these schemes have exploited the denseness of strange attractors and of unstable periodic orbits (UPOs.) The most well-known of these [59, 69], discussed and illustrated with a worked-out example in section 6.2, stabilizes the UPOs embedded in a chaotic attractor with a local-linear scheme and relies on the attractor's denseness to cause the trajectory to enter the controller's domain. Many other tacks are also possible, like statistical forecasting[58, 47]; several new ones are developed in this thesis.

Even if a periodic orbit does not always meet the specified control objectives, chaos can still be useful. In general, to make a point reachable, one need only identify parameter values that give rise to fixed points near the destination or a chaotic attractor that covers it; the high parameter sensitivity can reduce the size of the control action necessary to create these features. State-space regions where the system is sensitively dependent on initial conditions, once identified and characterized, can be used to magnify the effects of small control actions. The statistical properties of chaotic attractors determine the robustness of control schemes that use them.

3.5 Computational Methods

Much recent work has been devoted to developing programs that automate chunks of the scientific process. Dialogs with such programs are remarkably high-level, resembling the interaction between a professor and a graduate student. This has caused a proliferation of names of the form "The [professional title]'s assistant/apprentice[13]."

Many of these tools achieve their success by mixing metaphors[3]. For instance,

[13] some of which exhibit behavior more akin to that of the *Sorceror's* apprentice.

a program can use an ODE to generate numerical results, or it can reason with the equation directly in symbolic form to establish more general results (e.g., which sets of reactants can reach equilibria in a chemical reaction.) An algebraic representation of a transfer function and a geometric description of the changing positions of its poles — a root-locus plot — are each valuable (and non-interchangeable) at different points in a design. The combination of these algebraic/geometric and numeric/symbolic computing techniques, together with traditional AI techniques like search, rules and frames[14], multilevel descriptions, encoded domain knowledge, etc[82] and with more recently-developed areas like qualitative physics[80] — and even more traditional areas like dynamic programming[11] — has created a powerful paradigm for scientific computation.

Using this arsenal of techniques, intelligent design assistant programs can prepare experiments, monitor and interpret their results, and present them to the user formulated in easy-to-understand, qualitative terms[1, 30, 84, 86]. Others can streamline and accelerate computation by automatically designing, optimizing and using special-purpose numerical engines for specific problems[2, 77].

[14]Expert systems, incidentally, have been used in automatic control for at least two decades[33] — part of the general area of intelligent control[44, 68], a unification of control theory and AI.

Chapter 4

State-Space Mapping

Perfect Moment's mapper autonomously explores the state-space behavior of a nonlinear system. Its inputs are an ODE, a pair of origin and destination points, a specification of optimality cost, definitions of the corners of a state-space region and an iteration depth. Over the course of its run, it manipulates the control parameter and identifies relevant state-space features. The end result is a set of state-space portraits, spaced so as to sample the system's different behavior patterns, each comprising just enough trajectories to characterize the dynamics at each step.

A flowchart of the mapper's actions appears in figure 4.1. Following some background material (section 4.1,) trajectory generation and discretization are covered in section 4.2. Section 4.3 describes the dynamics classification scheme and section 4.4 discusses the use of its results — the other two boxes and three diamonds in figure 4.1 — to select representative control parameter values.

4.1 Cell-to-Cell Mapping

The cell-to-cell mapping formalism of Hsu[41], used extensively here in a slightly-modified form, is briefly reviewed in this section.

The region of interest is partitioned on a grid. Each of the n state variables x_i, $i = 1 \ldots n$ is divided into m equal intervals h_i, so the space contains $M = \prod_i m_i$ cells. At each pass of the search, m_i and h_i may be different. Each parallelepiped in this grid is identified by an n-vector of coordinates $z_i \in Z^n$, called a cell vector and denoted $\vec{z}$, whose ith coordinate gives the grid height in the ith dimension. *Perfect Moment* represents the cell vectors `z` with Scheme vectors `#(x1 x2 x3 ... xn)`. A state-space trajectory ϕ_t is a series of n-vectors of real numbers. The discretized version Φ_t is the list of the cells touched by ϕ_t — identified by

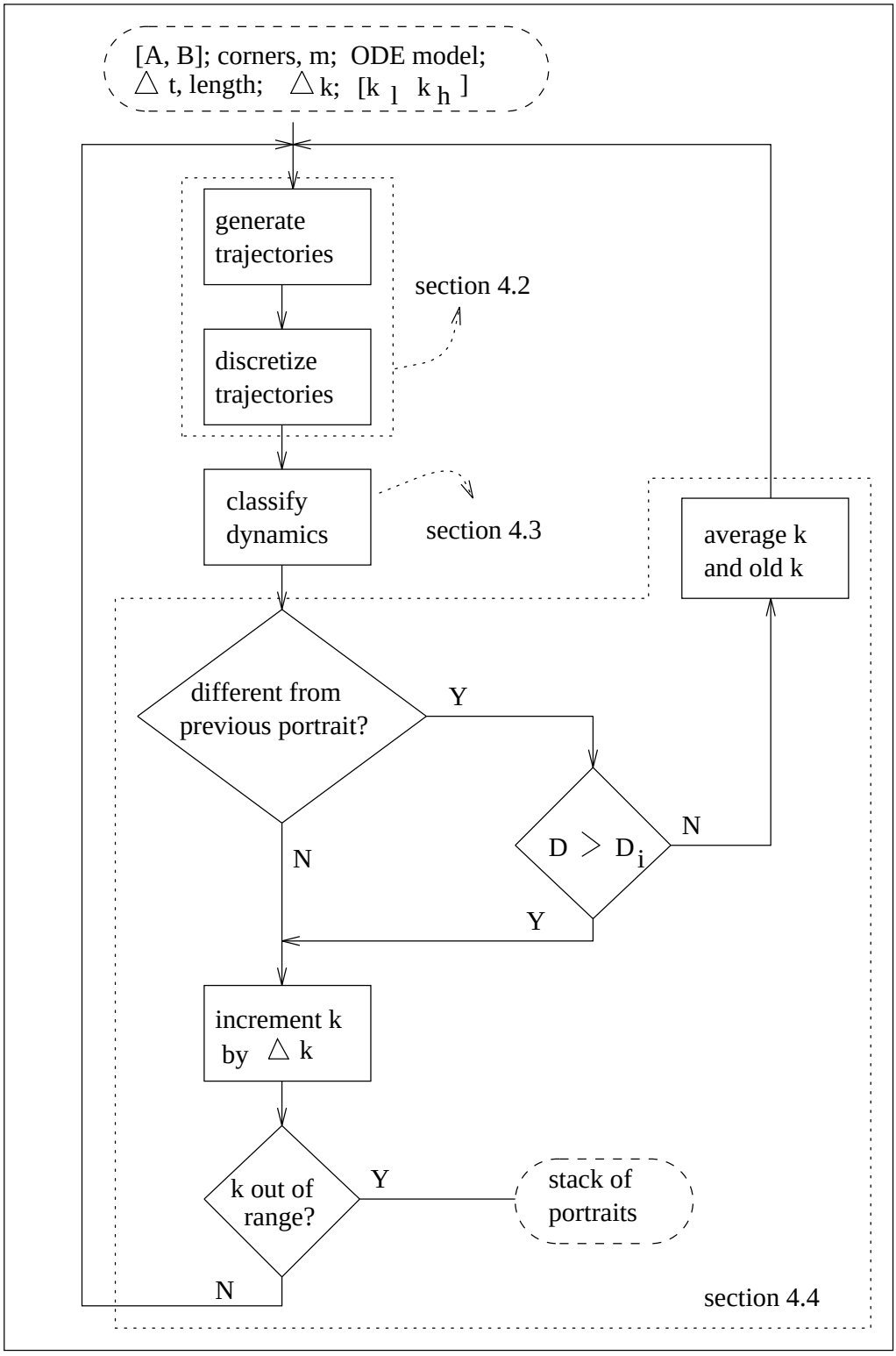


Figure 4.1: The mapper's flowchart: dashed ovals are inputs and dotted boundaries group the topics according to the section where they are discussed

n-vectors of integers between 0 and $m-1$, inclusive —and the time of entry to each cell. A description of the flow can be given as a set of M mappings, each of which describes the travels of a trajectory that starts at the center of a cell. A trajectory exiting the grid altogether is said to have entered the sink cell.

4.2 Trajectory Generation and Discretization

4.2.1 Timestep Selection and Trajectory Length

A fourth-order Runge-Kutta numerical integration algorithm, which locally approximates the system's state-space movement with the first four terms of its Taylor series, is used to generate trajectories $\phi_t(\vec{x}_0)$ from initial conditions x_0 (or $\phi_t(x_0, t_0)$ from (x_0, t_0), for nonautonomous systems.) The Scheme function `runge-kutta4` operates upon a system derivative like the following, which models a simple pendulum:

```
(lambda (*state* t)
  (let ((theta (first *state*))
        (omega (second *state*)))
  (list omega
        (+ (* (square natfreq) theta)
           (* -1 alpha omega)))))
```

An *adaptive* `runge-kutta4`, which dynamically fits the timestep to the system's natural frequencies, is applied first — to a trajectory emanating from the origin for each $k_{low} + n\Delta k$ in the range — with a local truncation error specification that can be met by using roughly 100 timesteps per orbit. The *smallest* timestep Δt_b that the integrator is forced to use is chosen as the baseline for the entire procedure. This technique is an odd but not unnatural use of an integration method: to "measure" a system's highest natural frequency. The discussion below clarifies the need for the seemingly-excessive caution — the universal use of the smallest timestep — on *Perfect Moment*'s part.

Several factors affect the timestep choice: the system's dynamics and the "dynamics" introduced by the mapping and search algorithms. All of these are influenced by the optimization criteria. Issues concerning the former are discussed at length in section 3.1. The next four paragraphs address the latter.

The grid places an upper bound on the timestep. If Δt is too large, adjacent points in a trajectory will not fall in adjacent cells, which causes difficulties both in computation of optimality cost (i.e., unconnected paths) and in dynamics classification. *Perfect Moment* reduces the timestep automatically if the discretized trajectory is not connected, as shown in the following transcript:

```
(generating trajectory set with timestep = .04 and *parameter* = 0)
(reducing timestep...)
(generating trajectory set with timestep = .02 and *parameter* = 0)
(generating trajectory set with timestep = .02 and *parameter* = 5)
```

Note also that, once the timestep has been reduced, all further portraits at that grid spacing will start from the reduced value: the first attempt to construct the parameter = 5 map is made with $\Delta t = 0.02$, not $\Delta t = 0.04$. This caution is excessive if the system is slower at the parameter values that occur later in the process. Incidentally, no timestep reduction took place in the scenario of section 2.4 because the grid was very coarse.

The timestep is also reduced automatically — by the path finder — when the cell size changes between successive passes of the search. If this reduction were not performed explicitly, the mapper's reduction algorithm, described in the previous paragraph, would accomplish the same thing, driven by the smaller cell size, but at the cost of $\log_2 \frac{h_{i+1}}{h_i}$ extra integrations on the first portrait constructed at the h_i grid spacing. This automatic reduction keeps the exploration scales constant and roughly equalizes the amount of time spent in each pass of the mapping process, modulo bifurcations in system- and search-induced dynamics.

After the baseline timestep is determined, only fixed-step integration is used. An adaptive integrator's automatic, dynamics-driven timestep modifications will interfere with the grid-driven rules described above (and others discussed in later chapters,) causing unpredictable behavior.

Trajectory *length* is equally critical. The dynamics classification scheme of section 4.3 relies heavily on the identification of different types of attractors, so trajectories must be long enough (1) to reach their $\omega-$limit sets ($\leftrightarrow$ attractors) and (2) for *Perfect Moment* to recognize that they have done so. From a control standpoint, longer trajectories have the added advantage of improving reachability. The drawback is, again, CPU time, which grows linearly with the number of integration steps.

As in the choice of the timestep, a conservative baseline trajectory length is established at the beginning of the run and then modified later on. *Perfect Moment* integrates all trajectories for 1000 timesteps or until they exit the grid. This attempt to capture ten orbits also ties into the classification algorithm. Though it works reasonably well, this choice is somewhat arbitrary, particularly if the system is stiff and the 100-steps-per-orbit error only plays a limiting role in a small part of each trajectory. A better approach would be to lengthen the trajectories — and perhaps the region of interest, as discussed further in the next chapter — until the destinations of some large fraction[1] of the trajectories were recognizable to the dynamics classifier. It would make sense to limit this trajectory expansion

[1]the arbitrary choice then becomes the definition of "large"

Figure 4.2: Using shorter trajectories for local tasks: if the control objectives are only separated by a short state-space distance, generating long trajectories, like those shown on page 24, is wasteful

with a user-specified parameter, similar to the iteration depth D_i, to keep the program from performing endless expansions and integrations in a vain attempt to find a nonexistent attractor. Alternatively, one could perform some sort of formal mathematical analysis with, e.g., Newton-Raphson, to establish what kinds of attractors exist. However, this technique could not be applied to experimental data — one of the ultimate goals of this project and the original driving force behind the multiple-scales method.

When the distance between the origin and destination is small compared to the grid spacing, as in the second stage of the Rössler example in section 2.4, only the first few points of the trajectories are actually evaluated by the path finder and the work devoted to the generation and analysis of the rest of their length is in some sense wasted; when the task at hand is to move a small amount up and to the right, the ultimate destination of the trajectory used to do so is almost always immaterial. When this occurs, it might be sensible to bypass *Perfect Moment*'s dynamics classification machinery and run a first pass with shorter integrations (trajectory lengths on the order of the distance between points to be connected and no attempt made to identify attractor types or refine the parameter step), then proceed with the full-length, exhaustively-classified search only if the shorter method failed. Such a scheme would make the figure on page 24 of the scenario look more like figure 4.2.

For obvious reasons, whenever the timestep reduction rules fire, the program automatically increases the number of steps so as to keep the trajectory length constant.

4.2.2 Selection of Starting Points

The distribution of the initial conditions $\{\vec{x}_0\}$ determines how well a portrait samples the true dynamics. The intent of the algorithms described in this section is to use the cell size to find a good balance between selecting enough trajectories to be representative and causing the execution time to escalate too far. The choices here are more critical than the linear-time tradeoffs in the previous section, as the run time is roughly $O(m^n)$ in the number of dimensions n and the number of cells per axis m.

The mapper divides the specified n-space region into rectangular parallelepipeds; notation and definitions for this process are given in section 4.1. A trajectory is generated from each cell center, except in the cells containing the control origin **o**

and destination **d**. In the former, **o** itself is used as the initial condition to generate the origin trajectory $\phi_t(\mathbf{o})$. In the latter, **d** is used as the initial condition and integrated backwards in time, producing the destination trajectory $\phi_{-t}(\mathbf{d})$.

The initial choice h_0 for the cell size is made by the user and heuristically revised downwards by the path finder over the course of the run, governed by intermediate results: the dynamics observed on the portraits and the locations and properties of any partial paths. Though this choice is indirectly affected by the mapper's actions, the cell size is *not* revised during a particular mapping pass. Most of the intelligence involved is exercised by the path finder and discussed in section 5.3.

The backwards-time integration of $\phi_{-t}(\mathbf{d})$ applies only when the system under control is a *autonomous, non-singular* ODE or a *reversible* physical system. For example, it would make little sense to explore a wealth of different starting conditions for trajectories of a driven-vertex pendulum when the drive arm starts at rest at $\theta = -.6\pi$. This restriction is a symptom of a deeper issue, one related to the disclaimers in the Rössler and Lorenz examples in this thesis: control parameters aren't simply sterile mathematical coefficients and must be considered in light of the system's physics. Dissipative parameters, like resistance, are easy to change. However, when energy storage is involved (e.g., charge on a nonlinear capacitor, aileron position on an airplane, etc.) or quick changes are difficult (e.g., the Rayleigh number of a fluid,) one cannot blithely vary parameters to meet control objectives. The ability to access parameters and change them quickly enough are two of the most serious issues stemming from this work; questions arise on these topics at almost every presentation and both are pursued further in section 7.4.

The problem introduced in the previous paragraph severely limits the number of trajectories that *Perfect Moment* has the freedom to explore and use, and this necessitates an adaptation in the prescription for generating starting conditions. If backwards-time evolution from the control destination makes no sense, only trajectories that emanate from the point defined by the residual system *and controller* state are permitted. The only possible variation is the control parameter. The program follows an entirely different mode of operation in this case: rather than generating trajectories sprinkled across a region, it evaluates a fan of trajectories from a single point, much like standard shooting algorithms[76] and other traditional planning/control methods. Two of the examples in this thesis, the driven pendulum of section 7.2 and the phase-locked loop of section 7.3, are subject to these constraints.

4.2.3 Discretization

Discretized trajectories $\Phi_t(z_0)$ are constructed from the state-space trajectories $\phi_t(x_0)$ generated by `runge-kutta4` using the region boundaries and cell divisions h_i. On figure 4.3, where the corners are (-π/2, -25), (π/2, 25), the trajectory

Figure 4.3: Discretizing a trajectory: this ten-point state-space trajectory $\phi_t(t)$ starts at the point (-.785 -4.167), the center of the grid cell `#(1 2)`. The discretized version of this trajectory, $\Phi(z)$, is `((#(1 2) .01) (#(1 3) .09))`. Region corners are (-$\pi/2$, -25), ($\pi/2$, 25) and $m = 6$.

$\phi_t(-.785 - 4.167)$ emanating from the centerpoint of the cell `#(1 2)` looks like:

```
(#(.01 -.8243301481988154 -3.69606170664237)
 #(.02 -.8588665426793466 -3.2087127367357464)
 #(.03 -.8884587196329566 -2.7076548013206416)
 #(.04 -.9129824248098427 -2.1954285322680795)
 #(.05 -.9323380339115039 -1.6744214660178298)
 #(.06 -.946449102403905 -1.1468856530999094)
 #(.07 -.9552611320528511 -.6149630837455898)
 #(.08 -.9587406229724326 -.08071704932096757)
 #(.09 -.9568744610672818 .4538324556428124)
 #(.10 -.9496696718366109 .986671176123371))
```

and Φ_t`(#(1 2))` looks like:

```
((#(1 2) .01) (#(1 3) .09))
```

Each element of Φ_t contains a cell vector and the time of entry. In the first element, the time of entry reflects a single timestep (0.01.) The timestep that was used to generate a trajectory can be determined in this way and thus need not be explicitly carried through the run. If it is negative, the trajectory is running in backwards time. The state-space point $\vec{x}_0$ that was used as a trajectory's initial condition can be determined from the cell vector $\vec{z}_0$, given (1) the boundaries and division of the region and (2) the origin and destination points. The reverse engineering of the timestep and the starting-point are combined in the function `find-starting-info`. The time spent in a particular cell is the difference between its timestamp and that of the next cell; e.g., for cell `#(1 2)`, $.09 - .01 = .08$ time units.

4.3 Dynamics Classification

Perfect Moment classifies discretized trajectories in terms of the ω-limit sets to which they relax. Only dissipative systems are treated in this thesis; extensions to conservative systems — which do not have attractors, as their equations preserve state-space volumes — would require adapting the classification scheme described

Figure 4.4: Examples of trajectories that are misclassified as "enroute to a fixed cell" because the grid is too coarse

below to recognize island chains, KAM torii and other features of Hamiltonian chaos[2].

Four types of attractors exist in dissipative chaotic systems: fixed points, limit cycles, quasiperiodic orbits, and chaotic attractors. The finite extent and discretization of the region under investigation alter their apparent properties and also add two possibilities to the list of possible $\omega-$limit sets from which the program chooses the classification of an individual trajectory:

- fixed cell
- period-n limit cycle
- chaotic attractor
- sink cell
- unknown

Unstable or metastable fixed points and limit cycles are omitted from this list because of the low probability that a cell-center initial condition will land in such a measure-zero set; even if this were to occur, the integrator's local truncation error would rapidly move the trajectory away. Quasiperiodic orbits were omitted by mistake; the implications of this oversight are discussed later in this section.

A trajectory that relaxes to and remains within a single cell is defined as a fixed point, regardless of what it is doing inside that cell. See figure 4.4. *Perfect Moment* defines fixed *cells* rather than fixed *points* to emphasize the distinction. This resolution-induced myopia has a variety of interesting implications that are explored later in this section. The Scheme `classification` data structure for a trajectory whose $\omega-$limit set is a fixed cell looks like:

```
(relaxing to the fixed cell #(2 46 13))
```

In a linear system, *Perfect Moment* would classify all trajectories as relaxing to the single globally-asymptotic fixed point — or to the sink cell, if that fixed point is out of range or unstable.

Limit cycles appear in discretized trajectories as finite repeating sequences of cells. A trajectory relaxing to the period-one limit cycle in figure 4.5 would be classified as:

[2]This type of analysis is performed by Ken Yip's KAM program[84].

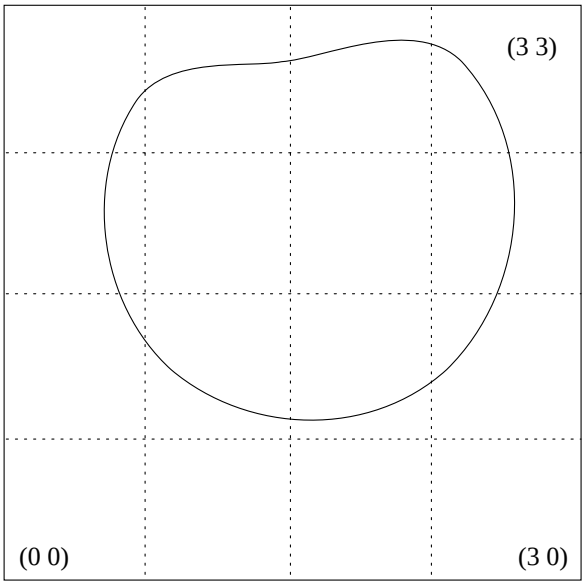


Figure 4.5: A period-one limit cycle

```
(relaxing to the period-1 limit cycle
((#(1 1) 0.00) (#(2 1) 0.65) (#(3 1) 1.25)
 (#(3 2) 1.67) (#(3 3) 2.29) (#(2 3) 2.84)
 (#(1 3) 3.45) (#(0 3) 4.09) (#(0 2) 4.29)
 (#(0 1) 4.88) (#(1 1) 5.03))
```

The `classification` of this structure contains only a description of a single orbit; its initial time is artificially offset to zero in order to facilitate periodicity computations. Note that the elapsed time in cell `#(0 3)` — 4.29 minus 4.09 or 0.20 — reflects how much less of the trajectory passes through that cell. Like fixed cells, limit cycles can be misclassified if the cell size is large.

Chaotic attractors are *non*repeating sequences of cells:

```
(relaxing to the chaotic attractor that touches the cells
 (#(1 1) #(1 2) #(1 3) ... ))
```

The set of cells touched is important in the determination of when an attractor approaches a control destination. If the grid is coarse, several orbits may be required to see the chaotic trajectory deviate from apparent periodicity, particularly if it wanders near an unstable periodic orbit (UPO) and stays near it for a few cycles[3].

A trajectory that exits the grid is defined as follows:

[3]This type of event is actually not as rare as it might seem; see section 6.2.

`(relaxing to the sink cell)`

The "unknown" classification (`unknown`) takes care of the Ross Perot problem[4]. The most common cause of such a classification — transients that have not died out — could be avoided by lengthening the integration or expanding the region as described in section 4.2.1.

In order to extract and characterize a trajectory's ω-limit set, the transient section must be stripped off and the balance examined for the symptomatic cell-sequence patterns. A synopsis of the algorithm that performs this classification is given below. It handles transients by working backwards from the end of the trajectory.

1. Find last cell $\vec{z}_l$ and time of entry t_l of the trajectory Φ_t
2. If $\vec{z}_l =$ `#(sink-cell)`, the destination is the sink cell
3. If $t_l < 0.5\times$ (total trajectory time), the destination is the fixed cell $\vec{z}_l$
4. Look back through Φ_t for the next occurrence of $\vec{z}_l$, recording the sequence of cells touched
5. Repeat step 4 four more times (or to the beginning of the trajectory)
6. If the pattern is constant, $\vec{z}_l$ is on a limit cycle; extract the core sequence and period
7. If the pattern is *not* constant, but the set of cells touched *is*, the destination is a strange attractor
8. Otherwise, the destination is unknown

The trajectory length choice — 1000 steps or ten orbits — and the factor of 0.5 in step 3 above address the sequence recognition and transient elimination issues. The implicit assumptions here are (1) that trajectories will equilibrate after less than five "natural frequency" periods and (2) any periodic behavior can be recognized after five repetitions.

Of course, these assumptions may be completely invalid. They depend on the success of the length and Δt choices and on the system: its underlying form and the values of its parameters. In a stiff system with poles separated by d decades, there will be $100/d$ timesteps in the slowest orbit; if $d > 5$, the $\omega-$limit sets of all trajectories will automatically be either `fixed cell` or `sink cell`. Even in non-stiff systems, coefficient changes can cause problems: damping coefficients have the obvious effects on whether a five-orbit period is adequate to attain — and

[4]a mixture of "none of the above" and "who knows what it's doing anyway."

recognize — equilibrium. This, again, may cause misclassification, typically in the form of limit cycles that are mistaken for strange attractors and fixed points or quasiperiodic orbits that are mistaken for unknown attractors. Finally, the choice of five orbits makes *Perfect Moment* unable to recognize a period-n limit cycle for $n > 5$.

Solving all of these problems would simply require more integration. It is impossible to determine, without actually performing the run[5], what length will suffice, but a better measure of the convergence rates could be obtained by computing the largest *negative* Lyapunov exponent[6]. Other logical approaches would be (1) to increase the number of steps until the classification reaches a steady state or (2) to multiply the trajectory length by some fixed factor if no trajectory exits the grid but all attractors remain `unknown`.

A second problem is the degradation of resolution and accuracy with increasing cell size. In the Rössler scenario, for example, the mapper was unable to distinguish between the period-one and the period-two limit cycles because of the coarseness of the discretization. Strange attractors and limit cycles can be mistaken for one another; fixed cells can be confused with almost any other member of the list. In the limiting case of $m = 1$, in fact, *all* trajectories are destined either for the fixed cell `#(0 0 0 ... )` — the only cell in the grid — or for the sink cell.

One possible solution to this problem would entail repeating the run with smaller and smaller cells until the classification results themselves reach a steady state. This solution would be extremely expensive, as run time is exponential in the number of cells on each grid axis. Alternatively, if one is attempting to distinguish between a limit cycle and a strange attractor, one could forego analysis of the discretized trajectories and compute Lyapunov exponents directly from the raw state-space trajectories ϕ_t. This scheme would also allow quasiperiodic orbits to be distinguished from strange attractors, which is how they are currently (mis)classified if they require more than five minor orbits per major orbit.

The final problem is the region's limited extent. *Perfect Moment* blindly generates m^n `#(sink-cell)` trajectories from every cell in the basin if any part of an attractor falls outside the grid, even though a human expert could easily extrapolate from the partial data on the visible region. As a first cut at solving this, one could allow trajectories to extend beyond the region boundaries; the return of any trajectory that exits the grid would be a clear indication that the region under consideration was chosen too small. Another solution is proposed in section 4.4.

The apparently-arbitrary design choices (transients limited to less than five orbits, etc) are trial-and-error, seat-of-the-pants compromises, based on intuition, observations of the systems, and patience level: they are sufficient to cause *Perfect Moment* to find reasonable solutions in less than the time it took for its author

[5]Note the common thread in the chaos definitions on page 33

[6]λ_1 of the *negative*-time evolution; see section 3.2.1.

to get frustrated. They are inputs to and coefficients in Scheme procedures and are easy to change. Many of the heroic measures that a user should attempt if a search fails involve these coefficients; see Appendix A for more details.

In summary, this classification scheme is *not* precise, nor is it intended to be. There are several reasons for this. First, the algorithm that uses the mapper's output examines it only on the scale of the current cell size. Second, an exact method will not work on noisy, experimental data. Finally, an engineering system — with real sensors, actuators, finite-precision arithmetic, etc — operates in a hard-to-measure and uncertain world, so high precisions are moot.

4.4 Parameter Spacing Between Portraits

The algorithms in this section — the collection of operations in the large dotted grouping in figure 4.1 — use the results of the classification scheme in the previous section to select parameter values at which to construct portraits so as to sample all of the interesting and useful dynamics within the specified parameter range. This necessarily involves some assumptions about what constitutes "interesting and useful."

Portraits are constructed and analyzed at each $k_{low} + n\Delta k$ for $(k_{low} + n\Delta k) \in [k_{low}\ k_{high}]$, using the machinery described in the two previous sections. If two successive portraits, at k_i and k_{i+1}, are significantly different, a portrait is constructed at $\frac{k_i+k_{i+1}}{2}$. *Perfect Moment* bases this decision on changes in the dynamics classification, defining the following as "significant:"

- bifurcations
- movement of an attractor into or out of the cell that contains the destination

The first criterion is both interesting to nonlinear dynamicists and useful from a control standpoint. The path finder ought to be very interested in fixed points near the destination; it can also exploit certain properties of strange attractors, but only if it knows that they exist. On the other hand, as discussed in conjunction with figure 4.2, it is sometimes wasteful to bother classifying the dynamics. The second criterion is purely control-oriented. Recall that the destination mentioned in the second item can be the control objective *or* another segment's endpoint, depending on the stage of the process. It might be interesting to try turning off the first criterion for searches whose control objectives do not dictate the use of a particular type of attractor. A variety of other useful additions to this list — that have powerful and far-reaching implications — are discussed in chapter 8.

A bifurcation is signaled by:

- a change in attractor category (the list on page 53)
- a change in a limit cycle's period

A variety of spurious bifurcations are detected when the classification scheme makes mistakes. In the Rössler example in chapter 2, for example, a limit cycle was reclassified as "unknown" after it grew to touch the sides of the region and was truncated:

```
(zeroing in because of bifurcation between
  (limit cycle at param = .35) to
  (sink-cell at param = .375))
(map: parameter = .3625)

(zeroing in because of bifurcation between
  (limit cycle at param = .35) to
  (sink-cell at param = .3625))
(iteration depth reached!)
```

This limit cycle, by the way, is the same one whose bifurcation from period one to period two was overlooked; all of these problems stem from the large cell size that was used in that scenario.

To account for cases where transient length or encounters with the region boundaries may pose problems for some trajectories in the basins and not others, the `unknown` classification received special treatment. If *some* of the trajectories mutate into `unknown`s but the others remain on the same attractor, *Perfect Moment* infers that the attractor still exists. One case where this inference solves an existing problem is when an attractor's damping moves through the range where detection after five cycles is problematic. To solve some of the cutoff problems, this scheme could also be extended to `sink-cell` trajectories, perhaps incorporating some stored information about what face of the search region the attractor was expanding towards,

The zeroing-in process is repeated up to $D_i - 1$ times, where D_i is the user-specified iteration depth. This parameter has several purposes. It limits how carefully *Perfect Moment* refines its perception of the positions of the boundaries between perceived bifurcations. This limited depth has clear practical implications if the resolution of the actuators and sensors is known. If, for example, full scale for the actuator is 0 to 12 units and it is transmitted to the system through a 8-bit DAC, choosing an initial Δk of 3 volts dictates a maximum iteration depth of seven[7]. D_i could be used to bypass some or all of the dynamics classification: the mapper will not zero in after the D_i^{th} pass, so classification on that pass is moot. D_i also lets the user limit accuracy for, e.g., a first cut at a design.

[7] $\frac{\Delta k}{2^{D_i-1}} \geq \frac{12}{2^8}$.

With the values in the previous paragraph, if the mapper recognizes a single, easy-to-classify bifurcation at $k = 1.6$, portraits would be constructed at the following k values: [0, 1.5, 1.6875, 1.875, 2.25, 3, 6, 9, 12]. Figure 4.6 gives a more graphic view of a slightly more complex scenario where $n = 3$, $\Delta k = \frac{1}{4}(k_{high} - k_{low})$ and $D_i = 4$. *Perfect Moment* zeroes in twice on the figure; once because a limit cycle bifurcates into a strange attractor and once because that strange attractor enters the destination cell. The tickmarks on the parameter axis indicate the iteration depth at which each portrait was constructed: heavy, normal, dashed and dotted for passes 1, 2, 3, and 4.

Some interesting pathological situations are created by the discrete sampling of the dynamics. For example, the initial parameter range and step might cause successive portraits to land on period-three limit cycles on either side of a chaos band. This coincidence is unlikely — though not impossible — and there is no easy way to avoid the resulting lack of distinction[8]. A similar but more tractable problem would arise if successive portraits showed period-two and period-three limit cycles, implying the existence of an interstitial chaos band. Inference rules that use this kind of nonlinear dynamics knowledge, much like those used by the KAM program[84], would not only speed the portrait-spacing process, but also make it far more intelligent and interesting.

The machinery described in this chapter produces a set of portraits, like those in figure 4.6, at parameter values spaced so as to be representative of the system's behavior. To accomplish this, the distribution and characteristics of the trajectories on each member of the set are automatically chosen so that individual portrait is representative of the dynamics and recognizable to the program. Nonlinear dynamics knowledge is used to guide the process so as to meet these requirements without excessive computational complexity.

[8]It turns out that the intervening topology changes often leave recognizable signatures in the dynamics; see section 7.2.

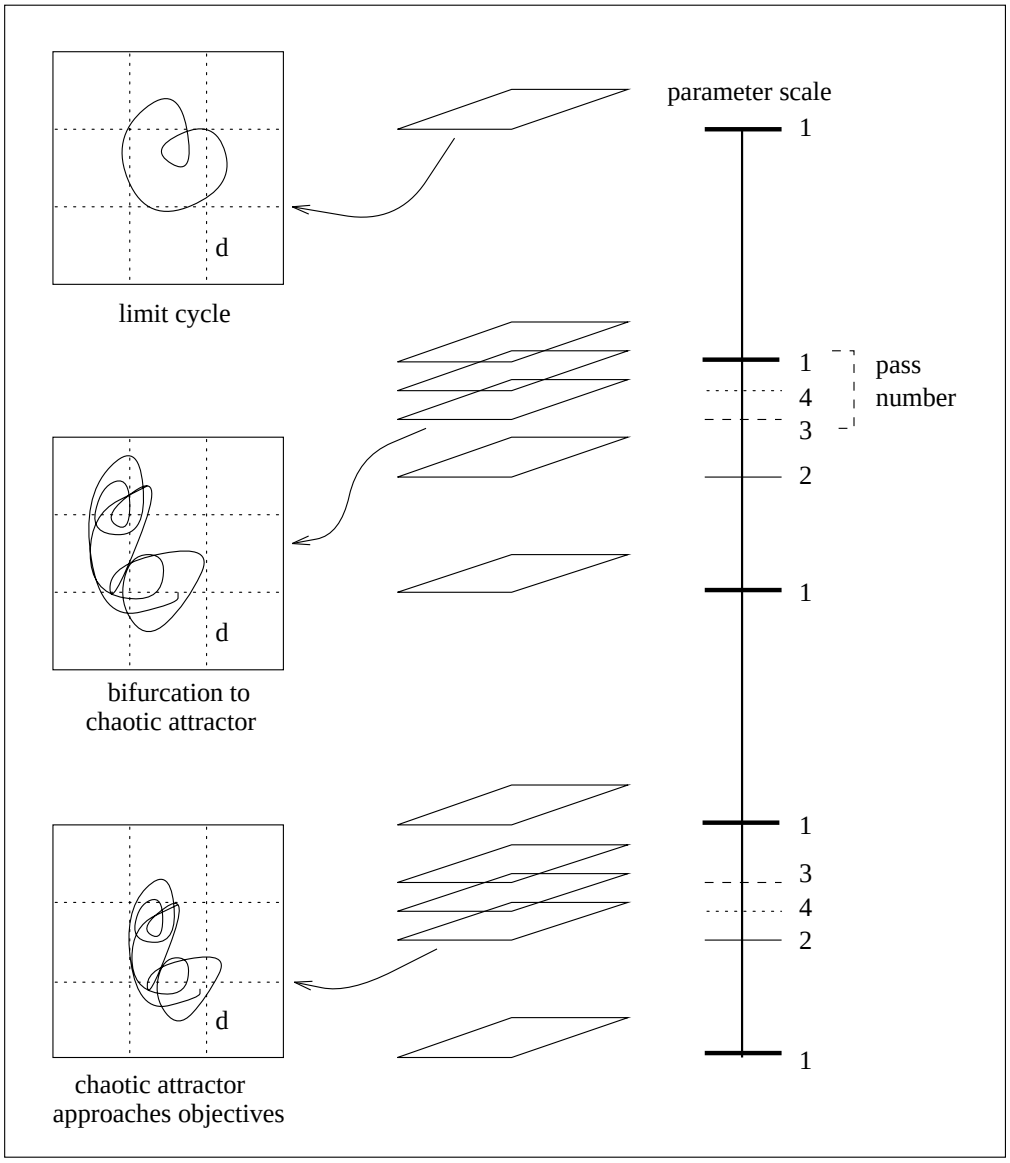


Figure 4.6: A stack of state-space portraits produced by the mapper: the initial parameter step was one-fourth the parameter range (the vertical line) and the iteration depth was 4. The mapper reduced the parameter step in two ranges, once because of a bifurcation and once because an attractor entered the destination (**d**) endpoint cell

Chapter 5

Segment Selection and Path Synthesis

The path finder searches the state-space portraits that are constructed by the mapper for a set of trajectory segments that meets the control objectives. Useful segments may be drawn from any region on any portrait — they may be sections of attractors or of transient trajectories that have not yet reached an attractor.

Figure 5.1 shows a flowchart of the path finder's operations; section 5.1 describes the flow of control between the different components of this figure. Section 5.2 covers segment selection, section 5.3 the work that is necessary to coordinate region and cell-size information between successive passes, and section 5.4 the programs that account for and are affected by gaps in the path (tolerance checking, computing the control recipe, etc.)

5.1 Structure of the Search

Each path finder invocation is an attempt to connect two state-space points. A gross path between the endpoint cells containing the origin and the destination is found first; see figure 5.2. The path finder then recursively calls itself to connect the endpoints of this *core segment* to the control objectives. Thus, at the n^{th} level, *Perfect Moment* has up to 2^{n-1} instantiations of the path finder on the recursion stack and up to 2^n leaf-node processes working to connect pairs of points. On any branch, the paths found at the $(n+1)^{st}$ level are usually smaller than those found at the n^{th}. See figure 5.3. The tree will be narrower by one branch for each segment endpoint that lands close enough to an objective to meet the tolerance, e.g., if the path finder chooses $\phi_t(\mathbf{o})$ or $\phi_{-t}(\mathbf{d}.)$ In that case, it only needs to work on the other end. If the core segment on part (a) of figure 5.3 emanated from the control origin **a** instead of the point **c**, the branches enclosed by the dotted line on

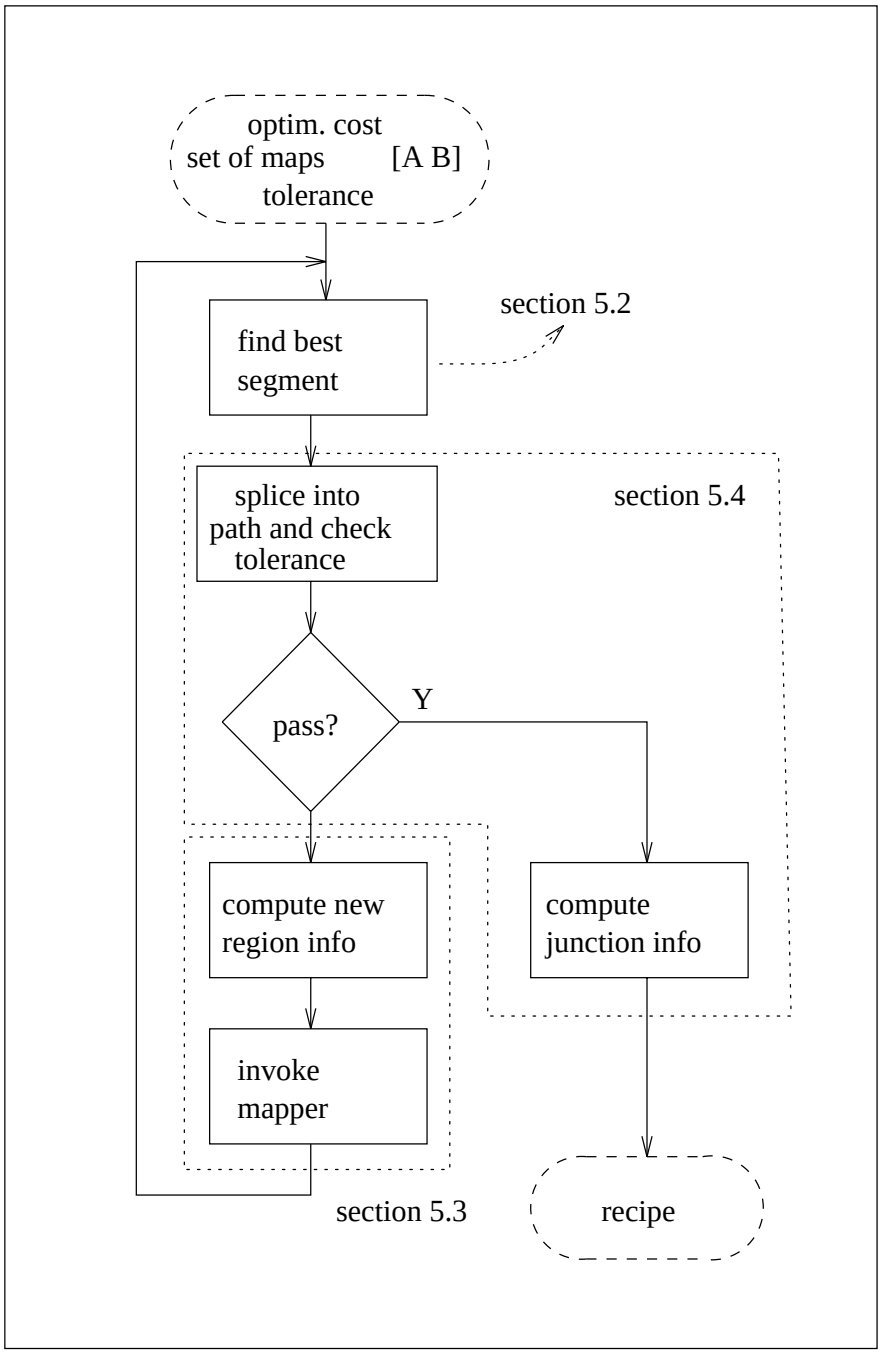


Figure 5.1: The path finder's flowchart: dotted boundaries group the topics according to the section where they are discussed

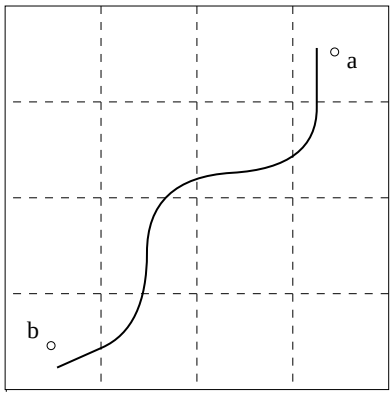


Figure 5.2: A rough-scale path between two state-space points. *Perfect Moment* looks for a trajectory segment connecting the cell that contains the origin (**a**) and the cell that contains the destination (**b**)

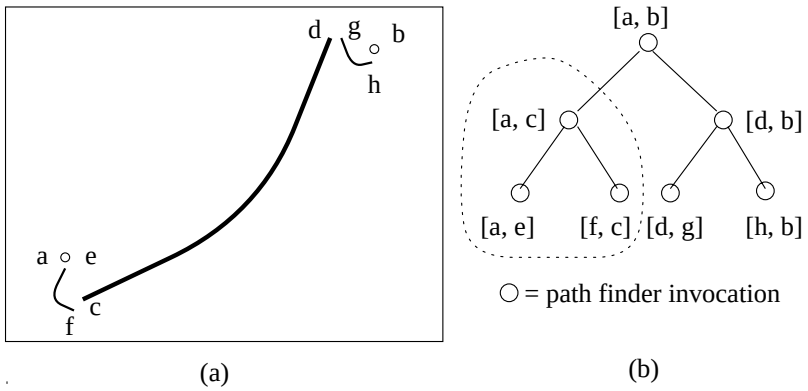


Figure 5.3: A segmented path (a) and the path finder calls that created it (b). [x, y] denote pairs of points to be connected at each pass; later calls are lower down in the tree and connect closer pairs

part (b) would not exist.

The optimization criteria used to evaluate the segments consist of user-specified Scheme procedures that operate upon time and state variables. Euclidean length, relative to the specified axes, is minimized in most of the examples in this thesis. Changes to and generalizations of this choice are obvious and easy; some interesting ones are suggested in section 5.2 and the code for several is given in appendix A.

The interleaved mapping and search processes, repeated on finer and finer scales, fill the gaps in the evolving partial path until the resulting ensemble of segments meets the specified tolerance. Tolerance checking is not simply a matter of halting when the endpoint of the last segment found falls within a specified distance of the control destination. The errors introduced by residual gaps *throughout* the path must be evolved forwards and factored in, as must the influence of the linear controller that aids in recovery from errors made at the segment junctions. This computation is fairly involved and is addressed in section 5.4. The implementation of the junction controller itself is discussed in chapter 6.

Because each path finder call is similar, the terms "origin" and "destination" can be confusing. On the first pass and on the outside leaves of the 2^n calls at the n^{th} pass, the actual top-level control objectives — **a** and **b** on figures 5.2 and 5.3 — are involved, but on all others, the origins and destinations are endpoints of partial paths. In both this chapter and the previous one, the distinction is often immaterial because the discussion concerns the unvarying function of program modules. Where it is important, as in the tolerance computations and in section 5.3.1, the terms *control* origin or *control* destination (sometimes *ultimate* as well) are used.

The search process could be (and has been) likened to dynamic programming. Another question that commonly arises at this point is "This looks like standard AI search — how does it *use chaos*?" The answer to both concerns is that *Perfect Moment* uses nonlinear dynamics and properties of chaotic systems to make the estimates and choices that minimize computational complexity and enhance its ability to attain control objectives. It *is* in some sense a dynamic programming approach: one wherein nonlinear and chaotic dynamics are used to prune choices off the search tree. Domain knowledge is "used" as a human engineer would: heuristics about the desirability of certain kinds of things (i.e., fixed points at the control destination) and about how these things arise and mutate are used to search out and exploit those features.

5.1.1 Data Structures

The search process uses a data structure called an `evaluation` that contains information about the optimality weight of a particular trajectory, discretized or not. Evaluations look like:

```
(metric intermediate-value trajectory)
```

and are manipulated in the obvious way[4] by the selectors and constructor:

- `get-metric`
- `get-intermediate-value`
- `get-trajectory`
- `make-evaluation`

The `trajectory` slot holds an entire trajectory. The `metric` is computed from that trajectory by the user-specified optimality functions discussed in the next section. The `intermediate-value` slot is used, for example, to hold an interim partial path while comparisons are made down the rest of a trajectory's length.

Partial paths are represented as lists of `segments`, which look like:

```
(initial-point final-point
 parameter
 corners
 variational-derivative
 sensitivity-derivative)
```

`parameter` is a real number, the `derivatives` are procedures, and the other entries are n-vectors. These structures are manipulated with similar constructors and selectors. In addition to the endpoints, the variational and sensitivity derivatives and the parameter value of the map from which the segment was selected, grid management requires information about the corners of the region to be carried along through the search. The linearized information about responses to state and parameter changes at the endpoints is also used by the tolerance computation and the junction controller.

5.2 Finding the Best Segment in a Stack of Portraits

Segment selection from a portrait — a list of m^n discretized trajectories — proceeds in two stages. First, the trajectories that are clearly inadequate are eliminated. For those that pass this filter, full state-space versions are then reconstructed and tested, narrowing the field to a single result.

5.2.1 Rough-Scale Filter

An *optimization function* takes a discretized trajectory $\Phi_t(\vec{z}_0)$ and computes an `evaluation` that represents a rough estimate of its optimality weight. Optimization functions typically minimize or maximize some function of some subset of the state variables, according to the requirements of the application and the design. For example, a function that finds the minimum pathlength along a trajectory between two given cells would simply count the number of cells between all in-order occurrences of those cells, then return an `evaluation` containing the minimum of that count, the trajectory segment that starts at the appropriate point, and the full trajectory itself, the latter for purposes of reconstruction and tiebreaking:

```
(pp traj)
==> ((#(1 2) .01) (#(1 3) .07) (#(2 3) .19)
     (#(3 3) .27) (#(4 3) .35))

((shortest-path-between-two-cells '(#(1 3) #(3 3))) traj))
==>(1
    ((#(1 3) .07) (#(2 3) .19) (#(3 3) .27) (#(4 3) .35))
    ((#(1 2) .01) (#(1 3) .07) (#(2 3) .19) (#(3 3) .27)
     (#(4 3) .35)))
```

In this example, a single cell separates the occurrences of the cells `#(1 3)` and `#(3 3)`; `get-metric` applied to the evaluation returned by the optimization function (`(shortest-path-between-two-cells ... traj)`) would return 1. If several paths exist in `traj` between the specified cells, those that share the lowest metric value are returned in reverse order of encounter. The intermediate value slot contains the segment whose head is the shortest sequence of cells in `traj` that starts with `#(1 3)` and ends with `#(3 3)`.

Metrics to fit almost any specification can be constructed using appropriate manipulations of the cell vector and/or the time of entry. Distance can be measured along any subset of the coordinates using one of a variety of different norms. Functions of the coordinates (e.g., βv^2 for friction) may be useful quantities to minimize or maximize. In applications that dictate state-space regions to be sought out or avoided, functions can assign relative weights to individual grid cells in many different ways. For example, δ_{ijk} can distinguish cell `#(`ijk`)`; some smoother discontinuity or perhaps even a fuzzy weighting function[1] can be used to weight groups of cells. One could minimize or maximize trajectory *speed* — or aim for a particular time of arrival — simply by using the difference between the timestamps. Examples of some of these are in appendix A.

The function `find-best-trajectory` selects the best trajectory from a portrait, applying an optimization function and keeping track of the running total of

[1]This approach was suggested to me by Benoit Cellier.

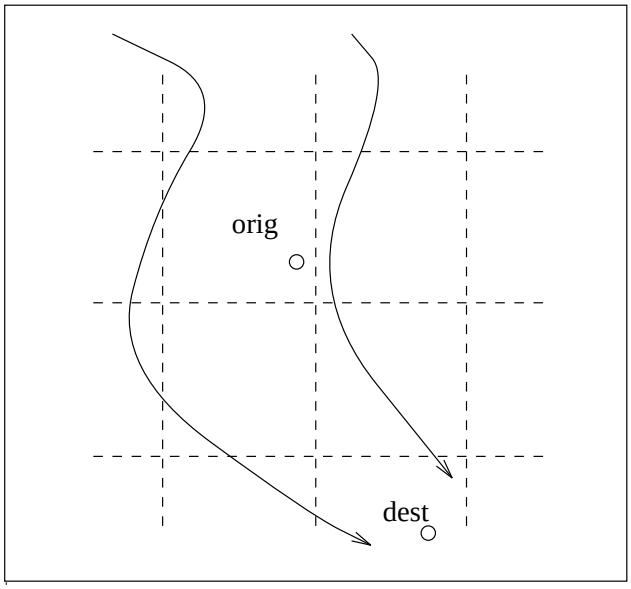


Figure 5.4: A case where the path finder fails to find the best trajectory because it considers only those that enter the endpoint cell

the cost. It turns out to be more efficient[2] to first filter out any discretized trajectories that completely miss the specified cells before applying the optimization function:

```
(find-best-trajectory
   'minimize
   (ones-that-touch-in-order portrait orig-and-dest-cells)
   (shortest-path-between-two-cells orig-and-dest-cells))
```

The `minimize` token tells `find-best-trajectory` how to optimize the results of applying the function

```
(shortest-path-between-two-cells orig-and-dest-cells)
```

to the prefiltered trajectory

```
(ones-that-touch-in-order portrait orig-and-dest-cells)
```

Trajectories that do not enter both endpoint cells (the cells that contain the two points to be connected) are not considered. Thus, *Perfect Moment* will make a less-than-optimal choice in a situation like the one depicted in figure 5.4, where the right-hand trajectory is clearly better but misses the origin cell. This type of problem is inherent to the gridded search; it could be fixed using more-intelligent region-selection heuristics and is discussed further in section 5.3.2.

[2] e.g., one-third faster for a representative case of a 64-trajectory driven pendulum portrait where three trajectories touched the specified cells in order and all three had the same metric

Time step magnification	Time without plotting	with plotting
1	0.185 sec	0.340 sec
2	0.215 sec	0.416 sec
5	0.311 sec	0.561 sec
10	0.475 sec	0.773 sec

Table 5.1: Effects of time step magnification on *Perfect Moment*'s run time. This table is based on a single map/search pass (four portraits constructed on a 4×4 grid) of the driven pendulum ODE on a Hewlett-Packard 9000/720 workstation. The graphics add significant overhead — constant and multiplicative — because all m^n trajectories are reconstructed before being plotted

5.2.2 Fine-Scale Reconstruction

A *tiebreaker function* takes a discretized trajectory $\Phi_t(\vec{z}_0)$, reconstructs the state-space version $\phi_t(\vec{x}_0)$, and returns an evaluation reflecting its optimality weight. The Δt and the initial condition are reverse-engineered from the first trajectory element of $\Phi_t(\vec{z}_0)$ by the function `find-starting-info` of section 4.2.3. $\phi_t(\vec{x}_0)$ is reconstituted from this information using a time step that is *smaller* than the initial one by a factor κ. Currently, $\kappa = 2$, which is a tradeoff between improved integrator accuracy and increasing run time. See table 5.1. On all four portraits involved, the optimization function filtered out all but two of the 16 trajectories; the time taken to do so is roughly constant for a given grid, as every discretized trajectory must be checked. The growth is roughly $O(\kappa^{0.35})$ if the trajectory set is plotted on the graphics screen and $O(\kappa^{0.4})$ if not. In general, these numbers depend on the percentage of trajectories that have to be reconstructed. If the problem and objectives are such that the ratio eliminated by the first pass is lower or higher than 14:16, the growth would be faster or slower, respectively, than in the two right hand columns of the table.

Reconstruction and in-line optimality assessment are together only negligibly slower than carrying around the entire pointset in memory, mapping down the list and computing the optimality cost. In reality, the heap is finite and clogging it up with trajectories affects the run time of the rest of the program, so reconstruction clearly wins.

The finer-scale look at the actual state-space points lets the tiebreaker function compute a better measure of the metric. Just as important, the smaller step size gives a better measurement of the dynamics. Very rarely, an interesting quandary arises: the better simulation causes the trajectory to move so much that it is no longer the best or even misses the cell. This effect can cause *Perfect Moment* to choose suboptimal paths.

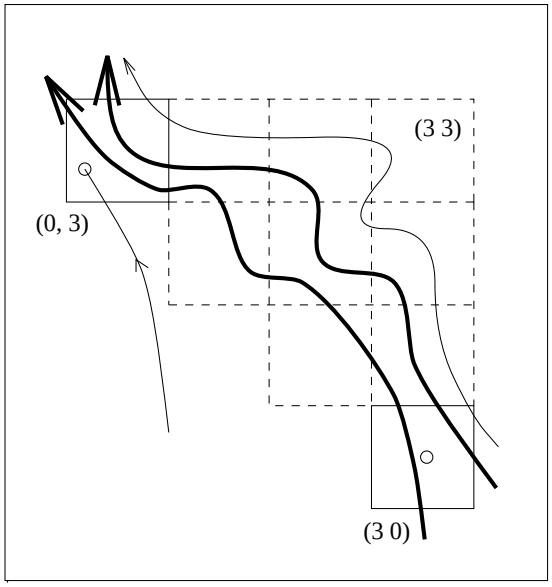


Figure 5.5: Two-pass algorithm for segment selection: the first pass selects the two thick trajectories, as they both touch five cells between origin and destination. The second pass chooses the lower of the two, as its endpoints fall closer to the objectives

Because of this finer resolution — and depending upon the form of the problem and the initial parameters of the run, as demonstrated above — tiebreaker functions are *at least* an order of magnitude slower than optimization functions. However, because the latter are used first to weed out clearly-useless trajectories, the tiebreaker functions are typically applied to at least an order of magnitude fewer trajectories.

The tiebreaker function used in the bulk of this thesis, `minimum-sum-distance-tiebreaker`, computes the sum of the euclidean distances between the objectives and the segment's points of closest approach to them. Note that this function is **not** just the $\Re^n$ analog of `shortest-path-between-two-cells`. The use of dissimilar optimization and tiebreaker functions is an effective way to synthesize two control aims. This hybrid approach combines global and local design goals: it relies on the cell size being small enough so that all of the candidates found by the optimization function are close to the same length, then uses a differently-focused tiebreaker function to favor trajectories that closely approach the control objectives.

The first pass of `find-best-segment`, which combines everything discussed in this section, applied to the portrait shown in figure 5.5, would select the two thicker trajectories, as they both touch five cells on their travels between the cells `#(3 0)` and `#(0 3)`, while the other trajectory touches six.

The second pass of `find-best-segment` would determine the initial conditions

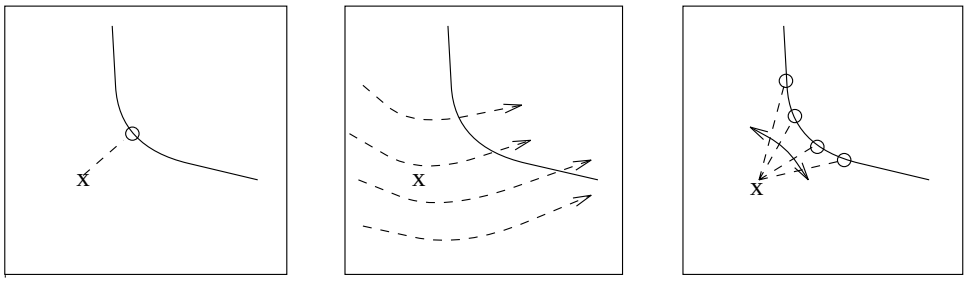


Figure 5.6: Extracting a segment from a trajectory: (a) ending the segment at the point of closest approach (b) a case where the vector field makes that problematic (c) a more-intelligent approach, where a *range* of endpoints are evaluated

and time steps for both trajectories, divide the time steps by $\kappa = 2$, repeat the integration, apply the tiebreaker function and select the lower one, as the sum of the distances between it and the control objectives is lower than the corresponding sum for the higher trajectory.

Finally, the program extracts a segment of the "best" trajectory on the portrait. *Perfect Moment* takes the easy way out and ends the segment at the points of closest approach to the objectives, as in figure 5.6(a). However, this extraction algorithm may cause problems on the *next* pass. For example, the vector field may only cross that gap obliquely for the allowed parameter values, as in part (b) of the figure. Worse yet, the flow may be completely opposite to the desired direction of travel. A more intelligent approach that solves this problem would be to search for connections between the objective and a *range* of points on the trajectory, as in part (c). However, such a scheme would increase the dimension of the target manifold and cause a one-step growth in the dimension of the search with each pass. A first approximation to this that does not increase the search space would be a tiebreaker function that chooses segments and endpoints by evaluating distances to objectives *and directions of vector fields*. For example, the backwards-time trajectories from the destination with $k = k_l$ and $k = k_h$ could be intersected with the trajectory to find the range of preferred values for the segment endpoint, or used to determine a range of vector directions to favor in the choice. Such a scheme would produce paths with local overshoots at switchpoints and control destinations, made in order to get to regions where the vector field goes the right way.

After a segment is chosen, it is compared to the best segments from the other portraits in the set. Finally, the variations and sensitivity derivatives are computed at the endpoints of the overall winner[3] and the entire structure is spliced into the

[3]In a real system, this "computation" would entail moving the system to the state-space point where the junction falls, making small parameter and state changes, and recording the local effects; this is how the derivatives are computed in [59]. Physically getting the system to an arbitrary state-space point to even perform this exploration may present serious problems; this is discussed at greater length in section 7.4.

Figure 5.7: Initial search region and discretization with $R = 2$ and $m = 5$. The bounding box of the origin (**o**) and the destination (**d**) is expanded by a factor of two along all state-space axes and each axis is divided into five intervals

controlled trajectory and checked against the specified tolerance. If it fails the test, the path finder proceeds to synthesize the information required to dispatch the two recursive calls that bridge the gaps between the segment that was just found and the two points that it was intended to connect.

5.3 Search Region Selection and Refinement

Perfect Moment's manipulations of the size and shape of the search region and of the resolution with which it is mapped are driven both by the control requirements and by the dynamics of the system itself. The grid and the rules that manipulate it guide the program's attention and effort to the regions where it is most required: because of interesting, complicated or useful dynamics or because of proximity to control objectives or to partial paths. This adaptive parameter spacing is spatially analogous to adaptive time step integration; where events occur rapidly, they should be examined more closely. A second, equally-important goal of this machinery is to allow counterintuitive moves — ones that send the system away from the apparently "correct" direction in order to reach a faster path. Extending the highway route-planning analogy, one would certainly choose to backtrack slightly if that detour led more quickly to an interstate[4].

Initial choices for both the discretization m and the overrange R, designated R_0 and m_0 and discussed in section 5.3.1, are made by the user, though *Perfect Moment* does provide some defaults. An R_0 value of 1.1 implies that the search region is formed by expanding the bounding box defined by the control origin and destination points by 10% in all directions. Choosing $m_0 = 3$ causes each edge of this region to be divided into three equal intervals, fixing the cell aspect ratio for the duration of the run. Figure 5.7 shows the situation with R_0 and m_0 set to the default values two and five.

These initial choices are revised dynamically by the program as the run — and the information about the system and the partial path — unfolds. Rules that use the variational system and the sensitivity derivative, evaluated at the points to be connected, are used to revise the grid spacing; the region boundaries are determined by the intermediate results of the search and the output of the dynamics classifier. Details of both revision algorithms are in section 5.3.2.

[4]Though the mechanism and the scale are somewhat different, the results are similar to the local-scale counterintuitive moves that would be produced by the suggested trajectory-intersection modification at the end of section 5.2.2.

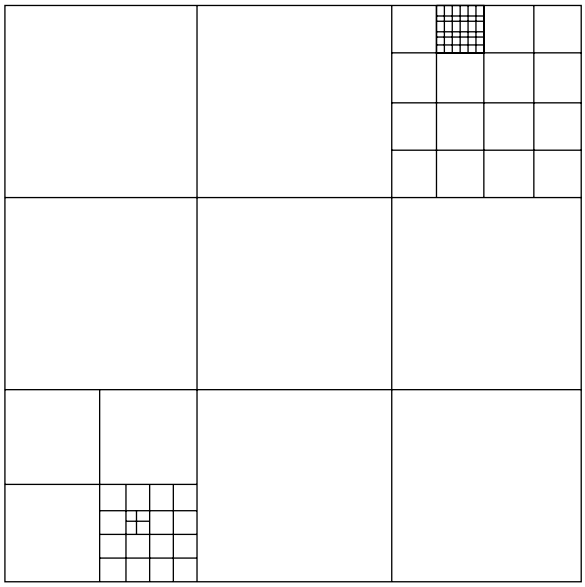


Figure 5.8: Variable-resolution grid: the local resolution is automatically adapted to the dynamics and control requirements in each region

An example of the grid spacing that this dynamic revision can produce is shown in figure 5.8. This particular grid pattern is an artifact of six segment searches: one with $m = 3$ in the entire region to find a core segment, three recursive searches with $m = 2, 4$, and then 2 to complete the lower left corner of the segmented path, and two recursive searches (with $m = 4$ and then 6) to complete the top right section. The spacing reflects local differences in either trajectory spreading or sensitivity; the top right is either more turbulent or more responsive to the control parameter than the bottom left. The only way to determine which of these two factors triggered the cell size step is to examine the transcript of the run. Similar pictures arise — for similar reasons but via different patterns and reasoning — in [57], wherein the area surrounding *every* point in the trajectory is expanded at each level and points are added in the interstices based on the results of the exploration. The basic difference between this scheme and *Perfect Moment*'s approach lies in the planning algorithms: the latter attempts to perform most of the control task with a single trajectory, then appends small actions to the ends; the former divides the task into more uniform subtasks and refines the subtask size downwards across the entire trajectory. Relative success of these two methods depends on the topology of the problem and the control requirements.

Whenever the search region is reduced, *Perfect Moment* can lose track of attractors because of truncation effects; see section 4.4 and page 24. At the same time, the enhanced resolution due to the finer grid used in that smaller region can bring smaller-scale attractors, like those hidden from the mapper's view in figure 4.4, into focus. At some point between the former — a portrait full of `(relaxing to the fixed cell #( ... ))`, frustrating a user who knows that the system is

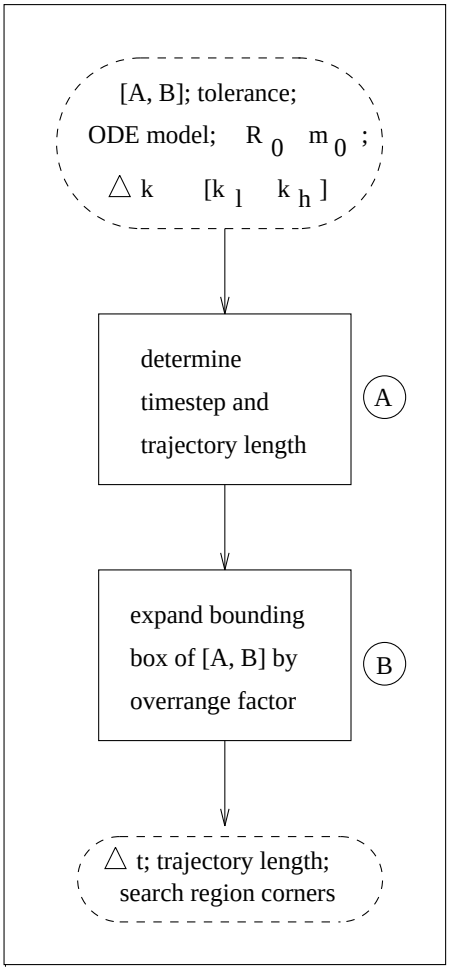


Figure 5.9: The initial operations required to set up the core segment search

richly chaotic — and the latter, where all trajectories go to the sink cell, exists an exploration scale on the order of each of the system's "natural" scales[5] within $[k_l\, k_h]$. Unless the user-supplied $[R\, m]$ or $[\Delta k\ D_i]$ combinations dictate otherwise, *Perfect Moment*'s revision rules cause it to automatically find those scales and explore the dynamics accordingly.

5.3.1 On The First Pass

The first pass requires some setup computations (the box marked **A** in figure 2.4;) these are depicted in more detail in figure 5.9. The determination of timestep and trajectory length is discussed in section 4.2.1; the region computations in the box labeled **B** are the topics of the remainder of this section.

[5] attractor size $\sim$ cell size

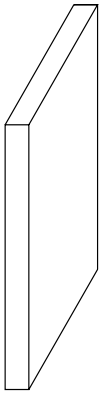


Figure 5.10: A possibly-problematic cell shape, dictated by the shape of the bounding box of the control objectives. If this shape differs greatly from the aspect ratios of the existing attractors, the short edge of this cell will drive the search region expansion to needless levels

If *Perfect Moment* restricted its attention to trajectories within the bounding box of the control objectives, counterintuitive moves would be impossible at the first level, where the scale is largest and they can exert the most leverage. The program uses $R_0 = 2$ as a default, but the user is free to specify another value (as long as $R_0 > 1$) or even a *vector* of values that defines a region of a completely different shape. The implicit assumption behind the factor of two is that recouping the optimality cost of a detour larger than the size of the distance to be covered is unlikely. This assumption is dead wrong if the vector field flows *from* the destination *to* the origin for all allowed parameter values. In this case, to travel from the latter to the former, one would have to find a way to use the larger-scale dynamics to get upstream from the destination, then ride the local dynamics back down. The only serious drawback to a larger R is that m must be higher to attain a given accuracy. If the user knows that the natural scale of the system's dynamics is much larger than the distance to be spanned, or that the local vector field goes exactly the wrong way for $k_l \leq k \leq k_h$, choosing a larger R is a good idea. Situations where one would wish to explicitly specify a different-shape region typically involve origin-destination pairs that are separated by a small distance on one axis and larger distances on others (or vice versa,) in which case the bounding box — and hence the cells *at all levels of the search* — would have an aspect ratio as in figure 5.10. The terms *small* and *large* here are clearly relative; if the attractor is tall and thin, a tall and thin cell may be an advantage. Regardless, hand-crafted changes should be allowed, but only if the user knows what he or she is doing, as bad choices can have fairly serious effects. Finally, for reasons that are discussed at the beginning of section 5.3.3, a value for R_0 must be specified *even when it is to be bypassed.*

The default value $m = 5$ is another seat-of-the-pants design choice, like the transient length on page 56. The main influences were personal perception of

computational and graphical complexity and the condition that m_0 must not be near an integer multiple of R_0 — the latter to prevent objectives from falling near a cell corner and causing problems like those shown in figure 5.4.

These "design choice" arguments are very far from rigorous. Firming up the underlying theory and using it to completely automate the choices would make *Perfect Moment* easier to use — particularly by novices — and much more palatable to theorists and mathematicians. Control theory and nonlinear dynamics would both play roles in these solutions, as would the wishful thinking about region revision that appears in chapter 4. For example, in the basin of attraction of an asymptotically-stable fixed point, only a few trajectories — those emanating from the ring of cells that form the discretized analog of the basin boundary —are necessary to characterize the dynamics. Some related issues that arise in problems of this nature are addressed in much more detail in [84], wherein domain knowledge about nonlinear systems is used to infer when important trajectories are missing from state-space portraits.

5.3.2 On Successive Passes

The same inputs — search region geometry, trajectory length and step — must be computed by the path finder and passed back to the mapper after every pass; after the first pass, they are based not on user input but on information about the partial path.

Having successfully completed a pass, the path finder would be in possession of a list of trajectory segments, each of which might look like:

```
(#(-.88 -2.06 -.74)                    ; startpoint
 #(1.93 -3.94 .67)                     ; endpoint
 .325                                  ; parameter value
 (#(-10. 4. -6.1) #(11. -10. 5.8)))    ; search region corners
 #[compound-procedure 23]              ; sensitivity deriv
 #[compound-procedure 24])             ; variational deriv
```

This particular segment was the result of a map/search invocation connecting `#(-1 -2 -1)` to `#(2 -4 .7)` with $m = 2$ in the Rössler scenario in chapter 2.4.

Suppose that the path has been analyzed by the tolerance algorithms of section 5.4 and judged to be inadequate because of one or both of the gaps between the endpoints of this particular segment and the objectives that it was supposed to connect. If neither gap is a problem, no further recursive calls are necessary on this branch. The path finder must compute the area(s) of and cell size(s) within the new search region(s), then dispatch the appropriate calls.

The next-level search regions are determined by:

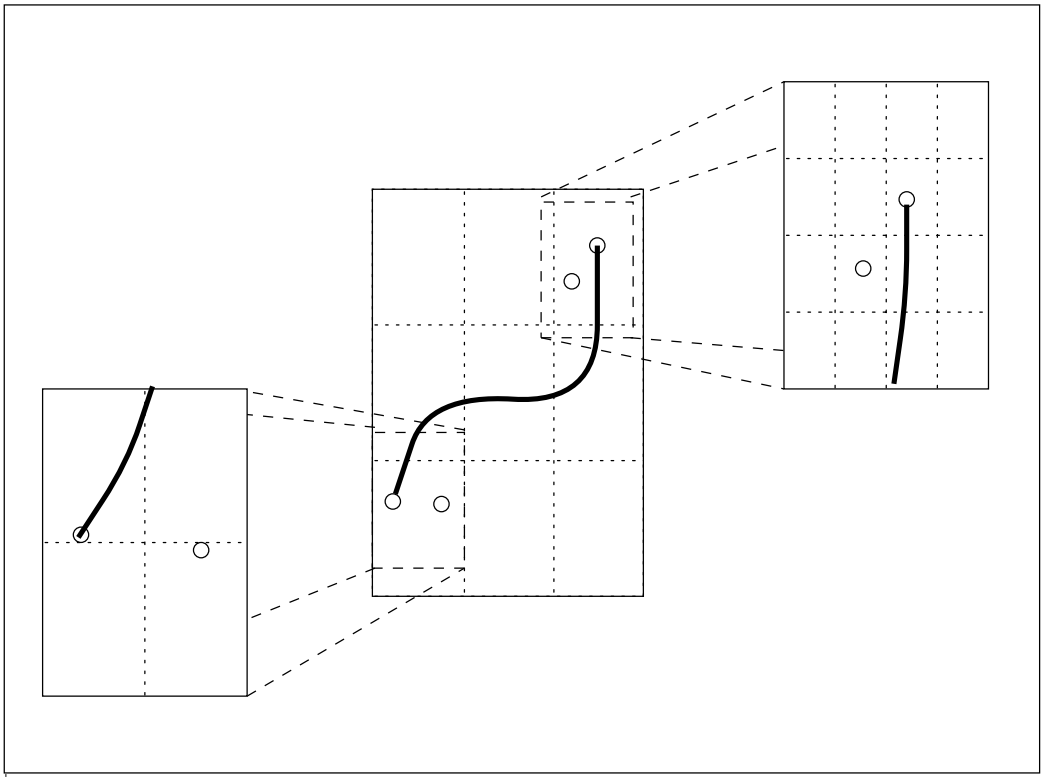


Figure 5.11: Refining regions between search passes: each endpoint cell is recentered around the pair of points that will be an [origin destination] pair on the next search pass. A new grid division is then computed in each region, reflecting the local dynamics therein. Note that the two regions are the same shape and size, but that the discretization can be different

- the cell size
- the origin and destination
- the segment endpoints
- the region corners

The endpoint cell — the parallelepiped that is

$|\frac{1}{2}$ `[#(11 -10 5.8) - #(-10 4 -6.1)]`$|$ = `#(10.5 14 11.9)`

in size and encloses the gap to be spanned — is first shifted so that its center falls on the midpoint between the new origin and destination; see the dashed boxes in figure 5.11. The shift is intended to allow for the situations shown in figure 5.12, where those points fall close to or even on opposite sides of a cell boundary. The situation shown in part (a) of the figure can arise, for example, if m_0 is close to an integer multiple of R_0. Part (b) can occur if the reduced reconstruction time step alters the dynamics sufficiently. This recentering strategy is critical: it allows

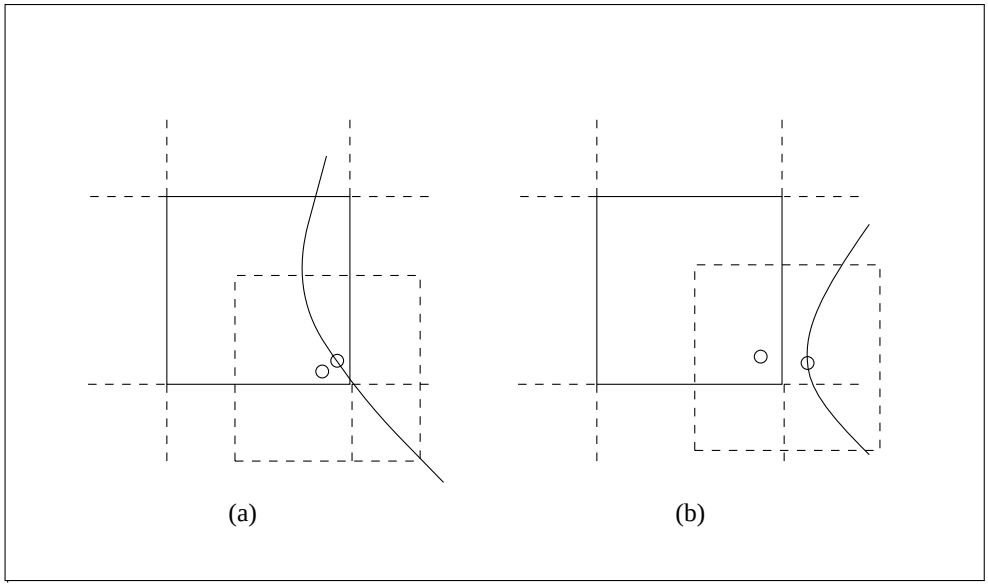


Figure 5.12: Problematic situations that can arise during the search and are partially solved by the shift of the new search region (dashed square) (a) important points near a cell boundary (b) numerical error that has caused a trajectory to move across a cell boundary

counterintuitive moves up to the size of the current grid square at any level of the search.

The sizes of the regions on both dispatched calls at any level are identical[6]: the dashed boxes in figure 5.11. However, the *discretizations* may be different in each. Within a particular branch of the tree, a segment found at the n^{th} level is usually smaller than one found at the $(n-1)^{st}$ level, but this does not hold across the entire tree, as the dynamics in different state-space regions dictate different grid divisions. In figure 5.8, for example, it is obvious that segments three levels deep in the top right are *not* larger than those four levels deep at the bottom left. Because of this, information about the corners used to choose a particular segment cannot be reconstructed from the original corners and level, so it must be carried along through the search.

After the first pass, the path finder is always connecting segments of unequal length[7], so all calls involve two different-size search regions: the one that was used to find the larger segment (e.g., the large solid box in figure 5.11) and the one used to find the smaller segment (the dashed boxes in figure 5.11.) No further expansion occurs unless the search fails; see section 5.3.3.

The path finder takes apart the `segment` data structure on page 75 — and its

[6]and inaccessible to the user at this level

[7]The control origin and destination are viewed as zero-length segments.

two neighbors — in order to build the next-level origin/destination pairs. It determines the regions from which those segments were chosen using `(subtract-vector high-corner low-corner)`, where `low-corner` and `high-corner` are the `car` and `cadr` of `get-corners` applied to the appropriate `segment`. This process is repeated for both `segments` that define each gap; the smaller of the two regions is divided into m chunks and recentered:

```
(recenter new-orig new-dest
  (scale-vector corners (/ 1.0 intervals)))
```

`intervals` is m and `corners` is the result of the `subtract-vector` call above.

The quantities that are used to determine the discretization for the next pass are:

- the current m
- the **new** origin and destination $\vec{x}_{orig}$ and $\vec{x}_{dest}$
- the variational system derivative

The basic idea behind most of the revision rules in the rest of this section is to choose the grid spacing so that a predefined variation — in state or in parameter — grows no more than roughly one cell size as a nominal trajectory travels through the endpoint cell containing the new origin. The defined state variation is a scaled version of the distance between the two points to be connected; in parameter space, the variation is the range $[k_l \;\; k_h]$ specified by the user. The reasons behind the latter should be clear. The former couples the control requirements into the exploration scale. The scaling indirectly defines an arbitrary threshold between "turbulent" and "not turbulent" and is discussed later in this section.

The next few pages illustrate the construction and use of the variational derivative, which measures the amount of "spreading" undergone by nearby trajectories and is used by the rules that manipulate the grid size between search passes. It is an indication of how errors evolve, so is also used in robustness calculations and in the computation of the eigenvalues and eigenvectors that are used in the programmable junction controller (chapter 6,) as well as in the determination of when a path meets the tolerance (section 5.4.)

Perfect Moment's `jacobian` function symbolically differentiates the input system to obtain the entries $\frac{\partial F^j}{\partial x_i}$, then compiles these expressions into a procedure that computes the derivative $\dot{\vec{\delta}}$ of the n^2-vector of variations $\{\delta_{ij}\}$. A combination of this procedure and the original system derivative is used to integrate the

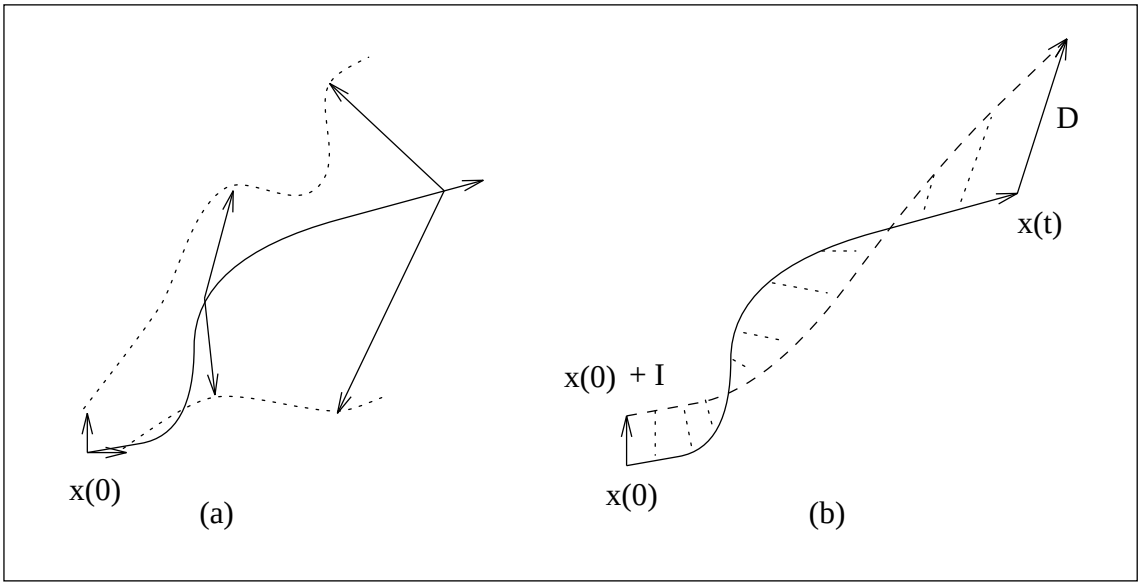


Figure 5.13: The geometry of the variational system: (a) the evolution of the variational system (b) system trajectories from $[\vec{x}_0]$ and $[\vec{x}_0 + \vec{I}]$

$(n^2 + n)$-dimensional variational equation

$$\left\{ \begin{array}{c} \dot{\vec{x}} \\ \dot{\vec{\delta}} \end{array} \right\} = \left\{ \begin{array}{c} \vec{F} \\ D_x \vec{F} \end{array} \right\}$$

from the initial condition

$$\left\{ \begin{array}{c} \vec{x}_0 \\ \vec{I} \end{array} \right\}$$

with $t = t_0$. The time evolution of the first n elements of this *augmented state vector* follows the trajectory $\phi_t(\vec{x}_0, t_0)$. The column sums of the matrix formed by the next n^2 elements are the evolved version of the initial variations, shown in part (a) of figure 5.13. Any basis can be used as the initial variation; $\vec{I}$ is simply convenient in cartesian space.

Note the approximation here: the state variables $\vec{x}$ in this vector are evolved using the full nonlinear equation, but the variations $\vec{\delta}$ are evolved using a derivative that is linearized around the starting point. Thus,

$$\left[\phi_t(\vec{x}_0, t_0) - \phi_t(\vec{x}_0 + \sum_{i=1}^{n} \hat{x}_i, t_0) \right] \neq \left[\sum_{i=1}^{n} \vec{e}_i(t) \right]$$

for all but $t \to 0$. The $\vec{e}_i(t)$ are the column sums of the time-evolved variational matrix. In other words, the separation between the two trajectories emanating from $[\vec{x}_0]$ and $[\vec{x}_0 + \vec{I}]$ — marked "D" in part (b) of figure 5.13 — is not necessarily

equal to the evolved variation (the sum of the two superimposed vectors in part (a) of the figure.)

One could certainly improve this accuracy by repeating the linearization and recomputing the variational-derivative procedure at every integration step, at the cost of $O(n^2)$ operations per point on the trajectory. Note, too, that the eigenvalues of the evolved variational state at each point are different from the eigenvalues of the Jacobian evaluated at that point. The latter are a local, instantaneous property of the system; the former depend on previous history and evolution.

Figure 5.14 gives a detailed example of this — the Lorenz system that is explored in section 7.1. Part (a) shows the information that is actually presented to *Perfect Moment*, part (b) the procedure that the program compiles from this information and passes to the numerical integration routines that evolve the system state, and part (c) the compiled Jacobian procedure that is used to evolve the variational system. The system derivative generator is a procedure that, given values for the parameters `a, r` and `b`, returns a second procedure — a system derivative — that takes a state vector `*state*` = (x y z) and a time `t` and returns a list whose components are the time derivatives of the components: $(\dot{x}\ \dot{y}\ \dot{z})$. *Perfect Moment*'s `runge-kutta4` integrator invokes this system derivative four times for each timestep. The Jacobian derivative generator is similar, but takes an *augmented* state vector

$$(x\ y\ z\ \delta_{xx}\ \delta_{xy}\ \delta_{xz}\ \delta_{yx}\ \delta_{yy}\ \delta_{yz}\ \delta_{zx}\ \delta_{zy}\ \delta_{zz})$$

and returns

$$(\dot{x}\ \dot{y}\ \dot{z}\ \dot{\delta}_{xx}\ \dot{\delta}_{xy}\ \dot{\delta}_{xz}\ \dot{\delta}_{yx}\ \dot{\delta}_{yy}\ \dot{\delta}_{yz}\ \dot{\delta}_{zx}\ \dot{\delta}_{zy}\ \dot{\delta}_{zz})$$

The Jacobian for the Lorenz system with parameter values $a = 16$, $r = 50$, $b = 4$, evaluated at the point (x y z) = (8 10 20) is

$$\begin{bmatrix} 16(\delta_{xy} - \delta_{xx}) & 30\delta_{xx} - \delta_{xy} - 8\delta_{xz} & 10\delta_{xx} + 8\delta_{xy} - 4\delta_{xz} \\ 16(\delta_{yy} - \delta_{yx}) & 30\delta_{yx} - \delta_{yy} - 8\delta_{yz} & 10\delta_{yz} + 8\delta_{yy} - 4\delta_{yz} \\ 16(\delta_{zy} - \delta_{zx}) & 30\delta_{zx} - \delta_{zy} - 8\delta_{zz} & 10\delta_{zx} + 8\delta_{zy} - 4\delta_{zz} \end{bmatrix}$$

The variational system integrated forwards from the initial condition (8 10 20 1 0 0 0 1 0 0 0 1) for 0.05 seconds — to the state-space point $\phi_{t=.05}(\vec{x}_0)$ = (13.087 23.455 24.204) — is roughly

$$\begin{bmatrix} .79 & .95 & .89 \\ .61 & 1.23 & .79 \\ -.14 & -.48 & .66 \end{bmatrix}$$

The column sums are [1.26 1.70 2.34] The actual Jacobian of the function at

```
; (a) Lorenz system as input to program
(define lorenz
   '((* a (- y x))
     (- (* r x) (+ y (* x z)))
     (- (* x y) (* b z))))
(define *state-variables* '(x y z))
(define *parameters* '(a r b))
(define *augmented-state-variables*
  '(x y z delxx delxy delxz delyx delyy delyz delzx delzy delzz))

; (b) System derivative generator
(lambda (a r b)
  (lambda (*state* t)
     (let ((x (list-ref *state* 0))
           (y (list-ref *state* 1))
           (z (list-ref *state* 2)))
          (list (* a (- y x))
                (- (* r x) (+ y (* x z)))
                (- (* x y) (* b z))))))

; (c) Variational system derivative generator
(lambda (a r b)
  (lambda (*augmented-state* t)
    (let ((x (list-ref *augmented-state* 0))
           ... as in (b) above ...
          (delxx (list-ref *augmented-state* 3))
           .
           .
           .
          (delzz (list-ref *augmented-state* 11)))
         (list (* a (- y x))
               (- (* r x) (+ y (* x z)))
               (- (* x y) (* b z))
               (+ (* (* a -1) delxx) (* a delxy))
               (+ (* (- r z) delxx) (+ (* -1 delxy) (* (- 0 x) delxz)))
               (+ (* y delxx) (+ (* x delxy) (* (- 0 b) delxz)))
               (+ (* (* a -1) delyx) (* a delyy))
               (+ (* (- r z) delyx) (+ (* -1 delyy) (* (- 0 x) delyz)))
               (+ (* y delyx) (+ (* x delyy) (* (- 0 b) delyz)))
               (+ (* (* a -1) delzx) (* a delzy))
               (+ (* (- r z) delzx) (+ (* -1 delzy) (* (- 0 x) delzz)))
               (+ (* y delzx) (+ (* x delzy) (* (- 0 b) delzz)))))))
```

Figure 5.14: (a) The Lorenz system as it is presented to *Perfect Moment* (b) compiled system derivative procedure used to evolve the system state (c) compiled Jacobian procedure used to evolve the variational system state

the point (13.087 23.455 24.204) is

$$\begin{bmatrix} -16 & 16 & 0 \\ 25.796 & -1 & -13.087 \\ 23.455 & -13.087 & -4 \end{bmatrix}$$

Finally, simply integrating (9 11 21) forwards for the same time period brings the state to (14.324 25.051 26.581): within (0.16 0.42 0.14)% of the point $\phi_{t=.05}(8\,10\,20)+(x_c\,y_c\,z_c)$. On a ten times longer integration, the difference — a result of the growing distance from the point where the linearization was made — grows to (0.29 0.32 0.6)%.

• • •

In the cell size revision, the m from the previous pass is used as a baseline and raised or lowered depending on tests that measure an evolved variation against the cell size. The identity matrix — the typical starting condition of a variational integration — is multiplied by a scaled version of the difference vector between the points to be connected. One assumption implicit in this choice is that moves directly towards the destination are more important than those in other directions. This approach is crude and reminiscent of steepest-descent techniques: standard, local-view-of-the-world nonlinear control. However, this heuristic is *only* used in the cell size determination and does *not* in any way constrain or predispose the path finder to steepest-descent paths. A second implicit assumption is that the distance between control objectives is a rough measure of the move that will be made at a particular pass and hence should affect the program's view of scale. This biased variational system is integrated forwards from the center of the new origin endpoint cell to its boundary, then compared to the cell size. Appropriate adjustments are made to m and the process is repeated across the parameter range.

A formal version of this revision algorithm is given below. Refer to figure 5.15 for graphical interpretations of the important quantities.

1. Subdivide the new region using the *previous* m. Compute the scaled difference vector $\vec{V} = \{\frac{\vec{x}_{dest} - \vec{x}_{orig}}{m^2}\}$. Compute the centerpoint $\vec{x}_c$ of the new, factor-of-m-smaller endpoint cell that contains the origin.

2. With $k = k_l$, integrate the augmented system $\{\dot{\vec{x}},\ \dot{\vec{\delta}}\}$ forwards from $\{\vec{x}_c,\ \vec{I}\cdot\vec{V}\}$ until the trajectory exits the cell at $\vec{x}_{boundary}$.

3. Compare the evolved variations with the cell size: compute the column sums of the variational matrix $\{\delta_{ij}\}$ at the cell boundary and divide the result, elementwise, by the vector of cell edge lengths $[h_1\ h_2\ \ldots h_n] = \frac{\vec{x}_{high-corner} - \vec{x}_{low-corner}}{m}$ and take the absolute value of each element to obtain the comparison vector

$\vec{C}^k$ (k identifies the parameter value involved.) Find the largest element c_j^k of $\vec{C}^k$. If $c_j^k > 1$, set $m = m \times c_j^k$; otherwise record c_j^k.

4. Repeat steps 2 and 3 for all $k_{low} + n\Delta k$ for $(k_{low} + n\Delta k) \in [k_{low} k_{high}]$.

5. If m remains unchanged after step 4 is complete, multiply it by the *maximum* of the values recorded in step 3.

6. Round m upwards to the next integer (or 2, whichever is greater.)

Step 6 avoids:

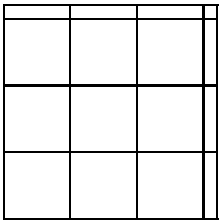


Figure 5.15 shows a schematic of a single pass of this algorithm on a 2D example. The area enclosed by the solid box is the shifted endpoint cell from the previous pass; $\vec{x}_{orig}$ and $\vec{x}_{dest}$ are either the old origin and the segment startpoint, or the segment endpoint and the old destination. Since this is not the first pass, these points need not be on the main diagonal, but the region is still centered around them because of the recentering rule. The old m was two, so the solid box is redivided into four cells to begin. The difference vector $\vec{x}_\Delta = \vec{x}_{dest} - \vec{x}_{orig}$ cannot be used, unscaled, as a variation; its worst-case length is $O(\sqrt{n \cdot m})$ cells, far larger than the normal units in variational integrations (~ 1.) The normal variation, shown at the left of figure 5.15, would be invisibly small on this scale. If the scaling were $\frac{1}{m}$, the variations would start out comparable to the cell size and always fail the (variation size $\sim$ cell size) test of step 3. For these reasons, the scaling used is $\frac{1}{m^2}$: $\vec{V} = \frac{1}{4}(\vec{I} \cdot \vec{x}_\Delta)$ here. These vectors are shown at the centerpoint $\vec{x}_c$ of the new origin endpoint cell, the initial condition for the integration of the (dashed) trajectory $\phi_t(\vec{x}_c)$. The *evolved* variation is shown at the intersection of $\phi_t(\vec{x}_c)$ with the cell boundary; note how much the magnitude and direction of the constituent vectors have changed. They are clearly larger than the cell edges, so m is adjusted up and then rounded to the next integer. Note that m can be reduced as far as two, and no further. When $m = 1$, the next pass will only generate two trajectories per portrait ($\phi_t(\mathbf{o})$ and $\phi_{-t}(\mathbf{d})$) and counterintuitive moves will be impossible.

Many alternatives and several areas for improvement are apparent in these steps. For example, m is scalar, so the cell aspect ratio is constant. If m were an n-vector, the cell *shape* could be adapted to the dynamics as well, eliminating the situation where one edge of the cell is always driving the expansion or shrinkage.

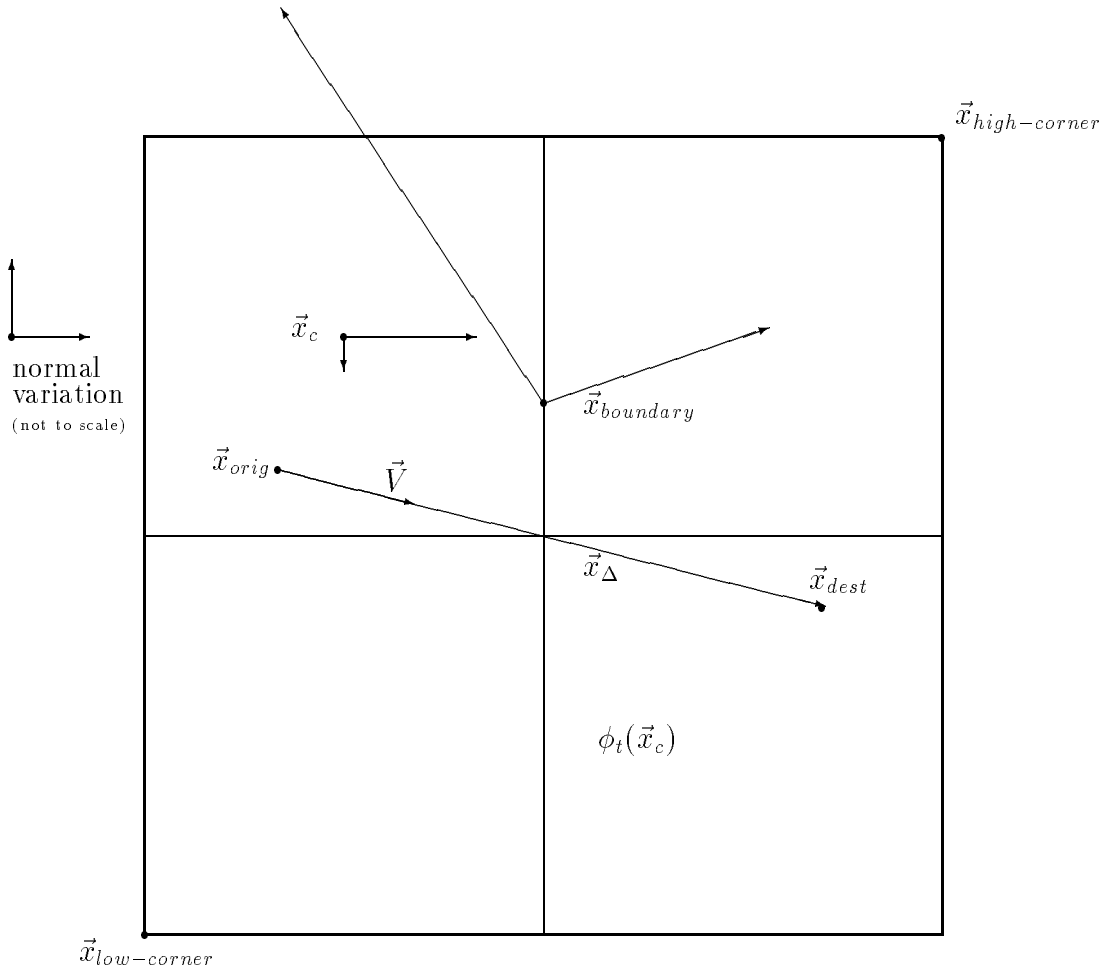


Figure 5.15: Refining the grid: if the variation $\vec{V}$, integrated forwards from the centerpoint $\vec{x}_c$ of the origin cell to its boundary, grows to be larger than the cell itself, the cell size is reduced. The initial variation is a scaled version of the difference vector $\vec{x}_\Delta$ between the origin and the destination

This modification would simply be a matter of bookkeeping. More interesting, subtle, and far-reaching changes could be made in the variational integration scheme. The difference vector could be scaled by $\frac{1}{n \cdot m}$ instead, for example. An different vector altogether, like the diagonal of the endpoint cell, could be used as well, perhaps with a different scaling factor. The initial condition could be the origin instead of the cell centerpoint, but that could cause the types of problems depicted in figure 5.12. The key design choice in the rule that uses these results — the factor of one in step 3 above — is another area for improvement. Currently, flows that diverge faster than one cell size per cell are considered turbulent enough to warrant finer discretization; this threshold effectively causes the cell size to be reduced until trajectories look roughly linear therein. Finally, the parameter range is checked only at the original Δk-spaced values, so highly-turbulent behavior that is hidden between steps will be missed and the cell size will be inadequate when the mapper zeroes in on this behavior — which it is designed to do. Using the inter-portrait parameter spacing computed by the mapper instead of the constant Δk would partially solve this problem. Many other interpretations, rules, and modifications are certainly possible as well.

The path finder's final actions are to reduce the time step by the ratio of the new m to the old m, as alluded to in section 4.2.1, and to invoke the mapper with the new origin, destination, region and intervals.

The revision rules in this section serve as a reasonably-successful first cut, from an engineering point of view, at adapting the resolution to turbulence, sensitivity, and to the changing control scale via automatic manipulation of the region and cell size.

5.3.3 If the Search Fails

The last layer of region and cell size revision rules concern actions that are taken if the path finder fails to find an adequate segment between the endpoint cells. This failure can occur for a variety of reasons. If the endpoints of the shortest segment on one pass are downstream of the destination, for instance, the metric returned by `minimum-sum-distance-tiebreaker` for all segments "found" will be longer than the original distance to be spanned. *Perfect Moment* divides failures into two classes, depending upon how many trajectories enter the sink cell, and treats each class differently.

If the search fails and all trajectories go to the sink cell, the region size is *increased by* R_0. In this event, m is left unchanged — *not* multiplied by R_0 and rounded upwards to maintain accuracy. This rule is meant to catch dynamics that were missed by the mapper because the region is too small. The "all trajectories exit the grid" test is basically a rough way to determine whether or not the region covers all the attractors that it touches; it works in conjunction with the method

described on page 58. One could obviously define a different (constant) fraction, or one that adapts to an important metric like the amount of turbulence. A yet-more intelligent approach might be to check the destinations of the trajectories outside the ring of cells level, in a cell-distance-from-the-center sense, with the objectives; this scheme would automatically aid in the solution of the problem mentioned in the first paragraph of this section. Ultimately, broader, AI-based improvements on the techniques for reasoning from partial data that are suggested on page 56, perhaps involving vision techniques and pattern recognition, would be the best solution.

The use of R_0 as the region scaling factor gives the user a hook into this process. This feature requires an R value to be given even when it is to be bypassed with a specific region specification; it also gives the user rope to hang him or herself, as serious problems can result from bad choices. For example, the expansion and discretization rules can cancel each other out if R_0 is large and the dynamics are laminar. In this case, the region might be expanded by by 4, then divided into three cells on a side, making later search regions *larger* instead of smaller.

If the search fails and all trajectories do *not* go to the sink cell, the number of intervals is multiplied by 1.5 and rounded upwards to the next integer. This rule originally doubled m in the fashion of standard binary-search algorithms everywhere, but the factor of two often, in practice, turned out to be excessive. More importantly, if the source of the *Perfect Moment*'s failure is that interesting state-space behavior is evading exploration because its initial condition is near a cell boundary, multiplying m by an integer will not help.

Since the case where the attractors' areas exceed the grid is caught by the first rule in this section, the second rule covers the case where the behavior has been sampled, but not on a fine enough grain. A better way to do this might be to lower the factor of one in step 3 of the algorithm on page 83; this would couple the actual dynamics, rather than a semi-arbitrary integer, into the m-modification.

After one or the other of these adjustments is made, the path finder calls the mapper with the new values and repeats the search. This transcript demonstrates the second rule in action:

```
(beginning mapping process)
(mapping region from
  #(0 -24.25 41.5 7.5) to
  #(0 24.75 -28.5 -2.5) with
  4 cells on a side)

(map: parameter = 20)

< ... >
```

```
(no path exists between o and d on param = 40 map
  - hit space to proceed)

(no path segment was found between #(-2.2 10 3) and #(2.7 3 2)
  for the parameters
  (klow 20 khigh 40 kstep10
   R 10 m 4 points 200))

(revising intervals!!)
(stop me if you want to change the parameter range or spacing)

(beginning mapping process)
(mapping region from  #(0 -24.25 41.5 7.5) to #(0 24.75 -28.5 -2.5)
  with 6. cells on a side)

< ... >

(segment found:
(#(6.29 -1.45 2.02)
 #(-5.64 10.01 2.56)
 25
 #(-24.25 41.5 7.5)
 #(24.75 -28.5 -2.5)
 #[compound-procedure 45]
 #[compound-procedure 46]))
```

The R and m on the previous pass were 10 and 4, respectively; the former was large enough to cause at least some of the trajectories to find attractors inside the search region, so the *second* search-failure rule fired and adjusted the m to 6, at which point *Perfect Moment* was able to find a path.

These two rules cost a lot in run time, but, as the transcript above shows, they sometimes solve the problem. If, however, *no* magnification or resolution would help, the various rules will blindly continue revising the region outwards to infinity and the cell size downwards to zero, and the user should intervene. There is no obvious way to cause *Perfect Moment* to recognize this ahead of time and yell for help.

5.4 Tolerance and Termination

The programs described in this section — grouped in the dashed box inside block **C** in figure 2.4 — ascertain whether or not a segmented path meets a specified

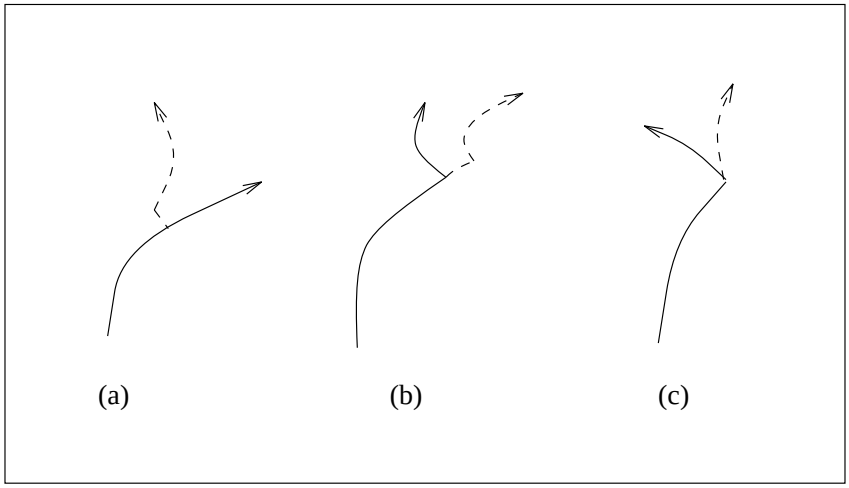


Figure 5.16: Error types at segment junctions: (a) additive (b) timing (c) parameter. Solid lines are the desired paths and dashed lines reflect the respective errors

control tolerance. These computations depend on the specified tolerance, on the properties of any existing partial path, and on the range of the linearized controller that the program constructs for each segment junction[8]. The techniques in this section do *not* address errors made at points away from the segment junctions; the implicit assumption here is that, to first order, errors are most likely to occur at the junctions. Tracking techniques — that extend local-linear control across the entire length of each segment — are discussed in chapter 6.

Without the linearized controller, errors made at the junctions would simply grow or shrink along the entire path, according to the nonlinear expansion or contraction in force at each point along its length. Errors at the origin end are potentially worse than errors near the destination because of their longer exposure to nonlinear amplification. If the path contains no strange attractor segments or unstable trajectories, the state-space "tube" around the trajectory is always contracting and errors are less of a problem. If the local-linear controller has a particularly wide range at one junction because of the value of the system's Jacobian, errors at that junction have less leverage.

Junction errors take three forms: state/additive, switch timing or parameter magnitude. The effects of each are shown in figure 5.16. A gap in the path — the concern addressed by the tolerance checker — is of the form (a). Timing and parameter magnitude errors are covered in chapter 6.

The following algorithm checks a single-segment path S^0 against the tolerance $\vec{T}$:

[8]The techniques involved in the design of such a controller are demonstrated in the worked-out example of section 3.3.1.

1. Check whether the segment endpoint falls within the tolerance region $\vec{T}$ around the control destination. If not, the gap between the destination and $\vec{S}^0_{final}$ is causing a tolerance problem and requires filling in.

   If step 1 fails and the segment comes from a trajectory with a positive Lyapunov exponent, the gaps at *both* ends are automatically remanded for further path finding.

   If step 1 succeeds, the gap at the *other* end of the segment is checked:

2. Compute the range $\vec{L}$ of the linear controller at $\vec{S}^0_{init}$.

3. Integrate the tolerance region $\vec{T}$ backwards along the segment (to $\vec{T}'$).

4. Intersect the regions defined by $\vec{T}'$ and $\vec{L}$.

5. If the origin lies in this region, the segment passes the check.

Perfect Moment's `lyapunov` function computes the largest positive Lyapunov exponent λ_1, operating on a variational derivative, an integration period, and a multiplier. The mathematics of this method are discussed in section 3.2.1; it can be extended to find the other $(n-1)$ λs[60], but *Perfect Moment* only uses λ_1.

See figure 5.17 for a schematic of this process. The tolerance, as currently specified to *Perfect Moment*, is an n-vector, so the target region is a rectangular n-parallelepiped — marked T on the figure. That region is evolved backwards to T', the larger rectangle around the trajectory's starting point. The ellipse represents the controller's domain; the origin must lie in the shaded intersection region to pass the tolerance check.

If the path contains more than one segment, this process is iterated backwards from the destination across all segments in the path, using the shaded region as the new T at each step. Any inter-segment gap that is not entirely contained by the corresponding shaded intersection region causes a tolerance failure.

The key assumptions here are that the linear controller (1) is accurate (2) achieves control much more quickly than the transit time of the trajectory through its region of control and (3) reacts to reprogramming of its parameters more quickly than the time taken to traverse the smallest segment. Much of chapter 6 is based on the various failure modes that invalidate these assumptions.

In nonautonomous systems, tolerance checking is much simpler. Only origin trajectories are generated and used; the path grows forwards from the control origin and reaches closer and closer to the destination as the run progresses. At any point, there is only a single gap; if that gap is smaller than the tolerance region at the destination, the path passes.

Finally, if the tolerance checker encounters a segment whose length is below the machine epsilon or below the effective resolution of any physical I/O devices, it

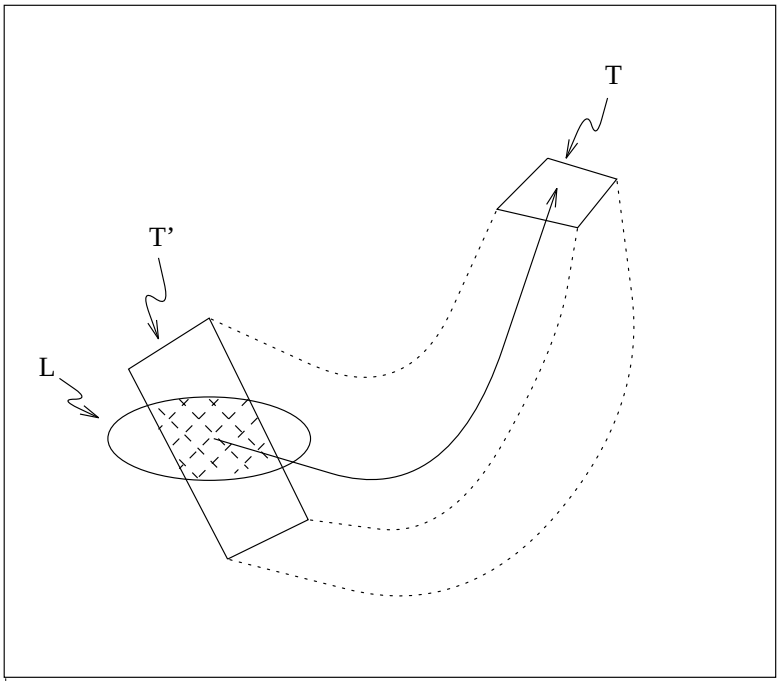


Figure 5.17: Computing whether a single-segment path meets a tolerance. The solid arrow is the path segment $\vec{S}$; T is the tolerance region at its endpoint. T' is the same region, integrated backwards to the starting point $\vec{S}_{init}$. L is the domain of the junction controller at $\vec{S}_{init}$. If $\vec{S}_{final}$ lies in T and $\vec{S}_{init}$ lies in the intersection of T' and L, the segment passes the check.

signals an error, at which point the user should restart the program with different parameters. This type of error can occur if, for example, the tolerance is fundamentally too small, or if *Perfect Moment* has made a bad choice and is struggling to make up for it.

Chapter 6

On-Line Control

Real-time control — the right-hand box in figure 2.3 — has two stages. The system must first be routed along the segmented path (section 6.1) and then stabilized upon its arrival at the destination (section 6.2.)

6.1 Navigating Along the Path

The system state can be caused to evolve along a trajectory consisting of a series of path segments $\{S^0, \ldots, S^l\}$ via the following set of control actions. Because of the recursive, longest-first nature of the path-finding algorithm, the segments are not followed in the order in which they are found, so the list must first be sorted into the proper order $\{\hat{S}^0, \ldots, \hat{S}^l\}$. Beginning at the control origin, the parameter is set to $\hat{k}_0$ to initiate the first segment $\hat{S}^0$, and the local-linear junction controller is programmed with the values at $\hat{S}^1$'s starting point. The state is then monitored and the controller is turned on when $\vec{x}$ enters the control range around the junction between $\hat{S}^0$ and $\hat{S}^1$. The parameter is then changed to $\hat{k}_1$ and the combined closed-loop dynamics of the system and the junction controller pull the trajectory towards $\hat{S}^1$'s starting point. When the trajectory arrives, the junction controller is turned off and the system proceeds along $\hat{S}^1$. As soon as the trajectory exits the control range[1], the junction controller is reprogrammed with the values at the next junction. This procedure is repeated through all segments in the path. After the final switch, if necessary, the system is stabilized at the destination by the same local controller.

Of course, this scenario can only be borne out in an ideal world.

In addition to the timing, state and parameter magnitude errors depicted on figure 5.16, the model can be wrong or the mechanical integrator can introduce

[1] and not before, in case errors require it to be turned on again

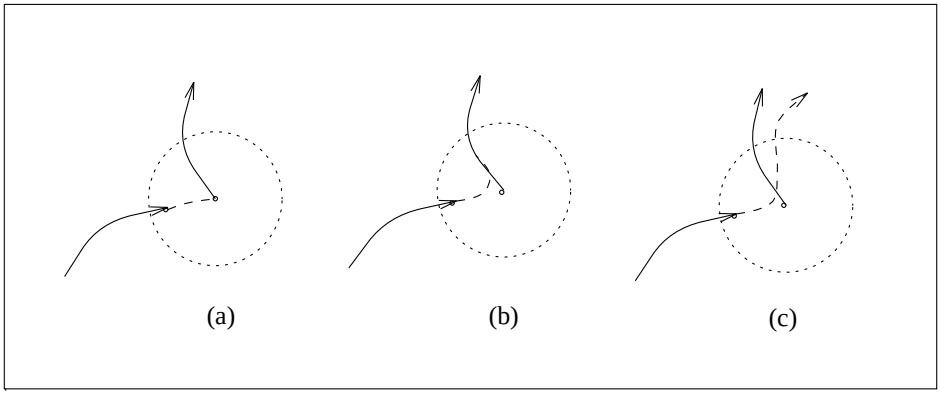


Figure 6.1: Control at the segment junctions: (a) ideal case (b) successful control (c) unsuccessful control. The dotted circle is the junction controller's range; the solid trajectories are the desired path segments and the dashed paths are the actual trajectories.

spurious dynamics. Errors are all but unavoidable; the optical encoder used to measure the position of the pendulum described in chapter 7, for example, has an angular resolution of 0.7 degree. Quantization error is inescapable because of the discrete nature of computer arithmetic in a continuous world. Actuators have finite response times; gusts of air hit pendulums, dust contaminates ball bearings, circuits pick up local radio stations, etc. The linear junction controller can only compensate for state and parameter errors if they are within its range — and then only if they occur at the junctions, where it is turned on. It is useless once it has been reprogrammed for a different junction. Incidentally, the reprogramming speed puts a lower bound on the segment length: if the new coefficient values do not reach the controller's registers and DACs before the trajectory reaches the control region around the next junction, the scheme will fail. If the dynamics are inaccurate because of modeling- or integrator-induced error, the design assumptions on which the junction linearization is based are invalid and control can be lost. This inaccuracy need not be large and can be highly sensitive to position and parameter because of the nonlinear amplification.

If the linear junction controller works perfectly, the trajectory follows a path like the one shown in part (a) of figure 6.1. The path in part (b), perhaps the result of a small error that has moved a system pole a bit, is still adequate, but the one in part (c) is not, perhaps because that same pole has moved too far.

State noise is the simplest problem to analyze. It is additive, resembling part (a) of figure 5.16, and need only be measured against the range of the junction controller. The very use of strange attractor segments, somewhat paradoxically, introduces a flavor of robustness: if the last segment in the path is part of a chaotic attractor, shadowing implies that it will eventually pass near that point again. Though it does "solve" the problem of state noise-induced loss of control, such a path would be unpredictably circuitous and probably far from optimal. If

the error is made upstream from a critical nonchaotic path segment, the correction task simply falls to the linear controller. Though no *exact* calculations of control failure at a junction due to additive state noise are possible unless the precise form of the noise vector is known, a *stochastic* measure of how often the controller succeeds can be obtained by dividing the area of its range at that junction by the RMS noise value[2].

Timing problems like those in part (b) of figure 5.16 typically arise from the combined reaction times of the computer, the actuator and the sensor. These can be approximated as state noise, given an estimate of the delay Δt: one simply computes the difference vector $\Delta t \cdot \vec{F}$ and treats the results as in the previous paragraph. If the sum of the Δts and the time required to program the junction controller are both much ($> 10\times$) smaller than the natural frequencies of the target system, timing errors can be disregarded. This requirement is one compelling reason to choose mechanical pendulums, not communications circuits, as test systems.

Parameter magnitude errors can cause serious problems, as discussed in previous chapters: nonexistent attractors, suddenly-positive λs, etc. Even small changes that do not cause such global, catastrophic effects can still send the system in the wrong direction; recall part (c) of figure 5.16. As with timing errors, if the error is known, its effects can be estimated and treated as state noise: $\Delta k \cdot \frac{\partial F}{\partial k}$.

A first-order upper bound on the achievable path length is

$$\frac{T}{error \times e^{\lambda_1 t_{path}}} \tag{6.1}$$

where T is the control tolerance, λ_1 is the largest positive Lyapunov exponent of any path segment, and t_{path} is the time to traverse the entire path. *error* can arise from several sources, summarized below:

1. environment
   (a) noise
   (b) system degradation
2. computer
   (a) machine epsilon
   (b) reaction time (interrupt servicing)
   (c) numerical integrator error
3. controller I/O

[2]assuming white noise.

(a) sensor resolution and delay

(b) actuator resolution and delay

4. model inaccuracy

The achievable path length may actually be even worse than equation (6.1) if the error causes λ_1 to change — for example, if a parameter magnitude error is made near a bifurcation boundary or in a range where the attractor is rapidly getting "more chaotic" (in the sense that its λs are rising quickly.) Avoidance of boundary regions could actually be programmed into the optimization functions, partially solving this problem. A worse problem is possible lack of correspondence between state-space features of the model and of the system; unlike the previous case, there is no easy solution to this problem.

Modeling errors have complicated, coupled, global nonlinear effects, much like those of numerical integrators; their effects are perhaps the hardest and most serious set of problems encountered in this research. Coefficient differences — "parametric uncertainties" — are effectively identical to the very parameter changes that *Perfect Moment* uses for control leverage. System degradation over time has the same effects. A recently-developed scheme[70] actually uses this preternatural sensitivity to construct better estimates of the parameters themselves. *Structural* differences, like the presence of an unmodeled saturation, friction, mechanical interference between two parts, etc, introduce worse problems, as they change the form of the equation and not just the coefficient values. The very nonlinear/chaotic sensitivity that is used here for leverage can, paradoxically, backfire and make a simple problem insoluble (i.e., a bifurcation that obliterates the chaotic attractor that is critical to the controller's performance.) Modeling accuracy is a serious problem; solving it requires accurate measurement technology, intelligent interpretation and use of the measured data, and a deep understanding of the underlying physics. This problem is fundamental and far-reaching — the sole focus of entire research groups in applied math departments — and I have no sweeping, original solutions to offer.

One *step* towards a solution would be to gather state-space portraits directly from the physical system. This would eliminate numerical integrator error (timestep issues and the necessity to corroborate results with several different integrators.) This tack is related to [50], a program that automatically produces and analyzes a parameter-space graph for chaotic and periodic behavior zones in an electronic circuit. The inherent problem here is sensor and actuator resolution, coupled with the ubiquitous nonlinear amplification. The very same amplification, surprisingly, can be highly beneficial: it can be used to enhance experimental parameter estimates[43], using nonlinear dynamics knowledge to combine separate measurements in such a way as to cancel out their error.

Another obvious and highly-useful modification would be to make this control

scheme *robust*: less sensitive to small state or parameter perturbations. Accomplishing this while at the same time preserving *Perfect Moment*'s original goals would require balancing some subtle tradeoffs, as the tension between robustness and sensitivity is fundamental and inescapable. One solution would be to *plan* on a global scale and *track* on a local scale, extending the use of the linear controller across the entire length of the path, rather than confining it to the junctions. This type of intelligent, computer-controlled tracking would move the boundary in figure 2.3 to the right. It would allow the controller to respond to errors made *anywhere* along the path and give it more time and space to recover from any single error. The critical obstacles are computer speed and the response time of the linear controller; the latter would have to respond continuously and instantly to reprogramming instructions and the former would have to compute those instructions accurately and on the fly.

Finally, one could couple some measure of the system's performance back into the control recipe — a flavor of *adaptive* control — making information flow both ways across the boundary in figure 2.3. Coupled with the tracking scheme described in the previous paragraph, this type of intelligent, computer-controlled parameter estimation and model adaptation would allow *Perfect Moment* to adapt to inaccurate models or changing systems.

6.2 Stabilizing the System at the Destination

Standard linear control can be used to stabilize the system at the destination after the reference trajectory has been traversed. A detailed example of this process is given in section 3.3.1. Alternatively, if the destination lies on or near a strange attractor, one can find and stabilize a nearby unstable periodic orbit. This section describes and illustrates that approach. Of course, these techniques do not make all points controllable, even in a highly-restricted neighborhood; see the caveats in chapter 8.

Since any chaotic trajectory covers its attractor densely, it will eventually have a close encounter with every one of the embedded unstable periodic orbits. When this happens, the trajectory retraces its path for some number of cycles: more or less, depending on how close it was to the orbit in the first place. This denseness is exploited by *Perfect Moment*'s `gunaratne`[3] function. Points on a Poincaré section that return to their own small neighborhoods after m piercings are assumed to be very close to m-cycles. Averages of tight bunches of such points are taken to be good approximations to unstable fixed points. The size of the neighborhood (ϵ) is fixed at two or three orders of magnitude smaller than the signal and the separation between bunches is defined as 2ϵ. The averaging refines the estimate

[3]the name of the first author of the paper [36] that suggests this technique

and also serves to reduce noise by a square root. `gunaratne` takes a Poincaré section and an ϵ-order, and returns a vector of lists of candidate points, sorted according to the number of cycles before their return. The ϵ-order choice is a tradeoff. For a given data set, use of a large ϵ-order causes very few points to satisfy the criteria, but those that do will be closer to the UPO. If the order is small, the bunches contain more points, but are more diffuse. The latter is used here because it makes the bunches easier to recognize.

Continuing the Lorenz example begun in section 5.3.2, the seventh element of the vector `(gunaratne psection order)` — where `psection` is 3000 piercings of the $y = 20$ plane — contains the following bunch of four points, among others:

```
 ...
(102 (-21.160 60.151))
(105 (-21.151 59.941))
(109 (-21.058 59.767))
(946 (-21.181 60.163))
 ...
```

These four points are candidates for a seven-cycle. The first entry in each element is the number of the point that *ends* the periodic orbit; thus, these orbits started on, respectively, the $95^{th}, 98^{th}, 102^{nd}$, and 939^{th} piercings of the plane. Note that the 95^{th} point must have been closer to the UPO than the others were, as it returned twice. These four points are then averaged to find a better approximation to the true UPO — (-21.138 60.005). This point returns to the neighborhood *four* times, then escapes. Better estimates and longer periods spent on the UPO simply require more points. [36] used 32000 points and noted 50-100 points per bunch.

Once found, an unstable periodic orbit can be stabilized using a control scheme developed by a group at the University of Maryland[59, 69], wherein the system's dependence upon the parameter is linearized about the fixed point on the $n-1$-dimensional surface of section through the orbit. This technique works where the linearization is a good approximation: in the $n-1$-dimensional "control parallelogram" around the point, whose size is determined by the control parameter's range, its effects on the orbit and the orbit's unperturbed stability properties. A small change in k causes a $k = k_0$ periodic orbit to return, after m cycles, not to its original coordinates $\vec{\mathcal{P}}_0$, but to some nearby point $\vec{\mathcal{P}}$. The vector $\vec{g}_k$ measures this effect:

$$\vec{g}_k \equiv \frac{\partial \vec{\mathcal{P}}_0}{\partial k}|_{k_0} \approx \frac{1}{k-k_0}(\vec{\mathcal{P}} - \vec{\mathcal{P}}_0) \tag{6.2}$$

The stability properties are determined by integrating the variational system around the orbit to obtain a Jacobian. f^j is the j^{th} component of the system equations (3.1) and the δ_{ik} are variations around $\vec{\mathcal{P}}_0$. In a three-dimensional system, the single unstable eigenvector $\hat{e}_u$ and eigenvalue λ_u of the Jacobian matrix, together

with the admissible variation of the parameter k^* around k_0 and the vector $\vec{g}_k$, determine P_{c_k}, the size of the control parallelogram, according to

$$P_{c_k} = k^* |(1 - \lambda_u^{-1}) \vec{g}_k \cdot \hat{e}_u| \tag{6.3}$$

Details about the derivation of these formulae are given in [59]; an example is worked out in section 5.3.2. This method has been successfully demonstrated on a magnetoelastic ribbon[28].

Trajectory denseness and UPO embedding make this a self-contained control scheme. Since the control parallelogram surrounds a point that is in a chaotic attractor, all trajectories will eventually enter the controller's domain, be driven to the orbit, and, in the absence of noise, remain there indefinitely. The denseness of these orbits makes this technique practical if a chaotic attractor overlapping the target state exists; however, target acquisition is a problem. The delay before any particular trajectory wanders into the parallelogram is unpredictable, although it does depend stochastically on the ratio of the areas of the parallelogram and of the entire attractor. Target acquisition — active intervention to hasten this event — is still somewhat of an open problem, though it has been addressed recently[15, 16, 69] and is discussed at length in the next chapter of this thesis.

Chapter 7

Examples

All of the examples in this section are simulated. However, physical applications are the ultimate target of this work, so the models and controls in sections 7.2 and 7.3 reflect the physical parameters of a driven-vertex pendulum and a phase-locked loop circuit that have been constructed as test cases for these techniques. Modeling and experimental error, discussed in section 7.4, have caused serious problems in the extension of the simulated results to actual physical systems.

Most of the figures in this chapter are actual copies of the graphics output of the program — with most of the grid lines and trajectories edited out. The purpose of this pruning was to clarify the exposition; the values for m, R, etc, were chosen here to do real design, not for cosmetic purposes, so the real plots are fairly complex.

7.1 The Lorenz System

The Lorenz equations[52] are:

$$F(\vec{x}, a, r, b) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} a(y-x) \\ rx - y - xz \\ xy - bz \end{bmatrix} \tag{7.1}$$

This well-known system models convection in a heated fluid. It is an ODE truncation of a partial differential equation model — the Navier-Stokes equations. The state variable x is proportional to convective intensity, y is proportional to temperature variation, and z expresses the amount of deviation from linearity in the vertical convection profile. The coefficients a and r are physical parameters of the fluid — the Prandtl and Rayleigh numbers — and b is an aspect ratio. Edward Lorenz discovered exponential trajectory separation on a chaotic attractor

Figure 7.1: Lorenz attractor for a = 16, r = 45, and b = 4

Figure 7.2: $x - z$ section at y=-15 of Lorenz attractor for a = 16, r = 45 and b = 4

in a series of simulations of these equations on an analog computer. He found that if an initial condition was entered to a different number of decimal places, the trajectories diverged exponentially in the time domain, but covered the same attractor. Practically every text or review of chaos has a picture of the Lorenz attractor, most of them at or near the parameter values $a = 16$, $r = 45$ and $b = 4$, as in figure 7.1. This figure shows an $x-z$ projection of a single trajectory covering a three-dimensional object, so the apparent crossings do not represent uniqueness violations. The parameter values and the initial state are shown at the bottom of the figure. The latter is immaterial unless it falls outside the basin boundary, which is far larger than the area enclosed by the solid box. Within the basin, the position of the initial condition only affects the time and path to reach the attractor and the order in which the whorls are traced out by the trajectory, not the geometric structure of the attractor. The values in the upper right and lower left corners of the figure are normalized axis coordinates. An $x - z$ section of the same attractor, through the $y = -15$ plane, is shown in figure 7.2. Note the fractal nature of both the section and the attractor itself.

The Hausdorff dimension of the latter is approximately 2.1. *Perfect Moment*'s `fractal-dimension` procedure uses the grid to obtain an extremely quick and dirty implementation of the definition (3.5), using grid cells as the "balls of radius ϵ" in equation 3.5. Figure 7.2, incidentally, represents a much longer integration than figure 7.1 and both are much longer than *Perfect Moment*'s standard ten orbits. In the portraits that are automatically constructed by the program, trajectories converge on the attractor from all angles, so it appears densely-covered despite the finite (ten-orbit) length of each individual trajectories; the longer integration was used in figures 7.1 and 7.2 to duplicate this effect and make the structure of the objects visible.

One of the unstable periodic orbits (UPOs) embedded in the $r = 50$ attractor — a seven-cycle — passes through the point $\vec{\mathcal{P}}^7$ = `#(-24.673 -19.983 68.207)`. This UPO is plotted in $x - z$ projection and $y = -20$ section in figure 7.3(a) and (b), respectively. ¡It was found by the `gunaratne` procedure of section 6.2, using a 3000-point Poincaré section and an ϵ-order of two.

This UPO can be stabilized using the local-linear method of [59], as discussed

Figure 7.3: Trajectory and $x - z$ section ($y = -20$) of unstable seven-cycle embedded in $r = 50$ Lorenz attractor

Figure 7.4: Time to acquire and achieve control on an unstable periodic orbit

Figure 7.5: Two stable fixed points

in section 3.4. The eigenvalues of the variational system, integrated once around the seven-cycle from $\vec{\mathcal{P}}^7$ are $\lambda_u = 0.734$ and $\lambda_s = -502.077$, with the associated eigenvectors $\hat{e}_u = 0.834\hat{x} + 0.552\hat{z}$ and $\hat{e}_s = 0.685\hat{x} + 0.728\hat{z}$. A 1/2% change in r causes the point $\vec{\mathcal{P}}^7$ to return not to itself, but to $\vec{\mathcal{P}}^{7'}$ = `#(-23.929 -19.983 66.498)`, so

$$\vec{g}_r \equiv \frac{\partial \vec{\mathcal{P}}^7}{\partial r}|_{r=50} \approx \frac{1}{\Delta r}(\vec{\mathcal{P}}^{7'} - \vec{\mathcal{P}}^7) = 70.8\hat{x} - 170.9\hat{z}$$

The allowed range of variation of r, for a 1% control tolerance, is ± 0.0075. Using these values in equation (6.3) gives $P_{c_r} = 0.096$. A trajectory emanating from the point `#(8 29 64)` will enter the control parallelogram after traveling 25223 normalized distance units around the attractor in 104 normalized time units. See figure 7.4.

The "eyes" of the Lorenz attractor contain two unstable fixed points. As r is lowered from the values in the previous figures, a bifurcation at $r = 24$ causes these points to become stable; see figure 7.5, constructed at $r = 20$. Note the different basins of attraction. These fixed points — whose coordinates are `#(8.72 8.72 19.00)` and `#(-8.72 -8.72 19.00)`[1] on these axes — move about the space as r is varied, but always remain in the bottom third of the region.

Stable periodic orbits exist at much higher rs. For example, orbits that wind once, three times or four times around each lobe exist at $r = 350$, 126.52, and 132.5, respectively, all with $a = 10$ and $b = 8/3$. With these a and b values, the one-cycle actually persists for all $r > 312$[75]. More details about the structure and properties of Lorenz attractors may be found in [52], [75], or [81].

7.1.1 Targeting and Stabilizing an Unstable Periodic Orbit

Perfect Moment was given the task of navigating between the two points marked by crosses in figure 7.6, starting at the rightmost (**A**) and ending at the leftmost (**B**.) On the axes of the figure, which are identical to those of all previous figures in this chapter, the coordinates `#(x y z)` of these points are `#(8 29 64)` and `#(-24.5 -20 68)`. The parameter r is used as the control parameter and a and b are fixed. No serious claims are made or implied here about whether changing

[1]The symmetry is not coincidence; see [81] for details.

Figure 7.6: Origin and destination points

this parameter — a fluid's Rayleigh number — is practical or even possible; this example is purely a mathematical one[2]. Realistically speaking, the *entire example* bears little resemblance to nature — it is a particular ODE truncation of a particular PDE model of a complex and difficult-to-observe physical system.

The destination **B** is near the unstable periodic orbit discussed earlier in this section. More precisely, the *latter* is near the *former* — the `gunaratne` function was specifically directed to explore the small region surrounding **B**. As long as the destination falls on or near a chaotic attractor, the algorithm should always be able to find a UPO within any given ϵ. The non-zero fractal dimension of the attractor and the denseness of the trajectories on and UPOs in that attractor make such a coincidence much more likely than the discovery of a parameter value that moves one of the fixed points ($D_h = 0$) to the same proximity.

Since an unstable periodic orbit exists near the destination, the local stabilization method of [59] can be invoked once a trajectory attains the control parallelogram. *Perfect Moment*'s task here is to hasten target acquisition and improve on the 25223 distance unit trajectory of figure 7.4.

The inputs to the program are:

- equations (7.1)
- origin and destination: `#(8 29 64)` and `#(-24.5 -20 68)`
- search region `#(-45 -205 -5)` to `#(55 195 95)`
- intervals $m_0 = 5$
- parameter range: $[r_l \; r_h] = [10 \; 60]$ and step: $\Delta r = 5$
- tolerance $T = 0.096$
- optimization criteria:
  - `shortest-path-between-two-cells`
  - `minimum-sum-distance-tiebreaker`
- iteration depth $D_i = 3$

[2]Lorenz himself explored the parameter space outside the range ($r \approx 1$) within which the equations are considered to be an accurate physical model.

These values are actually a second attempt. They were adjusted after a highly unsuccessful first pass that used the default values $R_0 = 2$ and $m_0 = 5$, wherein the search failed repeatedly. The graphics output showed a series of pictures that were clearly (at least to a human; see page 56) small sections of larger attractors.

The underlying problem is that the z-separation between **A** and **B** is too small. The aspect ratio of the bounding box of the two points is very different from that of the box that bounds the attractors that exist for $10 \leq r \leq 60$. This problem and its solution were discussed in section 5.3.1: the user overrides the fixed R_0 and explicitly specifies the corners of a region that contains the attractor for all allowed r values. To determine the shape of that region, the user attempts a run with ridiculously-high R, looks at the cells touched by the attractor, and computes their bounding box. The overrange R_0 would have to be 35.5 for the region to enclose the attractor of figure 7.1; in that case, m would have to be 58 to attain the same x axis exploration accuracy as that given by the numbers in the bulleted list above. The performance gain from this hand crafting is significant: $O(5^3)$ versus $O(58^3)$ in run time, or approximately 1560 times faster for the same accuracy. This region specification duplicates the axes on the previous figures in this section (again, the latter actually duplicates the former.) The region is a $100 \times 400 \times 100$ rectangular parallelepiped centered at `#(5 -5 45)`, so all cells have an aspect ratio of 1:4:1.

The other choices are far less involved. $m_0 = 5$, for instance, was used simply to test the success of the default. The user happened to know that the range [10 60] holds several different kinds of interesting dynamics, and she chose that because wants to make this example interesting so people will read her thesis. T is the size of the control parallelogram around $\vec{\mathcal{P}}^7$ that was computed for a 1% control tolerance. The Δr and the D_i are fairly arbitrary; their combined resolution requires the putative "actuator" to be accurate to 0.1 units. The optimization criteria were discussed at length in chapter 5.

On the first pass, the region was divided into 125 cells, each `#(20 80 20)` in size[3]. 25 cells would be visible on a view of any of the three faces of the region. The adaptive integrator settled on a .0005 time unit step. This choice is governed by the trajectory speed at the tips of the attractor's lobes; note the point spacing on the various figures in this section. All trajectories were 1000 steps long, as discussed in section 4.2.1.

The mapper was then invoked with these parameters. It proceeded to construct portraits at $r = 10, 15 \ldots 60$, analyze the dynamics, and reduce the Δr as required by the rules on page 57.

On the first three portraits, all trajectories were classified as relaxing to one or the other of the two stable low-r fixed points. As mentioned before, these points

[3]If the overrange machinery had not been bypassed and the default values had been used, the cells would have been `#(13 19.6 1.6)` on a side: an aspect ratio of `#(8.125:12.25:1)`.

Figure 7.7: Two stable fixed points: movement with changing r value: (a) $r = 10$ (b) $r = 15$ (c) $r = 20$

Figure 7.8: A "fixed point" for a = 16, r = 25 and b = 4: (a) ten orbits (b) 50 orbits

move (upwards on the axes in the figures) as the parameter is raised; at the same time, the damping of the oscillation around the points shrinks. See figure 7.7. Identical initial conditions were used for the two trajectories in each of parts (a), (b) and (c) of the figure; note that a basin boundary has moved across the lower of the two points between $r = 10$ and $r = 15$. *Perfect Moment* does not respond to this by increasing the parameter range resolution because none of the cells involved contains a control objective.

At $r = 25$, integrator accuracy caused the program to err in its analysis of the dynamics — numerical damping had smothered a chaotic attractor into two fixed points. Moreover, the mapper almost lost track of these numerically-induced fixed points. Note that the trajectory on figure 7.8 takes one turn around the *opposite* basin on its way to what with a longer trajectory (part (b) of the figure) would be recognized as a fixed point. This detour, coupled with the large cell size, short trajectory length and low damping of the oscillation, prevents the trajectory from settling to the fixed cell in the allotted ten orbits, so it was classified as `unknown`. However, trajectories in adjacent cells — other parts of the same basin — *do* relax far enough to cause the cells to be classified as fixed, so the mapper did not signal a bifurcation. See page 58 for discussion of this rule. A smaller time step, smaller cell size and longer trajectory length would let the classification algorithm explicitly determine that this particular trajectory is chaotic, but at the cost of significant, distributed performance loss across the entire run. These tradeoffs are discussed in chapter 4.

Figure 7.9 shows a trajectory for $r = 30$, just above the point where, at least to the program's perception, the fixed points of figure 7.7 and 7.8 mutated into a chaotic attractor. The trajectory performs several circuits of the dark oval around the newly-unstable right-hand fixed point, almost succumbing to the fixed point's pull before it spirals out and goes to the other lobe. The same dark oval exists on figure 7.8, but there the state-space contraction was larger than the expansion. Because of this (recognized) bifurcation, the mapper reduced Δr and constructed portraits at $r = 27.5$ and 28.75. The former contained a fixed cell and the latter a strange attractor, so the true bifurcation lay between the two, but a more exact measure of its location was inaccessible because of D_i (or, indirectly, the limits

Figure 7.9: Lorenz attractor for r = 30

Figure 7.10: Lorenz attractors for $r = 35$ and $r = 40$

Figure 7.11: Lorenz attractors for $r = 56.25$ and $r = 60$

that dictated it.)

As r was raised further, the attractor shifted and expanded; nothing interesting enough to pique the mapper's attention occurred until one of the lobes entered the endpoint cell containing **B**. *Perfect Moment* sensed this event at $r = 40$, again reducing Δr twice and constructing portraits at $r = 37.5$ and $r = 36.25$. The range containing the actual event was narrowed to [36.25 37.5]. Portraits of the $r = 35$ and $r = 40$ attractors are shown in figure 7.10. (draw-cross -24.5 68 .5) The $r = 36.25$ and 37.5 attractors are almost identical to the $r = 35$ one, but each is successively higher, closer to the destination endpoint cell. The integration is, as in figure 7.1, much longer than ten orbits in an attempt to visually duplicate the coverage effect of the 125 trajectories that the mapper normally generates.

The mapper then produces four more portraits at the $\Delta r = 5$ spacing: $r =$ 40, 50, 55 and 60. All are garden-variety chaotic attractors, but the latter extends over the region boundary (the user didn't get it quite right when she was hand-crafting the region to include all attractors) and is classified as a `sink-cell` trajectory, causing the mapper to zero in again. The $r = 57.5$ attractor also exits the region, but the $r = 56.25$ one does not; see figure 7.11.

The interesting part of the transcript of the mapper's run looks like:

```
(map: parameter = 10)
(map: parameter = 15)
(map: parameter = 20)
(map: parameter = 25)
(map: parameter = 30)

(zeroing in because of bifurcation between
  (fixed cell at param = 25) to
  (strange attractor at param = 30))
(map: parameter = 27.5)

(zeroing in because of bifurcation between
  (fixed cell at param = 27.5)
  (strange attractor at param = 30))
(map: parameter = 28.75)

(zeroing in because of bifurcation between
  (fixed cell at param = 27.5)
```

```
  (strange attractor at param = 28.75))
(map: parameter = 28.125)
(iteration depth reached!)

(map: parameter = 35)
(map: parameter = 40)
(zeroing in because of attractor
  entering endpoint cell at param = 40)
(map: parameter = 37.5)

(zeroing in because of attractor
  entering endpoint cell at param = 37.5)
(map: parameter = 36.25)

(zeroing in because of attractor
  entering endpoint cell at param = 37.5)
(map: parameter = 36.875)
(iteration depth reached!)

(map: parameter = 45)
(map: parameter = 50)
(map: parameter = 55)
(map: parameter = 60)

(zeroing in because of bifurcation between
  (strange attractor at pa<ram = 55)
  (sink-cell at param = 60))
(map: parameter = 57.5)

(zeroing in because of bifurcation between
  (strange attractor at param = 55)
  (sink-cell at param = 57.5))
(map: parameter = 56.25)

(zeroing in because of bifurcation between
  (strange attractor at param = 56.25)
  (sink-cell at param = 57.5))
(map: parameter = 56.875)
(iteration depth reached!)
```

The resulting collection of portraits was then examined by the optimization and tiebreaker functions for paths between the grid squares containing **A** and **B**. The $r = 50$ portrait contained the best segment, shown in figure 7.12 and hereafter designated S^0:

Figure 7.12: Core segment (r = 50) and new search region for the next pass

```
(segment found:
  (#(8.82 -4.37 57.21)
   #(-24.51 -19.99 68.05)
   50
   (#(-45 -205 -5) #(55 195 95))
   #[compound-procedure 87]
   #[compound-procedure 88])
```

Its endpoint S^0_{final} was within the tolerance of **B**, so no additional path segments were required on the destination end of the core segment. However, the gap at the other end *did* present a problem. The nature of the projection makes the distances deceptive; **A** is actually quite far above (in the y-direction suppressed in this projection) the nearest threads of the $r = 50$ attractor. The trajectory $\phi_t(\mathbf{A}, 50)$ emanating from the origin with $r = 50$ misses the control parallelogram completely. In fact, this trajectory is the very one upon which it was *Perfect Moment*'s mission to improve (see figure 7.4.)

To find the subpath that spans the gap, the mapping and search were repeated in the region surrounding **A** and S^0_{init}. The new region, outlined by the square in figure 7.12, was determined by recentering the old endpoint cell that contained the offending gap; see the dashed square in figure 7.12. The new grid spacing within this cell was then determined, starting at $m = 5$ (the second layer of dotted lines in the figure) and working upwards or downwards according to the revision rules in section 5.3.2.

The new $20 \times 80 \times 20$ search region in the dashed box was divided into 125 $4 \times 16 \times 4$ cells. The centerpoint of the new origin endpoint cell — `#(3 4 2)` in the new coordinate system — was $\vec{x}_c$ = `#(9 27 65)`. The difference vector $\vec{x}_\Delta$ from **A** to S^0_{init} was

```
#(8.8230 -4.3716 57.2139) - #(8 29 64)

= #(.8230 -33.3716 -6.7861)
```

and the *scaled* difference vector $\vec{V}$ was $\frac{1}{5^2}\vec{x}_\Delta$ = `#(.0329 -1.3349 -.2714)`.

The variational system (part (c) of figure 5.14) was integrated forwards in time, with $r = 10$, from the initial condition

```
#(9 27 65 .0329 0 0 0 -1.3349 0 0 0 -.2714)
```

until the state-variable section of the trajectory exceeded the bounds of the cell at $t = 0.0635$ and

```
#(6.227 -8.143 55.085        ; x y z
```

```
-.010 -.036 -.007          ; del.x.x del.x.y del.x.z
-.366 .100 -.503           ; del.y.x del.y.y del.y.z
 .046 .084 -.174)          ; del.z.x del.z.y del.z.z
```

Comparing the magnitude of the column sums of the variational system `#(-.330 .148 -.684)` to the cell size `#(4 16 4)` yielded a vector of c_j^{10} values `#(.0825 .0093 0.1712)`, all less than one. The largest — c_3^{10} — was recorded. This entire process was repeated for $r = 15, 20, \ldots, 60$. The scaled and evolved variation never exceeded the cell size, so the maximum of all the c_j^ss, $c_3^{50} = 0.224$, was used to adjust m to $\lceil 0.224 \times 5 \rceil = 2$.

The mapping is repeated with these new region and grid parameters. The process is similar to the first pass, but the size of the region affects the dynamics classification. All of the attractors are either completely or partially outside the new region, so all trajectories are classified as headed for the `sink-cell` and the mapper never reduces Δr and the portraits are evenly spaced at $r = [10, 15 \ldots 60]$. The $r = 35$ attractor is the first to even touch the search region; the $r = 50$ attractor, if not truncated, would cover about half of its volume.

The segment S^1, found on the $r = 40$ map, connects the new endpoint cells around **A** and around S^0's endpoint. The two-segment path $[S^1, S^0]$ fails the tolerance check because its endpoint is outside the range of the linearized controller at S^0_{init}. Because the gap at the endpoint causes problems and the Lyapunov exponent is positive on the attractor from which S^1 was drawn, the gap at the beginning is automatically refined as well.

In the recentering and redivision of both $4 \times 16 \times 4$ endpoint cells, the initial condition for the variation is smaller than the one used in the $1^{st} \rightarrow 2^{nd}$ pass revision. $\vec{x}_\Delta$ — and hence $\vec{V}$ — are at least a factor of two smaller, the integration distance over which the variations can evolve is shorter, the dynamics don't have any local wild singularities that remained undetected in the previous pass or that respond particularly emphatically to the new direction of the variational system[4], and thus all $c_j^r \ll 1$ and the $\frac{m_{i+1}}{m_i} \geq 2$ cap (step 6 of the synopsis on page 83) kicks in to set $m = 2$.

Working with $4 \times 16 \times 4$ search regions and $2 \times 8 \times 2$ cells, the third pass of the mapper again generated eleven equally-spaced portraits whose trajectories almost all went to the `sink-cell`. Searching these portraits, the path finder chose two very short segments S^2 and S^3, both at $r = 10$, that connected S^1 to **A** and to S^0, to the final satisfaction of the tolerance check.

A schematized version of the overall path $\Pi_{AB} = \{S^2, S^1, S^3, S^0\}$ is shown in figure 7.13. It is composed of the four segments discussed in the previous

[4]if one direction $\vec{e}_{trouble}$ has a huge Lyapunov exponent and the first-pass $\vec{x}_\Delta$ happened to be orthogonal to it, the variations will be much more spectacular on the second pass and m will be revised far upwards.

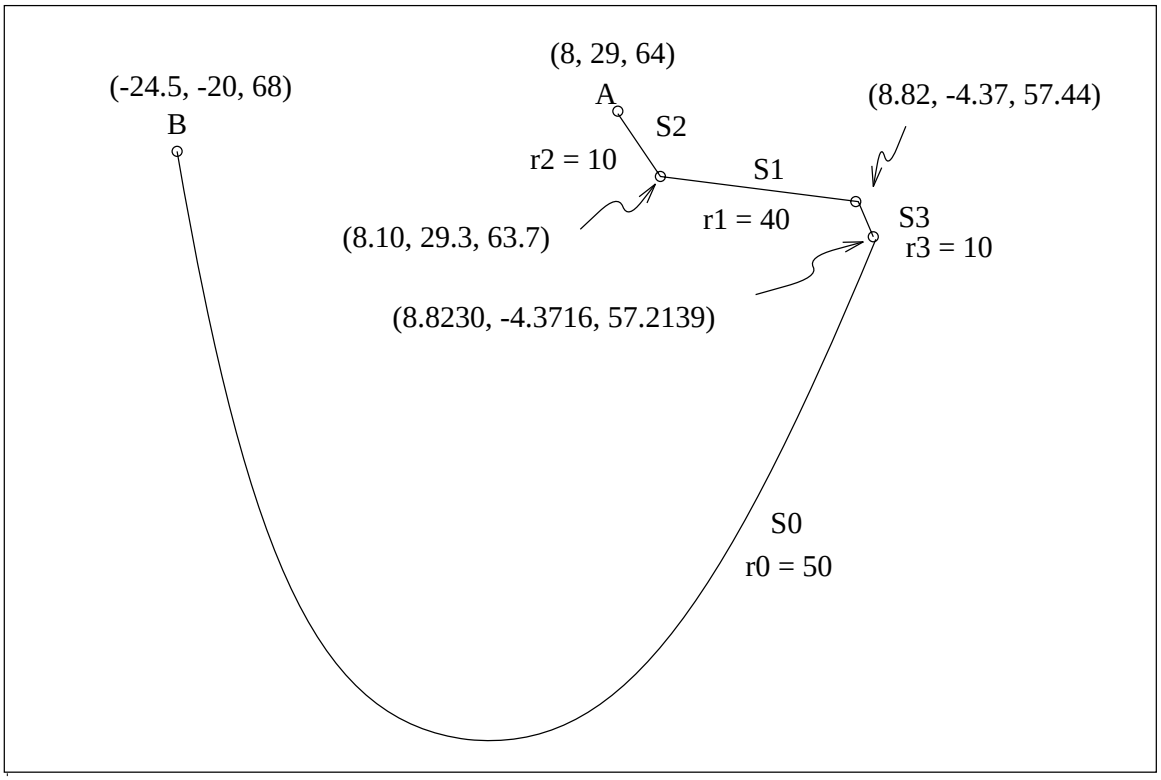


Figure 7.13: Schematized segmented path between **A** and **B**

Figure 7.14: Numerical integration of segmented path

paragraphs. The two longer segments are segments of chaotic attractors; the two shorter ones, enlarged so as to be visible, are sections of transient trajectories ultimately destined for one of the system's fixed points. The former are examples of a "strange attractor bridge", connecting two otherwise-unreachable points. The values for the r_i and the S^i_{final} are shown next to the segments and their junctions. The actual numerical integration of the path Π_{AB} from **A** to **B** along the path is shown in figure 7.14. On this (true) scale, the smaller connecting segments are invisible.

Several important observations and conclusions should be noted this point:

- The $\{S^2, S^1, S^3\}$ subpath actually moves the system state *directly away from* **B**. This locally counterintuitive move is made in order to reach a globally good path.
- The parameter switch that initiates S^0 is perhaps the most critical of the three, as nonlinear expansion along this segment's great length can severely magnify any error. This sensitivity was reflected in the care with which *Perfect Moment* was forced to refine the subpath that connected **A** to this point.
- Since S^0 actually enters the control parallelogram around **B**, this process of connecting to it amounts to a solution to the "target acquisition problem" of [59][5]. The path length between **A** and **B** is 130.1 normalized distance units, requiring 0.3567 normalized time units to traverse. The contrast to the case *without* active target acquisition[6] is striking.

7.1.2 Targeting a Fixed Point

If the control destination is near one of the system's low-r fixed points, one need not rely on UPO stabilization — parameter-controlled movement of the fixed points can be used to force convergence to the control objective. Though the destination can then be reached with a single segment from any point in the basin, the use of a multi-segment path can alter macroscopic quantities like convergence speed — and reachability, if the initial condition is *not* in the same basin as the desired destination. For example, a trajectory starting from the point **A** = `#(22.4 30.5 60)` at the value $r = 25$ would normally converge to the left hand fixed point **C**

[5] A solution to this problem is presented in a paper from the same group, currently in press in Physica A.

[6] 25533 and 104 units, respectively; see figure 7.4

Figure 7.15: Segmented path to a fixed point

= `#(-10.40 -10.40 27.01)` on figure 7.15 along the tightly-wound spiral at the bottom left.

Perfect Moment found the right-hand path in the figure, acting upon inputs identical to those in section 7.1.1 except for the origin, destination and tolerance. This path contains two segments: an $r = 60$ trajectory that travels from **A** deep into the basin of the *other* fixed point, which lies near **B** = `#(10.40 10.40 27.01)`, followed by a short section of the $r = 25$ spiral that surrounds the second fixed point.

Note, again, that the controlled trajectory's initial counterintuitive move is directly away from the objective **B**. Since the origin(**A**) is actually in the basin of attraction of the *other* fixed point **C** —- as indicated by the uncontrolled trajectory from **A** in figure 7.15 — the strange attractor segment may again be thought of as forming a bridge: not only over the basin boundary, but also over most of the slow spiral around **B**, which speeds the convergence. Note also that no additional control is required here after the first segment is complete, unless uncertainty or noise exceed the distance from the fixed point to the basin boundary. One simply sets $r = 25$ and the system's fixed point holds the state within the tolerance.

7.2 The Driven Pendulum

The driven pendulum is arguably the most closely-studied simple chaotic system (e.g., [18, 27, 37]) and is of interest here for a variety of reasons. It has many practical applications, from robotics to offshore drilling platforms to earthquake-proofing of buildings. It is isomorphic to many other interesting systems, among them the Josephson junction and the phase-locked loop of section 7.3. Used here as an example, it serves to demonstrate how *Perfect Moment* works on a nonautonomous system. It is a good point of reference for comparisons to traditional nonlinear control, as virtually every text in the field presents it as *the* classic problem. Finally, as a test fixture for computer-aided control via a potentially ticklish algorithm, a time constant on the order of a hundred milliseconds is far less daunting than the kilohertz (or higher) bandwidths found in electronic circuits, even if one has access to a powerful computer with fast and accurate I/O.

Consider the pendulum shown in figure 7.16. A mass m on a rigid, massless rod of length l pivots freely in the plane of the paper around the point A. Writing $F = ma$ at the mass yields

$$ml\frac{d^2\theta}{dt^2} + \beta l\frac{d\theta}{dt} + mg\sin\theta = \frac{T(t)}{l} \tag{7.2}$$

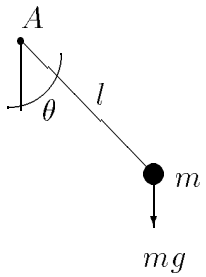


Figure 7.16: Geometry of the simple pendulum

Where β is the coefficient of the friction, referenced to the location of the mass, and $T(t)$ is the torque applied at the pivot.

Scaling time by $t = \sqrt{\frac{l}{g}}\tau$ and using the substitutions $\gamma(t) = \frac{T(t)}{mgl}$ and $B = \frac{\beta}{m}\sqrt{\frac{l}{g}}$ gives

$$\frac{d^2\theta}{d\tau^2} + B\frac{d\theta}{d\tau} + \sin\theta = \gamma \tag{7.3}$$

which can be rewritten as the following system of first-order equations:

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= \gamma - \sin\theta - B\omega \end{aligned} \tag{7.4}$$

The only differences between equation (7.4) and equations (3.7) or (7.2) are the scaling and the substitutions. Of all three equations, (7.2) is the most conducive to physical intuition, but the normalized forms are more common in the literature. This particular set of normalizations maps B to the damping ratio ($2 \times$ actual/critical); the time unit is the inverse of the natural frequency. Both [18] and [27] fix the damping ratio at $B = 1/4$ and apply a sinusoidal torque per unit mass $\gamma(t) = \epsilon gl\cos\Omega t$ at the pivot, where Ω is the ratio $\frac{\omega_d}{\omega_0}$ of the forcing frequency to the natural frequency $\omega_0 = \sqrt{g/l}$.

It is mechanically easier to move the pivot point than to apply a torque to the shaft. If the vertex of the device is driven vertically with a sinusoid of amplitude α and frequency ratio Ω, the equations are

$$ml\frac{d^2\theta}{d\tau^2} + \beta l\frac{d\theta}{d\tau} + \sin\theta(mg + m\alpha\Omega^2\sin\Omega\tau) = 0 \tag{7.5}$$

Gravity is effectively modulated ($\alpha\Omega^2\sin\Omega\tau$) by the movement of the pivot. The corresponding normalized, pasteurized, homogenized and condensed equations are

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= -\sin\theta(1 + A\Omega^2\sin\Omega t) - B\omega \end{aligned} \tag{7.6}$$

Figure 7.17: Movement of driven-vertex pendulum (twelve time-lapse images starting at $(t,\ \theta,\ \omega) = (0,\ 1.9,\ 0)$

The substitutions are the same as before, with the addition of $\alpha = gA$, which makes the modulating term dimensionless.

If the axis along which the vertex is driven is tilted sideways to an angle ϕ from the positive x-axis, the equation is

$$ml\ddot{\theta} + mg\sin\theta + m\alpha\Omega^2 \sin\Omega t \cos(\phi + \theta) + \beta l\dot{\theta} = 0 \tag{7.7}$$

which translates to

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= -\sin\theta(1 + A\Omega^2)\cos(\phi + \theta) - B\omega \end{aligned} \tag{7.8}$$

Finally, if the device is inclined to an angle ψ with respect to the positive z-axis, gravity is reduced to $g\cos\psi$ in all of the equations above. This reduction is mechanically useful; it reduces the speeds at which interesting things happen — so they are visible and so the design need not withstand the higher frequencies ($\rightarrow$ energies.)

In (7.2), (7.5) and (7.7), both the amplitude and the frequency of the drive play roles in chaotic state-space changes. The behavior — and mutations thereof — of all three systems is quite similar, though the particular parameter values that cause different bifurcations differ by the appropriate normalization factors. The driven-vertex versions are a bit different in spirit, as the torque actually delivered by the movement of the vertex depends on the angle of the pendulum. See figure 7.17.

If the drive is not turned on, the system cannot be chaotic — it does not have enough dimensions. The analysis is relatively simple; the simple pendulum is one of the few nonlinear systems that have a closed-form solution[7]. The device has a stable fixed point hanging downward ($\theta = 2n\pi$) and an unstable fixed point inverted ($\theta = \pm(2n-1)\pi$.) The state space is cylindrical, as suggested by these formulae; however, points separated by $2n\pi$ are *not* always identical for the purposes of control[8]. For small θ, one approximates $\sin\theta \approx \theta$; solutions to this approximated equation are similar to the voltage and current in a linear RLC circuit or a damped harmonic oscillator. For larger θ, this approximation fails and elliptic functions must be used to obtain a closed-form solution. The state-space portrait of an undamped, undriven device ($\beta = \gamma = 0$) is shown in figure 7.18. The closed elliptical orbits — "oscillating solutions" — at small θ deform into oblate spheroids and then into hyperbolic separatrices as the angle increases and the nonlinearity

[7]Hence much of its attention in textbooks.

[8]Consider, e.g., the long-term effects of ignoring the rotations of a telephone handset.

Figure 7.18: State-space portrait of the simple pendulum

Figure 7.19: Driven pendulum Poincaré sections (a) damped oscillations with small drive ($A = .01,\ \Omega = 30$) (b) chaotic behavior with larger drive ($A = 1,\ \Omega = 1.5$.) Rectangle corners are (0, -3) and (2π, 3).

begins to exert its influence. Outside the separatrices, the device follows "running" solutions, where the sign of the angular velocity remains constant. The curves on figure 7.18 can also be obtained by writing the Lagrangian (or Hamiltonian) for the pendulum and using conservation of energy. Damping ($\beta > 0$) simply makes the trajectories spiral inwards to one of the $2n\pi$ fixed points, perhaps rotating a few times around the cylinder in the process if the initial energy is high enough.

With a nonzero drive, the behavior is more complicated and interesting. For small-amplitude drives, the damping dominates, causing the oscillations to slowly die out, as shown in the Poincaré section of figure 7.19(a). The spiral appears to be sliced in half because of the way that the cylinder is unrolled onto the page. Larger amplitudes and/or frequencies cause chaotic behavior, as in figure 7.19(b). See [18, 19, 27] for simulations of this system over a wide range of drive frequencies and amplitudes, together with derivations of the resonance curves and stability regions that define and describe the different types of behavior. The discussion in this paragraph and the preceding one is valid for all of the equations in this section; in the discussion that follows, however, only the driven-vertex equation will be used.

In the experimental setup constructed as a testbed for *Perfect Moment* — which is modeled by equation (7.6) and shown in the photographs in figure 7.20 — changing A is difficult[9], so Ω was chosen as the control parameter. The coefficients in these simulations — damping, length, etc — were chosen to reflect those of this physical system as closely as possible. A positive input voltage causes the motor shaft to spin clockwise and a negative one causes it to spin counterclockwise; the direction in which the pendulum is driven depends on the position of the arm as well as on the shaft rotation. Thus, changing the *sign* of Ω is possible as well; this leverage is used by the junction controller but not in the segmented path. Note that the pendulum "bob" is a rectangular block of aluminum with a milled slot, not a ball on the end of a massless rod. The bob geometry shown in the picture reduces the moment of inertia — and the coefficient of the $\ddot{\theta}$ term in (7.6) — by about a factor of ten. The damping ratio is initially fixed at an experimentally-measured value of 0.25, but this value can become invalid as, for example, the bearings heat up or wear out. This variability is actually a symptom of a much harder and more interesting problem that will be revisited in section 7.4. Finally, note that gravity has half a dozen settings on this particular pendulum, one for

[9]Note the linkage in the closeup of part (b) of the figure.

Figure 7.20: Photographs of the experimental pendulum setup

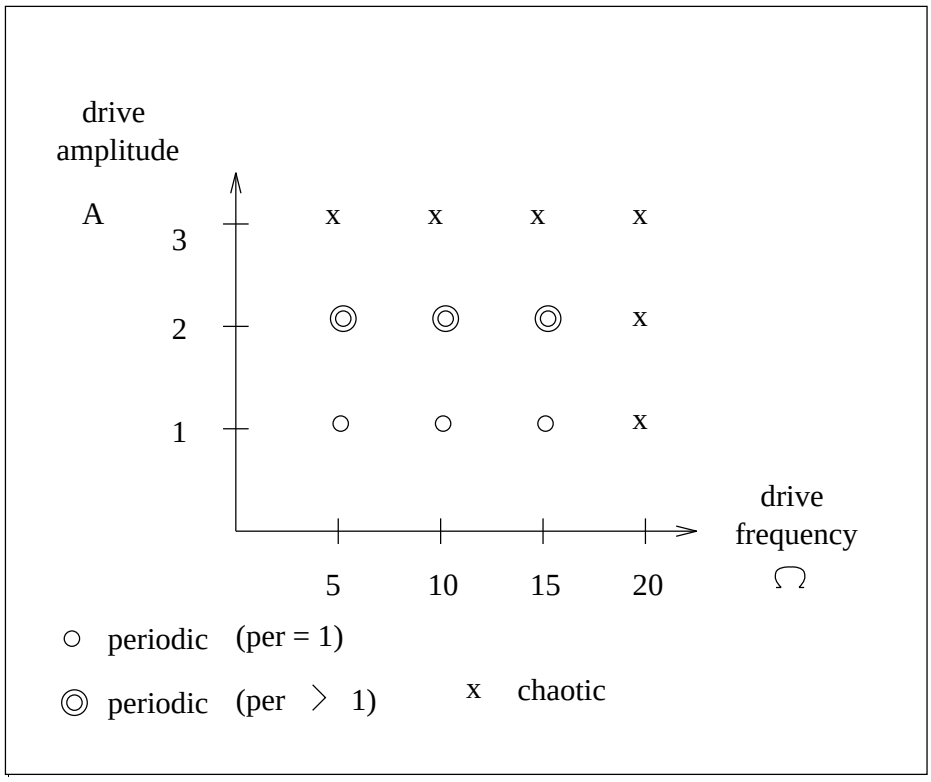


Figure 7.21: Parameter-space plot for the driven-vertex pendulum

Figure 7.22: Controlling the driven-vertex pendulum (a) origin and destination points (b) uncontrolled trajectory from origin

each set of holes drilled in the fan-shaped supports.

A coarse-grained parameter-space plot for (7.6) is shown in figure 7.21. The circles represent periodic behavior (single for period-one and double for period-two) and the $\times$s chaotic behavior. There is a wealth of detail hidden between these few points; see one of the cited papers for more detail. Fixing A constrains the domain of the system to one horizontal row of the graph.

Perfect Moment was used to balance the pendulum inverted — at $\mathbf{B} = (\pi, 0)$ — from some random initial condition $\mathbf{A} = (-\pi/2, 15)$. See figure 7.22(a). If the pendulum is started from $\mathbf{A}$ with no applied torque, the uncontrolled system follows the trajectory shown in figure 7.22(b). The initially-high kinetic energy gets dissipated over eight circuits of the cylinder; the trajectory then ceases running and begins to oscillate down to the fixed point. At no time does it closely approach $\mathbf{B}$.

While the linearized controller of section 3.1 could certainly balance this device, it would require large torques, many reversals of the motor voltage polarity at

The short-term control torques exerted by the global controller to bring the device into this range are allowed to be much ($\approx 20^2$ times; see below) higher. For the parameter variation that corresponds to one torque unit, the state-space range of the controller is far less than the $(\pi/2, 15)$ that would be required to capture the trajectory from **A**. Once the segmented path gets within striking range, however, it is a highly-efficient scheme.

As an interesting aside, a rather-mindless control scheme that relies only upon parametric resonance could also be used to stabilize the inverted pendulum with regard to small deviations. If the drive frequency is high enough — ≈ 2.4 Hz in the physical setup, at the "middle gravity" setting — the $2n\pi$ points lose their stability. At yet-higher frequencies (≈ 4.8 Hz), the $\pm(2n-1)\pi$ points actually become stable. Figure 7.23 shows two rough versions of the parameter-space stability region of the inverted equilibrium point, one computed for a starting position of $\theta = 2.9$ and one for $\theta = 3.1$. The latter is larger, as the energy required to stabilize the closer initial condition is smaller.

Inputs to the program were:

- the tolerance computed above for the one torque unit control input
- equations (7.6)
- a range [0 20] and step $\Delta\Omega = 5$ for the frequency parameter Ω
- an iteration depth of 2
- region parameters $R_0 = 1.5$ and $m_0 = 3$.

As discussed in chapters 4, 5 and 6, the mechanics of the program are very different when the target system is nonautonomous. At each step, *Perfect Moment* evaluates a fan of trajectories emanating from that level's origin, one for each value of the drive frequency. This restriction stems from the "state" requirements imposed by the position of drive arm, and the resulting behavior is very similar to that of shooting algorithms[76] or other schemes which try a variety of parameter values, evaluate the behavior, and use the results to modify the next attempt[87]. Because the state space is cylindrical, only the ω-axis gets multiplied by the overrange factor. The $R_0 = 1.5$ choice is large enough to include some but not all of the attractors. The cell aspect ratio was not out of line with the attractor aspect ratio, so no hand-crafting was performed to adjust the region and cell size. Finally, recall that tolerance checking is considerably simpler for nonautonomous systems because the segmented path evolves forwards from the origin and is always connected.

The other inputs are based on the parameters of the experimental setup. The natural frequency of the device at the second-to-lowest gravity setting — $\psi = \frac{\pi}{4}$ — is $\approx .4$ Hz and the motor is capable of ≈ 7 Hz, so the [0 20] range reflects the true limits of the physical system. The step and iteration depth require a resolution of 1 Hz ($\Delta\Omega = 2.5$.) This extremely pessimistic guess was based on bad experiences with the slop and hysteresis in the dial. The very same empirical observations suggested that the separation between different behavior types is *less than* 1 Hz. This combination has depressing implications for both analysis and

[10] i.e., `(mapping region from ... ) (zeroing in because of ... )`

Figure 7.24: True dynamics for $\Omega = 15$: a quasiperiodic orbit. (a) full-scale Poincaré section (b) close-up of small square region

synthesis; among other things, it means that many interesting effects escape notice because they fall between the mapper's snapshots. One obvious solution to the inadequate-resolution problem — to tilt the pendulum even further back and slow everything down — was not pursued because of a design parameter that limited long-term use in that position[11]. Incidentally, if the range and step for Ω are such that those snapshots occur at integer multiples of the natural frequency (i.e., $\Omega = 4$ instead of 5 here), and the drive amplitude is high enough to overcome the damping, the pendulum spins madly and the mapper never finds anything besides periodic running solutions. *Perfect Moment* has no way to recognize or recover from this pathological combination of inputs. In light of the prediction that virtually every parameter step will set off a bifurcation, the same ends could have been accomplished more quickly if $\Delta\Omega = 2.5$ and $D_i = 1$ were specified and the dynamics classification scheme were turned off. The rationale behind the small m-value is that the bifurcations will be so sweeping and easy-to-recognize that a coarse discretization will be adequate for recognition. The successes and failures of the combination of these various choices are discussed later in this section.

The mapper constructed six portraits, described here in the order in which they were produced. The classifications and the true dynamics, if different, are described after each Ω-value. All θ values are mod 2π.

Depth	Ω	**Classification**	**True Dynamics**
1	0	`fixed cell`	
1	5	`sink-cell`	running oscillation (periodic one-cycle) with $(\theta, \omega) \approx (.53\pi, -3.64)$ on the Poincaré section
2	2.5	`unknown`	running oscillation (periodic one-cycle) with $(\theta, \omega) \approx (0, -1.83)$ on the Poincaré section
1	10	`sink-cell`	running oscillation (periodic one-cycle) with $(\theta, \omega) \approx (.50\pi, 7.30)$ on the Poincaré section
1	15	`sink-cell`	quasiperiodic orbit; see figure 7.24.
1	20	`sink-cell`	chaotic attractor.

The mapper did not do very well here. The main cause of this failure is the limited region size; the cells are small enough to allow adequate classification, but the attractors leak over the boundaries. The intent of this example is to design an adequate controller, not to demonstrate the mapper's capabilities; the latter is emphasized in the other two examples in this chapter. All of these classifications could be improved with the investment of more computer time (via larger R and m.) In fact, that was how the "actual dynamics" data in the right hand column above were generated.

[11] I used radial-load bearings, not thrust or angular-contact bearings.

Figure 7.25: Portraits for Ω = (a) 5 (b) 7.5 (c) 15

Figure 7.26: The segmented path: (a) the core segment (b) the second-pass search region (c) the second segment

Three of the portraits — at $\Omega = 5$, 7.5 and 15 — contained trajectories that touch the cell containing **A** and the cell containing **B**. See figure 7.25. The latter was the best of the three:

```
(segment found:
 #(-1.57 -15)
 #(-2.8082638490023566 -.4785929790001656)
 15
 #(0 -22.5) #(6.283185308 7.5)
#[compound ... ]
#[compound ... ])
```

This segment, part of the origin trajectory $\phi_t(\mathbf{o})$, is plotted in part (a) of figure 7.26. The path did not pass the tolerance check, as its endpoint did not fall within the specified range of **B**, so another pass was necessary to close the (single) gap between this core segment and the objective.

The endpoint cell was recentered around the points `#(-2.808.. -.478..)` and `#(-3.14 0)`, then divided into nine smaller cells. Here the state space is *not* treated as cylindrical[12]; see part (b) of figure 7.26. The variational integrations from the new endpoint cell, tested across the parameter range, indicated a new m of 9. The three-fold increase was due, understandably, to the highest ($\Omega = 20$) parameter value. All trajectories exited this small region and were classified as `sink-cell`, so the mapper never reduced Ω and produced five equally-spaced plots.

The segment returned by the second mapping/search pass, found on the $\Omega = 0$ portrait and shown in part (c) of figure 7.26, brings the path into the local-linear controller's range, so no further iterations were required. The total path is shown in figure 7.27.

The physical interpretation of this path is interesting; the controller jerks the drive arm hard enough to get the pendulum upstream from the destination, then turns off the control ($\Omega = 0$) and lets the device coast into the linear controller's clutches. The linear controller performs a complex, continuously-varying control

[12] unless the search fails

Figure 7.27: The controlled trajectory

Figure 7.28: A controlled trajectory requiring less torque and more time. *Perfect Moment* was given a more limited frequency range and was forced to explore more subtle solutions. This trajectory "pumps" up from the initial condition over several cycles, attaining the control destination using one ninth the torque used in figure 7.27 — but over 45 times slower. A linear controller, allowed this amount of torque, would not be able to reach the destination

action, based on the system's evolving behavior, while *Perfect Moment* works with a single discrete switch. Conversely, the former can adapt to unforeseen behavior more readily.

If the local-linear controller were to attempt this task, the peak required torque be ≈ 130 normalized units. *Perfect Moment*'s maximum requirement is a fair bit more than this — on the order of Ω^2, or 225. In other words, if minimizing the maximum torque required from the controller is the criterion for success, the classic linear control wins in this case.

This comparison, however, is unfair. If the torque is limited to a value *lower* than 130 units, the linear controller will fail, but *Perfect Moment* can still succeed. The trajectory shown in figure 7.28 is the result of a run that is identical except for the parameter range [0 10] and step $\Delta\Omega = 2.5$ (both halved.) Here, *Perfect Moment* "pumps" the trajectory up from the origin using $\Omega = 3.5$ — ≈ 12 units of maximum torque. It makes one small correction at the end of this process, using a slightly higher ($\Omega = 5$) frequency to reach the linear controller's domain after 6.05 normalized time units. Allowed this amount of torque, a linear controller would lift the pendulum up to an angle χ ($mg \sin \chi = 25$, modulo normalizations) and then hold it there, unable to go higher. Incidentally, *Perfect Moment* only turned the drive to $\Omega = 5$ for the last 13% of the time.

The first set of parameter values subtly and effectively torpedoed the program by causing it to search for a fast, brute-force reference trajectory — a task that it is *not* very good at because of its quick-and-dirty approach to optimization[13]. The tighter requirements in the second run forced the program to retrench and find a roundabout trajectory that finessed the control objectives. The program would have selected this trajectory to begin with if (1) the original iteration depth had been three instead of two on the first run and (2) the rough-cut optimization function minimized closest approach distance instead of state-space pathlength.

The main *conceptual* differences between this section and the previous one all stem from the pendulum's nonautonomous nature. The *practical* differences give the example a different feel; choices are no longer antiseptic and mathematical, but driven — and limited — by real engineering constraints.

[13]Optimization clearly plays a more important role when the grain of the search space is excessively rough.

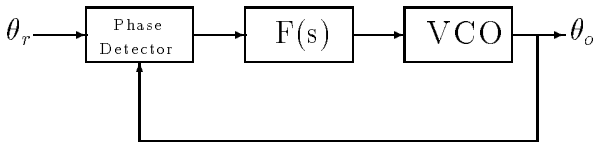


Figure 7.29: Phase-locked loop block diagram

7.3 The Phase-Locked Loop

A phase-locked loop (PLL) is an electronic circuit that tracks the frequency and phase of an input signal. In one particular class of these circuits, the differential equations that describe the evolution of the locking process are identical, within coefficient values, to equation (7.2). Inputs and implementation can be analog or digital; many PLLs contain a mix of both types of components. The heart of the circuit's feedback loop is a variable oscillator (VCO.) A phase detector (PD) — the main source of the nonlinearity and of the harmonics that necessitate the filter — compares the VCO's output to the input signal and uses the difference to adjust the VCO's phase and frequency so as to drive $(\theta_o - \theta_{ref})$ to zero. Figure 7.29 shows a block diagram of a simple PLL. Loop compensation is lumped into F(s) in this figure as well. Once a PLL is locked to a signal, changes in frequency can be tracked over some *lock range*. Initial lock can be acquired over a smaller *capture range*. The range of applications of such circuits is limited only by the imagination of the user; some of the more common are as frequency synthesizers, tracking filters and signal modulators and demodulators.

A digital PLL can be as simple as an XOR gate phase detector feeding a dressed-up counter VCO. An S-R latch is a somewhat better PD, a pair of D flipflops better still. A virtually endless collection of flipflops and gates can be tacked on until the circuit's timing and state are completely customized. The VCO can be a divided-down clock with a variable divisor or an autonomous oscillator. Design strategies depend on requirements; a crystal oscillator, for example, is stable but has a limited range. Digital PDs and VCOs can even be used in analog PLLs, if accompanied by good filters to smooth out the square corners and perhaps an accumulator or charge pump to convert pulse trains to DC levels.

Analog phase detection circuits are much more subtle. Multipliers are an obvious solution; multiplying the two sinusoids $v_R = a\sin\omega_0 t$ and $v_o = b\sin(\omega_0 t + \phi)$ gives

$$v_R v_o = \frac{ab}{2}[\cos\phi - \cos(2\omega_0 t + \phi)] \tag{7.9}$$

A low-pass filter with an appropriate cutoff frequency is typically used to filter out

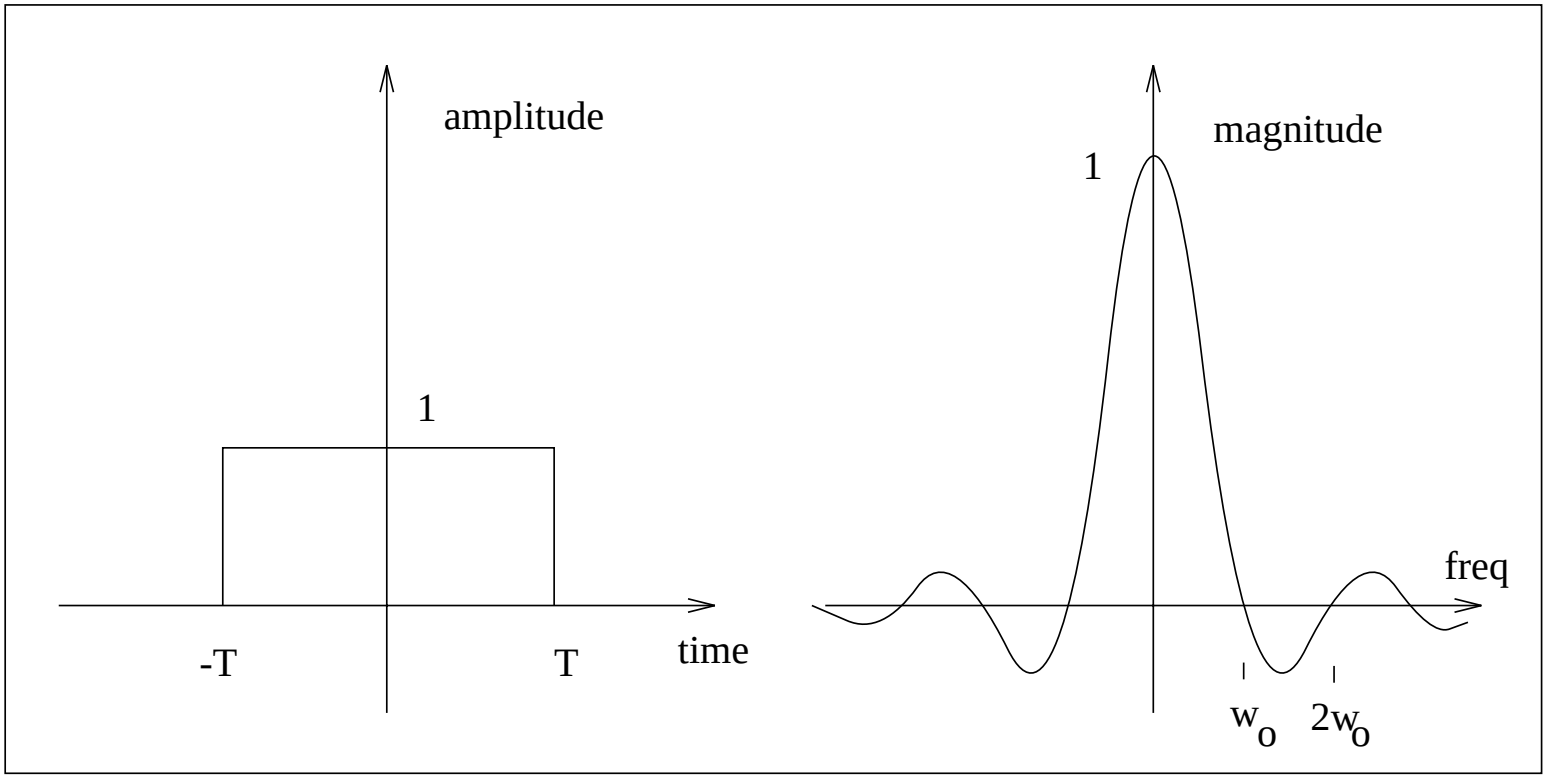


Figure 7.30: Time and frequency response of sample-and-hold: $\omega_0 = \frac{2\pi}{T}$

the second harmonic, leaving only the $\cos\phi$ term. Another type of analog phase detector uses sampling: v_R is strobed at the VCO's frequency. After the math is chased through, the equation contains a similar DC term[14] and *all* of the higher harmonics of the VCO frequency. Sample-and-hold circuits are sometimes used as filters in sampled PDs; if the hold time and sampling frequency are set up correctly, the S&H has zero gain at all of the troublesome harmonics. See figure 7.30 and the derivations later in this section. Filter designs can be found in handbooks, but designing around the linearity, switching speed and stability problems associated with multipliers, samplers, and sample-and-holds requires real creativity and skill.

PLL designers almost invariably linearize around the phase-locked state, which makes figure 7.29 look like figure 7.31. The VCO converts a phase to a frequency, so its transfer function is:

$$\frac{d\theta}{dt} = K_v V(t) \frac{\theta}{V(s)} = \frac{K_v}{s} \tag{7.10}$$

The Ks are the PD and VCO gains near the "locked" operating point. One can plug various kinds of inputs (step change in phase, steady ramp in frequency, etc)

[14] $\cos(\phi + \psi)$ instead of the $\cos\phi$ term in equation 7.9, where ψ relates input and VCO frequencies

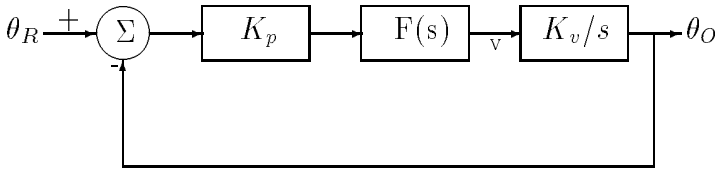


Figure 7.31: Linearized phase-locked loop block diagram

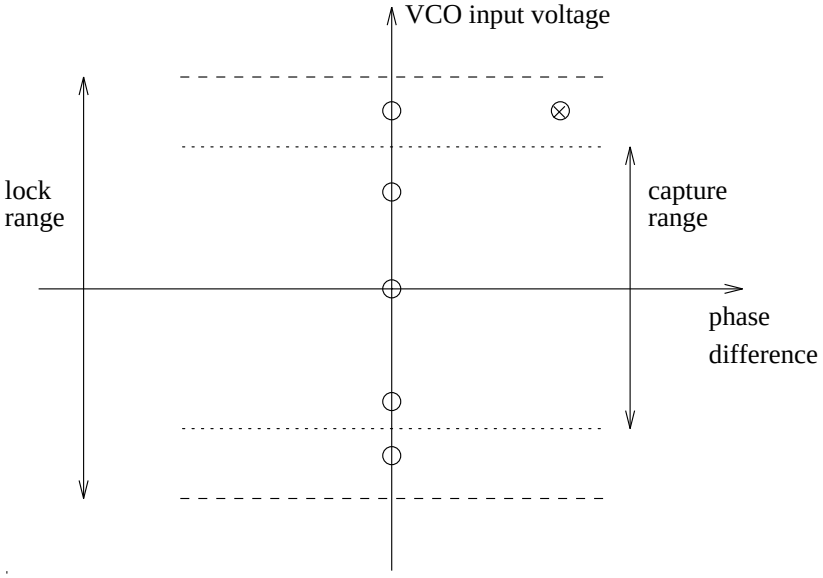


Figure 7.32: Lock and capture ranges

into this model, determine the loop's response and adjust F(s) to modify that behavior, much like the adjustment of closed-loop poles discussed in section 3.3. Samplers can be particularly hard to compensate, as the phase shift of a pure time delay is negative and increases linearly with frequency.

The capture and lock ranges depend in a complicated way on the global, nonlinear properties of the loop, particularly those of the nonlinear phase detector. These ranges define rectangular regions on the system's state space (whose axes are VCO input voltage and the phase difference $(\theta_o - \theta_i)$.) See figure 7.32. The points on the y-axis (zero phase difference) are locked states. The vertical value for each is the difference between that individual input frequency and the VCO's free-running frequency; the point at the origin represents a PLL being driven at the natural frequency of its VCO. The off-axis point $\otimes$ is outside the capture range.

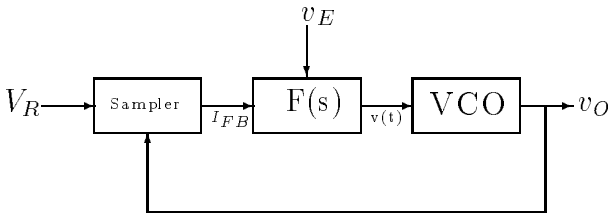


Figure 7.33: Sampled phase modulator

Note that its frequency is the same as that of the top "locked" point; the latter must have started out inside the capture range.

The lock range can be estimated by testing the linearized model on a frequency ramp input. If the loop is in lock but straining, the $\cos\theta$ term in the VCO's input ($K_v K_p \cos\theta$, assuming a slow ramp in frequency) approaches one, so the lock range is roughly equal to the DC forward path gain $K_v K_p$. The capture range is *smaller*: when the loop is out of lock, the difference voltage is beyond the filter's cutoff frequency and the loop has little or no leverage. Calculations for higher-order PLLs are made more difficult by poles at the origin and other mathematical problems[74].

When linearization fails, descriptions are vague and analysis becomes *ad hoc* and/or numerical: "A general expression for loop capture range is not available as the system is highly nonlinear."[74]. The exact form of the calculations depend on the dynamics of the phase detector.

Improving the capture range of a phase-locked loop is a global, nonlinear problem, an ideal test case for *Perfect Moment*. This application has several other attractive features: circuits are much easier to build and modify than machined metal objects (or low-temperature semiconductors — Josephson junctions are the other widely-studied variation on this system of equations[8].) Also, gathering data from a 3000 Hz PLL is far faster than studying a 0.5 Hz pendulum. These tradeoffs have led some authors, notably [27], to use PLLs in studies of pendulums.

A block diagram of the sampled PLL circuit that is actually used for the simulations in [27] is shown in figure 7.33. The main difference between this picture and figure 7.29 is the extra input v_E. This circuit is a phase modulator: if the input phase θ_R is constant, the output phase θ_O is proportional to the derivative of the modulating signal v_E, at least in the linear regime[15]. Without the driving term, incidentally, the equations are second-order, autonomous, and not chaotic. The sampled input, $V_R \sin(\omega_R \tau_n + \phi_0)$, and the perturbation, v_E, are routed to a

[15]Demodulation is even more straightforward: if a PLL is locked to a carrier signal, the VCO input voltage reflects changes in the carrier frequency.

first-order opamp RC filter. The differential equation for the VCO's input is

$$C\frac{dv}{dt} + \frac{v}{R} + \frac{V_R}{R_S}\sin(\omega_R\tau_n + \phi_0) = -\frac{v_E}{R_E} \tag{7.11}$$

where R and C set the filter's pole and the filter input voltages appear across R_S and R_E (the schematic is given in appendix B.) Substituting from equation (7.10) and recognizing that, because the sampler fires at the zero crossings of v_O, the sampled phase $(\omega_R\tau_n + \phi_0)$ is the same as θ gives

$$\frac{C}{K_v}\frac{d^2\theta}{dt^2} + \frac{1}{K_vR}\frac{d\theta}{dt} + \frac{V_R}{R_S}\sin\theta = -\frac{v_E}{R_E} \tag{7.12}$$

This derivation follows [27] and contains several strong simplifications; for example, the influence of gravity on the pendulum is continuous, while the circuit only samples the reference voltage discretely. These simplifications result in differences between the system and the model and all the attendant difficulties of those errors. Appendix B covers this development in more detail.

Comparing equations (7.12) and (7.2), one can identify interesting physical analogies between electrical and mechanical quantities. The $c\sin\theta$ term, for example, is due to gravity (via the pendulum's restoring torque) in equation (7.2) and due to the PLL's reference input v_R in equation (7.12). Damping and resistors are similarly related, as are momentum and capacitors. The voltage v_E corresponds to the external torque $T(t)$ applied to the pendulum. Once again, the equations are normalized to

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= \gamma - \sin\theta - B\omega \end{aligned} \tag{7.13}$$

identical to equation (7.2), but of course with different values for B, γ, etc. See appendix B for a table of coefficient correspondences between the phase-locked loop and the pendulum.

Though the physical interpretation is slightly different, the behavior of the system is, of course, identical to that of the pendulum. If the loop is locked to some reference frequency and the external drive kicks in, the circuit recovers — in the sense that it remains locked to V_R despite the external disturbance — for small-energy drives. It loses lock, almost always in a spectacularly chaotic fashion, for higher modulation amplitudes and frequencies.

A group at Berkeley is also exploring how to exploit nonlinear dynamics knowledge — and chaos itself — to improve the design and application of phase-locked loops. For example, lock range can be maximized by picking the best point in a stable basin whose (fractal) boundaries have been predetermined by a bifurcation analysis[12]. More recently, synchronization to a chaotic signal has been explored

for use in secure communications[23, 35]. This technique uses three coupled *digital* PLLs, two to transmit the chaotic signal and one to receive it[16].

Perfect Moment uses chaos to broaden the *capture* range of the circuit. The general idea is as follows: the program is presented with an input frequency Ω_x that is inside the lock range and outside the capture range: $\Omega_{capture} \leq \Omega_x \leq \Omega_{lock}$. It manipulates the frequency of the phase-modulating input $v_E(t)$ to find a value Ω_f that, in conjunction with the input Ω_x, creates a chaotic attractor that overlaps the capture range limits of the inner rectangle in figure 7.32. Whenever the input reference frequency Ω_x is detected, the loop is modulated at the predetermined frequency Ω_f until the system enters the rectangle. The drive is then immediately turned off, letting the circuit's original dynamics lock on to the signal. This type of strange attractor bridge, if it exists, effectively expands the circuit's capture range to include the entire lock range.

This scheme is robust with respect to state-space noise[17] because of the shadowing lemma. It does, however, have several potential drawbacks:

1. the time to achieve lock is effectively unpredictable in a particular case, though a stochastic average can be computed

2. parameter or modeling errors, if made near a parameter-space bifurcation boundary, have serious implications because they can cause the critical chaotic attractor to vanish

3. Ω_x must be known ahead of time and be detectable in real time

The first item is identical to the targeting problems in [59, 69], discussed in sections 3.4 and 7.1 and demonstrated in figure 7.4. From an engineering standpoint, this unpredictability is bearable: even if the worst-case time to achieve lock is long, the average time — essentially the ratio of the area of the capture range to that of the chaotic attractor — is reasonable. Moreover, the alternatives are infinitely long; recall that this frequency was *outside the capture range*. Rough solutions to the second and third drawbacks on this list could be obtained by widening the specifications and customizing the optimization functions. These functions could be programmed to use the dynamics classification information — that the mapper has computed anyway — to avoid chaotic attractors that are near bifurcation boundaries. One could direct the program to look for attractors that overlap a large state-space region around the objective, not just the endpoint cell. This type of extensive hand crafting was really not the point of this thesis and is not pursued here. Applying the techniques described here to a real PLL, rather than

[16]Synchronization between two systems linked by a chaotic signal was originally introduced by Pecora and Carroll[61].

[17]within the attractor's basin, of course

a simulation of one, will certainly require these problems to be worked out; the issues involved in doing so will probably fall more into the realm of circuit design than the development of programs that use domain knowledge to autonomously do interesting design tasks. See section 7.4.

The PLL is similar to the pendulum in that it is nonautonomous. It is different in that the classification accuracy is critical. These issues are reflected in the choice of inputs for the example that is presented in the remainder of this section. This usage is also different in spirit from the paths described in the previous example, as the control objective is a wide range and not a single point ("point to region steering.")

The target region is easy to characterize in this example: it is simply the original capture range. Two important modifications were used in the mapping process. The ten-orbit limit was extended to several thousand orbits and the mapper worked only with Poincaré sections. The former was motivated by the importance of making the right classification decision; the latter was an attempt to take less than a month to do so. The longer integrations are not the only source of added complexity. R_0 and m were also chosen very large to enhance the classification.

Most of the other input choices were guided by experimental constraints. The circuit of [27] was used as a model, so the published resistor and capacitor values give the parameter ranges and the ratios which relate them to the coefficients in equation (7.4). The frequency was again chosen as the control parameter, but the amplitude could easily have been used instead, in contrast to the previous example. Designers being what they are, it is fairly safe to assume that the input frequency is pushing the limits of the loop, so the modulating input should not be much higher. This motivates a parameter range choice of [0.1 1.5]. The drive amplitude was then chosen high enough to set off a variety of interesting dynamics in this range. The step was 0.4 and the iteration depth was 3, combining for a minimum parameter step of 0.1. The optimization function simply applied `touches-in-order?` to the trajectory and the tiebreaker function computed and maximized the fractal dimension *within the cells that cover the capture range.*

The mapper constructed and classified the following portraits, in this order:

Figure 7.34: Chaotic phase-locked loop attractors at various drive frequencies

Depth	Ω	**Classification**	**Interpretation**
1	.1	periodic	period-one
1	.5	chaotic	touches the capture range
2	.3	chaotic	far above the capture range
3	.2	periodic	period-one
3	.4	periodic	period-one
1	.9	periodic	period-one
2	.7	chaotic	touches the capture range
3	.8	chaotic	below capture range
1	1.2	periodic	period-one
1	1.5	periodic	period-one

Note that the chaotic and periodic zones are interleaved on the parameter space. The four chaotic attractors are shown on figure 7.34. The axes are the same as figure 7.32, but the cylinder is unrolled differently — the horizontal axis is $[0, 2\pi]$. The various periodic orbits would appear as points on these sections. The 0.3 attractor is qualitatively different from the other three; the latter can be deformed smoothly into one another. Because of the periodic regime at 0.4, it can be inferred that the system underwent at least two bifurcations as the frequency was raised from 0.3 to 0.5. Note that *Perfect Moment* only found this periodic window because the 0.3 attractor did not touch the capture range and the 0.5 attractor did; the *cause* of this was the large, bifurcation-induced change in the attractor's area. This discernment on the program's part is a good sign: it indicates that the dynamics classification algorithm may be better at finding hidden features than was predicted on page 59.

The search region parameters and the longer integrations did succeed in making the classifications very accurate, but they also made the run very slow. Each invocation takes most of a weekend to run.

The capture range is enclosed in the two half-rectangles at the crossing points of the axis lines on the figure; their range on the VCO-voltage axis is 2 normalized units, about one-twelth of the vertical range of the graph. The $\Omega = .5$ and $\Omega = .7$ attractors both touched these ranges. *Perfect Moment* chose the former because the fractal dimension in the destination endpoint cell — split in half on this plot — was larger. Note that this metric also rules out transients and periodic orbits. Incidentally, the tiebreaker function performed almost all of the work, as the first test was simplistic and almost all trajectories passed. No tolerance checking or additional segments were required. The former was programmed into the optimization function. The latter are moot because the problem was not set up that way: this example, in direct contrast to the pendulum, exercises the mapper to the exclusion of the path finder.

This development is by no means a universal solution to phase-locked loop design. The capture range is only reachable if a chaotic attractor that covers the appropriate area can be found for a particular reference frequency. Moreover, the initial condition must lie in the basin of that attractor. Success depends on the value chosen for the drive amplitude as well; a two-dimensional parameter-space search would certainly be better, but is not implemented here — see section 8.2.

7.4 Summary and Applications to Reality

Controlling the Lorenz system is unrealistic for a variety of reasons, foremost of which is finding one: no known physical system is exactly modeled by equations (7.1). The actuator that varies r is less of a problem than it would appear because of recent advances in research on electrorheological fluids, whose viscosity responds to an applied voltage. These fluids are used in electrically-controlled shock absorbers and computer-controlled exercise equipment; some respond as quickly as 1 millisecond, but require several thousand volts of drive[18]. Experimental computation of the Jacobian and sensitivity derivatives in this system would be made less difficult by its continuous/PDE nature, which makes it more likely that the particular state that the program wishes to recreate, vary and observe exists *somewhere* in the fluid. Note that a large-enough error will destroy the targeting improvement between figure 7.14 and figure 7.4 but will not destroy the overall control. The scheme is then reduced to that of [59] — interesting regardless, but nothing new. This sensitivity does *not* hold for the fixed point of section 7.1.2, where the gains should be much more immune to perturbations. All problems aside, this example is a classic and interesting paradigm in the field. It exhibits all of the properties that *Perfect Moment* is designed to exploit and presents none of the state-variable associated difficulties discussed in the last paragraph of section 4.2.2.

From a practical point of view, the other two examples in this chapter have more practical potential and value, but also present several practical difficulties, many of which could be addressed by the extensions of the junction controller that are presented at the end of section 6.1.

The pendulum's main problem is the slop in the drive system: the controller board's inaccuracy, the motor's dynamics (coast-down, overshoot, etc) and the backlash in the linkage (the gap between the cam follower and the slot in which it runs, etc.) Among other things, these problems may invalidate the assumption that the control parameter can be controlled to high accuracy (though it *can* change much faster than the the system's time scales.) Modeling is, of course, an issue: I dropped the motor one day, one of the gear teeth broke, the rotational motion was no longer smooth, causing a *visible* change in the system's dynamics. Similarly,

[18] I am indebted to Karon Maclean, John Morrell and Erik Vaaler for this information.

the (unshielded) bearings acquired foreign matter over the course of a year or two and the pendulum's Q decreased noticeably. In fact, the assumption that the damping is linear and viscous may well be inaccurate; the offset-linear damping in the bearings is probably much larger than the air friction, so equation 7.2 would be a better model if the βv ($\beta l\dot{\theta}$) term were replaced by $\beta_0 + \beta_1 v$ ($\beta_0 \approx \beta_2$.) These various structural and parametric uncertainties — and their potentially-sweeping effects — raise serious doubts about *Perfect Moment*'s ability to respond to changes in a target system. In this system, recreating state-space points and determining the derivatives experimentally is not difficult at all, as the reference trajectory grows forward from the control origin and each junction is *a priori* reachable.

The phase-locked loop has all of these problems, but in mitigated form. Electronic technology is more accurate than mechanical technology and no transduction between the two realms is necessary to communicate with a computer. Also, circuits don't change over time and environment quite as quickly as mechanical devices, thanks in part to decades of milspecs. This is particularly true of *digital* circuitry; a digital version of this phase-locked loop circuit would perhaps be a better test case for *Perfect Moment* because the modeling problems that it presents are less severe. The circuit's speed may pose a problem that is absent in the pendulum: the ratio of computer response time to device time constant is much smaller. Experimental derivative computations would not present serious problems in this application: for the same reason described at the end of the previous paragraph and because only one segment — a strange attractor at that, where shadowing renders the issue all but moot — is used. This circuit was constructed, but never demonstrated the advertized behavior, at least partially due to errors in the schematic in [27]. Once again, this drives home the sensitivity issue. If the circuit is chaotic for one resistor value and periodic when the value is 2% higher — let alone if one semiconductor is replaced by another — transferring all of the sterile mathematics involved in *Perfect Moment*'s gains to real systems is likely to be a difficult task.

The recurring theme here is nonlinear error amplification: in parameter, state, or model. These problems are inherent and fundamental. Paradoxically, nonlinearity and sensitivity that cause these problems are the very sources of *Perfect Moment*'s power, so solving them is very much worth whatever brain, computer and hardware effort it requires. Some possible solutions to these problems are proposed in the next chapter.

Chapter 8

Caveats, Lessons Learned, and Future Directions

A wide variety of shortcomings and problems are obvious in these algorithms. Some arise from the inherent properties of nonlinearity and are truly intractable. Problems of this form place fundamental restrictions on the domain and performance of the program and are discussed at the beginning of section 8.1. Problems that appear difficult but whose solutions probably only require more knowledge than the author possesses appear at the end of that section.

A long list of solutions to *Perfect Moment*'s distributed shortsightedness and various failure modes arose while this thesis was being written; a distilled and interpreted version of this list appears in section 8.2. Much of the discussion therein extends and generalizes the short, low-level suggestions that appear throughout this thesis, each in conjunction with the description of the algorithm step in which it arises.

8.1 Hard Problems

For these algorithms to work, the system state must be observable. If state variables are not directly accessible and information about the system state cannot be synthesized from outputs and other accessible signals (via "observers" — dynamical systems whose state variables are estimates of the state variables of another system[20, 32]), the system cannot be controlled using this — or any other — approach.

Secondly, if no known control scheme, linear or nonlinear, can stabilize the system in a satisfactory manner in the tolerance region surrounding the destination,

all of the machinery described in this thesis is moot[1].

Though the control parameters add dimensions to the space that can open conduits between previously-unreachable regions, some destinations may still be unreachable. The following rules describe *Perfect Moment*'s reachability conditions:

1. A valid attractor touches the destination and the origin is in its basin
2. The origin and destination are both in the same basin and the latter is near the trajectory from the origin to the attractor

   The following recursive extensions make this applicable to multi-segment paths:

3. The origin is in the basin of an attractor that has at least one point that satisfies condition 1 (i.e., it is in the basin of another valid attractor, at a different parameter value, that touches the destination)
4. The origin is in the basin of an attractor that has at least one point that satisfies condition 2

   ad infinitum

Whether or not *Perfect Moment* actually *finds* the trajectories described by items 2 and 4 in this list depends on the initial conditions — via R, m, and the rules that manipulate them — as well as on the nonlinear dynamics. Note that the higher the dimension of the attractor, the higher the probability that one of these four conditions will be satisfied.

As discussed in chapter 6, small control tolerances are fundamentally unachievable in the face of large Lyapunov exponents, inaccurate computer arithmetic, slow I/O or bad models. Slight timing or parameter value errors (e.g., quantization error) can be magnified exponentially, particularly if they occur at the beginning of a long segment or in a state-space region where either the vector field or the sensitivity derivative is large or is changing rapidly. This task is currently left to the junction controller, but could also be addressed in the planning algorithm. One could program these ideas into the optimization functions, forcing the program to choose paths in tamer areas. However, this modification would negate one of *Perfect Moment*'s primary goals: the *active use* of those less-tame areas. The search algorithm could also be altered to create paths with shorter segments, forcing the junction controller into a larger role, but this approach would subvert the planning philosophy that allows *Perfect Moment* to make counterintuitive moves and use short path segments to reach long, useful ones. In the limit where the ranges

[1]only because the goal state cannot be sustained, not because the program cannot find a path that gets there.

touch, incidentally, this forced use of shorter segments would reduce to Kalman's local-view-of-the-world scheme. One obvious solution to modeling problems is to gather data from systems and not simulations of systems. Experimental error, however, limits resolution and accuracy and can undermine any gains that might be obtained in this fashion.

The program currently handles systems that have a single control parameter. As mentioned in section 2.3, the coding involved in the generalization of this would be easy, but a higher-dimensional parameter space would slow the mapping and search processes and might make the mathematics somewhat different. Many of the structure theorems in nonlinear dynamics, for example, depend on dimension, and an additional, dynamically-varying control parameter increases the dimension of both the system *and* the search space. The number of control parameters typically plays a critical role in controllability calculations. In [59], for example, the number of control parameters required to assure controllability is equal to the number of unstable eigenvalues of the unstable periodic orbit to be stabilized. I have not investigated computation of these kinds of results for *Perfect Moment*.

The problem in the previous paragraph is a specific instance of a general lack of mathematical rigor. The design rules that manage the coarseness of the mapping and search make it impossible to prove many truly global and general results about what types of systems are particularly receptive or inimical to *Perfect Moment*'s approach, whether or not the paths that it finds are optimal, how long a given run will take, etc. Despite these problems with rigorous, satisfying mathematics — or perhaps *because* of the back-of-the-envelope engineering assumptions that are the underlying cause— this program, even in its current rough state, is a practical design tool.

8.2 Easier Problems

Many of the problems that were raised in the text could be solved with better integrations (longer; smaller timestep) and more generous bounds and design choices (e.g., more than five orbits to establish a periodic orbit.) The dynamics would be truer to the system, the classifications thereof would be better, locations of bifurcations would be more precisely explored, etc etc. All of these affect how close the path found by the program is to the true optimum, but all are computationally expensive — some exponentially so. The current choices define a particular balance of this tradeoff; different design requirements (e.g., control of something like a nuclear plant, where reliability and optimality are truly worth months of CPU time) would dictate a different set of choices.

Even with these approximations, *Perfect Moment* is slow: a typical run takes several hours on a state-of-the-art RISC workstation. Real-time control was not

Series	Machine	Time
HP 7000/900	720	0.07517 sec
HP	350	0.77640 sec
HP	375	0.25720 sec
Toolkit		0.11708 sec

Table 8.1: Comparisons of a 5000-step integration of the driven pendulum equations (interpreted, unoptimized Scheme code) on several different computing platforms

a goal of this thesis. The program is intended to find paths that will be followed many times or paths that are particularly important. For these reasons, only one round of serious effort was devoted to improving the speed. The mapping task was ported[2] to the Supercomputer Toolkit[2], without much success; see table 8.1 for comparisons to Hewlett-Packard series-300 and series-700 workstations. Of the three types of machines, the latter performed the fastest. The Toolkit is better at longer integrations that require fewer interventions, as communication and setup are expensive, and *Perfect Moment*'s tasks are too small and interactive to use it efficiently. The obvious next steps are to tighten up the code, use memory more efficiently, rewrite the entire program in a faster language than Scheme, and/or port it to a parallel computer. The obvious opportunities for parallelization are the cell-level concurrency in each mapping pass and the 2^n branches on the tree of path-finder calls.

A rich and interesting area for improvement is the nonlinear dynamics knowledge that the program encodes. For example, basin boundaries could be identified and used to limit mapping and search, or even to establish or rule out the reachability of a particular point. A related scheme is discussed in [85], where basin boundaries are used to group trajectories into equivalence classes, drastically reducing the number of data objects that must be manipulated and tested during the course of a search. One could also improve *Perfect Moment*'s IQ by adding to or modifying the entries in the table of conditions that govern when the inter-portrait parameter spacing is reduced (page 57,) perhaps taking into account the reachability conditions on page 132. One could also allow the *amount* of that reduction to adapt to the dynamics in some useful way, rather than its current binary search division.

Perfect Moment's knowledge about control theory could also be vastly improved. For example, no attempt is made here to use variations and sensitivity derivatives to search between existing trajectories and portraits and find the true optimum. The information that the mapper constructs is simply a rough cut at a

[2]The adaptive Runge-Kutta integrator for the Toolkit was written by Sasha Ferguson.

representative sampling of the state and parameter spaces. This choice is an engineering design tradeoff. Currently, if one wants a better optimum, one must specify a finer resolution: via m, Δk, and D_i. A powerful and useful modification would be to allow extrapolation in the state or parameter range between two trajectories that straddle an objective. Such a scheme would require that the tiebreaker functions be more intelligent (perhaps manipulating the metric differently,) that the mapper perform some extra analysis, and that a slight change in the control flow of the program be made to allow the extra levels of interaction between the path finder and the mapper that would be necessary for the extrapolation.

Parameter changes are not assumed to be small — in direct contrast to many classical techniques (and [59]) that judge success at least partially according to the size of the control parameter variation required to attain the objectives. In some sense, all that really matters is the energy delivered by the controller. With the same control-flow modification suggested in the previous paragraph, a minimum-energy constraint could be programmed into the optimality procedures, but it might severely restrict the program's available choices. More intelligent programmed constraints could also address the issue of speed of parameter changes. Currently, if the actuator cannot respond quickly enough, the reference trajectory overshoots the segment junction and exceeds the linear controller's range. Smaller Δks would reduce this problem and could also be encouraged by appropriate weights in the optimization functions.

A potentially-serious problem is hidden in the pendulum example. For the local controller to work when the pendulum arrives at the inverted point, the segmented path *must leave the drive arm in the right place.* Fortunately, the "right place" is a large region. The best places to leave it are at $\pm\frac{\pi}{2}$, where the motor has the most leverage. The only places where control will fail completely are at the ends of the range, where the local-linear controller has no leverage at all if the pendulum starts to fall the wrong way. The global-scale solution to this problem is very interesting: one could make the tolerance check a user-specified Scheme function as well (i.e., a path only meets tolerance if it ends near the destination *and* some condition on the drive phase is satisfied.) Note that this problem cannot just be solved by a better optimization function, as this condition only applies at the control destination and not at the partial path endpoints. Note, too, that *none* of these techniques would be possible if procedures could not be passed around as first-class procedure objects.

Finally, improvements are possible in the techniques herein that are traditionally defined as part of AI. The search could be organized better, perhaps by presorting the trajectories according to region; the multiple-scales algorithm would benefit from better organization as well. Rather than the interstate highway route-planning paradigm, with one long segment and a tree of recursively-constructed shorter segments that connect to it, a different planning algorithm could be used. One could perform the standard trick and split the problem into two subproblems,

aiming instead for a point between the origin and the destination, and derive the standard $O(n \log n)$ performance gain. Each time the path is split, a choice would have to be made about the location of the halfway point; traditionally, the midpoint of the segment joining the two endpoints is chosen. Like the modification of the optimization function to avoid bifurcation regions, this modification would conflict with *Perfect Moment*'s original goals: it would confine its choices and restrict its use of any interesting and useful dynamics that lie away from this path. This conflict could be partially circumvented by reintroducing the physics into the process, using nonlinear dynamics and partial path results to guide the choice of the halfway point. In the limit, this approach would produce an odd amalgam: a piecewise *non*linear control scheme. Another — related — planning algorithm would be to intersect all of the trajectories on a set of portraits and determine if any *combination* would be better than the single long segment that is currently sought out and used.

Chapter 9

Conclusion

In the most general terms, the implications of this research are that:

- A broader view and understanding of chaotic state-space features and the effects of parameters upon these features is a powerful tool, but its application requires great computational effort
- Understood and controlled, chaotic behavior can be profitably used to improve a system's design and performance

These observations are encouraging, considering that much of nature — hence most of the systems that people consider interesting, useful and in need of control — is chaotic.

Domain knowledge is to an AI program as education and prejudices are to a human: the fundamental guiding principles behind its behavior. *Perfect Moment* extends beyond the set of accepted design precepts and techniques of traditional nonlinear control in that it not only allows but actually encourages chaos. This concept, perhaps more than the individual results in chapter 7 or the algorithms in chapters 4, 5 and 6, is the truly novel contribution of this thesis.

A chaotic system's behavior, and thus its state-space features, are strongly affected by parameter values. Moreover, the trajectories that make up those features separate exponentially over time. Small changes in parameters or in state can thus have large and rapid effects; this leverage can be a powerful tool for a control algorithm. This thesis is a detailed description and demonstration of how fast and accurate computation can be used to synthesize paths through a chaotic system's state space that exploit this leverage to accomplish otherwise-impossible control tasks. Nonlinear dynamics provides the mathematical tools used by these algorithms to choose values, tolerances, heuristics and limits for the selection and synthesis of trajectory segments into segmented reference trajectories. *Perfect Mo-*

ment can find trajectories that are shorter and faster than those found by traditional control methods, make unreachable control objectives reachable and improve convergence. It does so by constructing strange attractor bridges, making counterintuitive moves, and utilizing regions of sensitive dependence. This performance gain does, of course, have a cost: the complexity of the tasks that are executed by the program and the accuracy demanded by the very sensitivity that gives the program its power make computation speed a vital issue. *Perfect Moment* does not always outperform classical linear and nonlinear control techniques. In fact, the program sometimes fails to find a path at all in a problem that is easily solved by the standard techniques. The converse is also true, however, making this type of technique a useful addition to the arsenal of control techniques.

The examples presented here demonstrate a variety of interesting uses of nonlinear and chaotic behavior:

- Targeting: a small control action at the "perfect moment" can accurately direct a trajectory to a distant state-space objective
- The program's planning algorithms allow it to synthesize reference trajectories that exploit both global and local dynamics (e.g., counterintuitive moves, pumping up the pendulum with a small torque, etc)
- The fractal dimension of a chaotic attractor can improve reachability and the unstable orbits embedded densely in such attractors can be located, targeted and stabilized
- Strange attractor bridges can allow trajectories to cross basin boundaries or other previously-insurmountable obstacles, improving both local and global convergence

This approach can be thought of as a new flavor of adaptive control — one that takes a global viewpoint and eschews almost all linearization. It extends the active use of chaos in control, which is currently very limited. Its AI flavor distinguishes it from classical control, particularly in its use of approximations and simplifying assumptions to balance design time against quality of the results. Because of these approximations, *Perfect Moment* has a completely different set of failure modes than traditional control methods — ones that make formal proofs difficult. Another significant deviation from the classical control paradigm is that parameter changes are *not* restricted in size here; this reflects the goal of finding and using — on a global scale — a wide variety of interesting dynamics.

Perfect Moment uses methods from several areas of science and engineering in its analysis of the target system and synthesis of the reference trajectory. None of the individual methods are considered — in their own fields — to be radically-new

technology[1], but the hybridization is very powerful. A few of the more important paradigms that make this combination possible are:

- Programs that work with the output of other programs (e.g., the interleaved mapper and path finder calls)
- Combined numerical and symbolic methods (e.g., a numerical integrator operating on a symbolically-differentiated function)
- Combined algebraic and geometric methods (e.g., the inter-step cell size reduction and the multiple-scales dynamics classification scheme that is based on that cell size)
- Linear control used to perform local corrections; global, high-cost control used to do large-scale planning
- Domain knowledge from both control theory and nonlinear dynamics used to make intelligent estimates and choices, improving both the results and the performance of the program as it constructs those results

Several nagging problems resist even this broad-based approach: modeling, numerical, and experimental errors — magnified by the very nonlinear amplification that give the program its power — and their effects on robustness. Modeling error is particularly pernicious, as it can quickly and easily negate all the gains derived by this program. In light of the difficulty of modeling systems to the accuracy demanded by the nonlinear amplification, this is a serious and imposing problem.

Though these programs have been successfully tested on simulated models of several systems, the ultimate goal of this project is the control of real physical devices. Versions of the driven-vertex pendulum and the phase-locked loop discussed in chapter 7 have been constructed and are in the midst of being instrumented for control. A double pendulum — an autonomously chaotic system wherein bifurcations are affected by damping, the ratio of the moments of the two bobs, etc — has also been constructed (see figure 9.1,) but its instrumentation is much more difficult because of the spatial rotation of the second pivot point and is proceeding more slowly[2]. Because of the problems outlined in section 7.4 and chapter 8, much effort will be required in the areas of system identification and observation — modeling error, sensor and actuator inaccuracy due to D/A and A/D conversion and time delay, etc — before the gains demonstrated in these simulations can be fully realized in the corresponding physical systems. A first step along this path, currently in process, is to obtain experimental verification of the state-space

[1]although several, notably the algorithms at the end of section 6.2, were developed in the past few years

[2]One —rather macabre — reference to controlling a double pendulum appears in the final scene of Umberto Eco's novel *Foucault's Pendulum* (Harcourt Brace Jovanovich, 1989.)

Figure 9.1: Photographs of the double pendulum

data given in chapter 7 using actuators, sensors, and computer I/O — rather than Runge-Kutta — to explore the target system's behavior. In addition to its positive effects on modeling error, this approach would probably be much faster than computer simulations, at least for the phase-locked loop. Experimental error, of course, would be a problem — similar to numerical error. Beyond instrumentation, a wealth of other improvements are possible, ranging from different planning algorithms, parallelization, use of different properties of chaos, and perhaps to better automation of the few initial choices required from the user. Immplementation of these changes would cause *Perfect Moment* to evolve from a design assistant to the equivalent of a very junior engineer.

The complexity of the tasks that are executed by this control program and the accuracy with which it must perform make it extremely complex and slow. All of this work is worth it, however; allowing a system to operate in its chaotic regimes creates new possibilities for better designs. Faster computers might be part of the ultimate solution, together with the understanding and algorithms gained in the course of this research, to attaining novel and effective control of useful systems via knowledge of nonlinear theory and intensive computation.

Appendix A

How to use Perfect Moment

A.1 Setting up the run

The scenario in chapter 2 gives a good example of how to run *Perfect Moment*; please read it first.

The first task is to write a system derivative procedure that takes a state n-vector and a time n-vector and produces an n-vector of derivatives at that point:

```
; *parameter* plays the role of the damping
(define (variable-damping-pend state t)
  (let ((theta (car state))
        (omega (cadr state)))
       (list omega
          (* -1
             (+ (* (sin theta) (+ (/ g *pendulum-length*)
                                  (/ (* *drive-amp*
                                        (sqr *drive-freq*)
                                        (sin (* *drive-freq* (car t))))
                                     *pendulum-length*)))
                (* *parameter* (/ omega *pendulum-mass*)))))))

(variable-damping-pend '(1 0) '(.1 .1))
;Value 3: (0 -54.97610434078257)
```

See `rossler` in the scenario or `lorenz` in figure 5.14 for other examples. The global variable `*parameter*` plays a special role: it is assumed to be the (single) control parameter.

To invoke the program, evaluate `(perfect-moment)`. You will be asked for the following information:

- Origin and destination: the n-vectors representing the specified and ending points
- Optimization function: a function which computes optimality weight from a discretized trajectory. Use `shortest-path-between-two-cells` or refer to section A.2 for instructions on writing your own
- Tiebreaker function: a function which computes optimality weight from a *non*discretized trajectory. Use `minimum-sum-distance-tiebreaker` or refer to section A.2 for instructions on writing your own
- Tolerance: this is currently an n-vector of percentages (dotted with the destination vector)
- Overrange factor (default = 2): use the default unless you know that the interesting dynamics occur on a larger scale
- Initial grid division (default = 5 cells/side): this is a reasonable choice for systems of less than four dimensions, unless you want to use the graphics
- Parameter range: this should reflect your actuator's physical range
- Parameter step: this defines the top-level grain of the search
- Iteration depth (default = 5): choose the default unless you get impatient or unless you don't want the program to zero in on the parameter scale.

  Make sure that the combination of range, step and depth don't overextend your actuator.
- Monitoring the run (g[raphics]/d[escriptive]/n[o]): selecting "g" can be interesting and useful; it can also be very confusing, particularly if n or m is high, and always slows the program down significantly.

  If "g" is chosen, the program will ask for display information; specify the two state-variables that define the projection that you wish to observe. If you do want to watch what's going on, use a smaller m.

  I rarely, if ever, use "n."

Typically, one would use the default R, a small m (e.g., 2), an iteration depth of 2, and watch the run's graphical output. Once assured that all dynamics are in bounds, one might then restart the run with a larger m. If truncated structures are recognizable on the graphics screen, one would repeat the initial run with a larger R. Recall, however, that m must be increased accordingly to preserve the same accuracy. If the aspect ratios of the bounding boxes of the attractors and the objectives are very different, one can explicitly enter two n-vectors instead of an R value; see section 7.1 for an example of a situation where such a choice is a good

idea. If the structures are clearly recognizable and fully contained in the search region, but do not set off the dynamics classification algorithm, the run requires a larger m.

Beyond this, the next step is to change the source code or the system.

Many potentially-useful code changes can be effected at the top level, inside the `perfect-moment` function itself. A fixed (smaller or larger) timestep, rather than the one determined by the adaptive integrator run, can be forced in the `let` statement. The length of the integration can easily be changed (the variable `points`.) Often a subtle alteration in an optimization or tiebreaker function will do the trick. Other changes can be made in the dynamics classification or search region refinement code, via the appropriate global variable that is examined by the functions `attractor-type, refine-region` and `refine-intervals`:

- `*orbits*`: the number of pattern repeats required to establish periodicity (currently five)
- `*transient-length*`: the fraction of the trajectory that is regarded as transient (currently 0.5)
- `*variation-scaling*`: the scaling of the difference vector for purposes of the variational integration (currently m^2)
- `*turbulence-threshold*`: the variation threshold that determines when the cell size is reduced (currently 1)
- `*m-expansion*`: the amount by which the grid size is reduced if the search fails (currently 1.5)

Changes in the initial condition for the variational integration used to refine the cell size can be made inside `refine-intervals` (e.g., using the origin rather than the cell centerpoint.) See chapters 4 and 5 for more details on all of these quantities.

The last recourse is to change the target system. This requires engineering, modeling, and programming skills. Perhaps the easiest way is to buy or build a better actuator — with a wider range or better resolution — and adjust the range and step accordingly. One would write a new system derivative, using `*parameter*` in a different coefficient, *in a physically-meaningful way*. Examples are: substituting a voltage- or current-controlled electronic component for a fixed-value one, adding a magnetic brake to a pendulum, etc. If parametric changes made to existing coefficients still do not give *Perfect Moment* enough leverage to find a path, structural changes may work (e.g., another input to an op amp, another sensor fed back to the pendulum controller, etc.)

Once the path is found, you can test it on physical hardware — if implemented and connected — simulate it, or quit the program ("h," "s" or "q," respectively.)

A.2 Specifying different optimization functions

This section presents the actual code for the optimization functions used in this thesis.

`(shortest-path-between-two-cells cells)`

returns a procedure that is applied to a trajectory and returns an evaluation. `minimum-sum-distance-tiebreaker` returns a function that takes a list of `evaluations` and returns the best one.

```
(define (shortest-path-between-two-cells cells)

(define (shortest-path traj from-cell to-cell best-yet)
   (let
     ((next-piece (rest-after-inclusive traj from-cell))
      (current-count (cells-between traj from-cell to-cell)))
     (if (null? next-piece) best-yet
         (shortest-path
             (cdr next-piece)
             from-cell to-cell
             (cond ((not (number? current-count)) best-yet)
                   ((> (get-metric best-yet) current-count)
                    (make-evaluation current-count
                                     next-piece
                                     'trash))
                   (else best-yet))))))

   (let
     ((from-cell (car cells))
      (to-cell (cadr cells)))
     (lambda (trajectory)
        (if (equal? from-cell to-cell)
    (make-evaluation -1
            ; the -1 signals that the points lie in the same square
                        (rest-after-inclusive trajectory from-cell)
                        trajectory)
            (let* ((partial-eval
                    (shortest-path
                      (rest-after-inclusive trajectory from-cell)
                      from-cell to-cell *starting-case*))
                   (metric (car partial-eval))
                   (int-val (cadr partial-eval)))
              (make-evaluation metric int-val trajectory))))))
```

```
(define (minimum-sum-distance-tiebreaker
            fcn points
            low high intervals
            special-points
            param-value magfactor)

  (define (inner set-of-evaluations current-best)
    (if (null? set-of-evaluations) current-best
(let ((this-eval (car set-of-evaluations)))
  (if (equal? this-eval *starting-case*)
      (make-evaluation
       (get-metric *starting-case*)
       'no-path-found
       'param-goes-here)
      (let*
  ((min-dist-to-orig
    (find-minimum-distance
     (reconstitute-trajectory (get-trajectory this-eval)
      special-points low high intervals
      fcn points param-value magfactor)
     (get-origin special-points)
     *starting-case*))
   (min-dist-to-dest
    (find-minimum-distance
     (reconstitute-trajectory (get-trajectory this-eval)
      special-points low high intervals
      fcn points param-value magfactor)
     (get-destination special-points)
     *starting-case*))
   (this-metric (+ (get-metric min-dist-to-orig)
   (get-metric min-dist-to-dest))))
(inner (cdr set-of-evaluations)
       (if (< this-metric
      (get-metric current-best))
   (make-evaluation this-metric
    (list min-dist-to-orig
  min-dist-to-dest)
    'endpoints)
   current-best)))))))

(lambda (evals) (inner evals *starting-case*)))
```

Some other suggestions:

- `fastest-time-optimizer/tiebreaker`: subtract the timestamps of the cells or points involved
- `path-length-tiebreaker`: the pointwise analog to `shortest-path-between two-cells`
- `minimum-energy-optimizer/tiebreaker`: compute the energy required from the controller to execute the segment
- `minimum-maximum-acceleration-optimizer/tiebreaker`: compute the peak acceleration on each segment
- `minimum-maximum-jerk-optimizer/tiebreaker`: compute the peak jerk on each segment

All optimization functions operate on discretized trajectories and compute with cells. All tiebreaker functions operate on state-space points.

Appendix B

The Phase-Locked Loop

The phase-locked loop circuit in figure B.1 was used by D'Humieres *et al*[27]; they state that it was "described by Bak[8] ...[and] patterned after that developed by Henry and Prober[38]." The schematic in figure B.1 follows the former[1]. The former paper gives a much better description of the nonlinear dynamics of equation 7.2 and its implications; the latter is a better reference for anyone who wants to really understand the electronics.

The VCO (2206) output is mixed with the 100kHz reference signal V_R, filtered to remove harmonics, then fed back (I_{BF}) to the VCO input via an opamp that performs loop compensation and allows the external modulating input V_E to be added in. Typical values for the compensation components R and C are given as 11KΩ and 0.1μF. The sampling and filtering operations are combined in a 398 sample-and-hold circuit in the manner discussed on page 122; the zero-crossings of the frequency response of this device must be tuned (the 1K pot and 10K resistor at the bottom of the figure) to hit these harmonics exactly. The VCO's initial freqeuncy range is tuned by the 1K pot and 6.8K resistor at pin 7. The 5K/100pF combination sets the one-shot pulsewidth — and hence the sampling window — to 1μsec; samples of this length are triggered by each rising edge on the VCO's output. The hold capacitor is .001μF, so leakage (via the impedance seen across pins 6 and 7 of the 398) is not a problem over the periods involved. The VCO gain factor[74] — the K_v in equation 7.10 — is 2000 Hz per volt.

Most of the mathematics (given in section 7.3) of squeezing equation 7.2 out of this circuit are straightforward. The exception is the samping: recall the statement "... recognizing that, because the sampler fires at the zero crossings of v_O, the sampled phase ($\omega_R \tau_n + \phi_0$) is the same as θ ..."

The reference voltage, $V_1 \sin(\omega_s t + \phi_0)$, sampled at time τ_n, is $V_1 \sin(\omega_s \tau_n + \phi_0)$,

[1]This diagram contains enough errors to make it impossible to reconstruct, so I have not been able to duplicate their experiment exactly.

variable	pendulum	phase-locked loop
θ	angular position	phase difference between oscillators
$\dot{\theta}$	angular velocity	kV
$mg\sin\theta$	restoring torque	feedback current
ml	inertial momentum	C/k
βl	viscous damping	1/kR
$\gamma(t)$ applied torque	applied current	V_E/R_E
$\Omega_0 = \sqrt{g/l}$	natural frequency	$\sqrt{kV_1/R_SC}$
$Q = \sqrt{m^2g/\beta^2 l}$	quality factor	$\sqrt{kV_1CR^2/R_S}$

Table B.1: Correspondences between the different coefficients

where $\omega_s = 2\pi \times 100kHz$. The next sample is taken at τ_{n+1}, where

$$\omega_s(\tau_{n+1} - \tau_n) + k\int_{\tau_n}^{\tau_{n+1}} V(t)\,dt = 2\pi$$

Substituting $\theta_n = (2n+1)\pi - \omega_s\tau_n - \phi_0$ in the equation above yields

$$\begin{aligned}\theta_{n+1} &= \theta_n + 2\pi - \omega_s(\tau_{n+1} - \tau_n) \\ &= \theta_n + k\int_{\tau_n}^{\tau_{n+1}} V(t)\,dt\end{aligned}$$

The next steps involve three (sequential) assumptions:

1. ω_s is larger than the highest frequency encountered on $V(t)$ between successive samples
2. Thus, $V(t)$ can be considered as constant
3. The time between the samples can be neglected

(1) and (2) imply that

$$V(t) = \lim_{\omega_s\to\infty}\left[\frac{1}{k}\left[\frac{\theta_{n+1} - \theta_n}{\tau_{n+1} - \tau_n}\right]\right] = \frac{\dot{\theta}}{k}$$

or $\dot{\theta} = k\,V(t)$ — the substitution that turns equation 7.11 into equation 7.12.

Appendix A of [27] also discusses a few interesting drawbacks that stem from the non-ideal dynamics of the sample-and-hold (its overshoot, in particular,) that introduce phase-dependent errors and hence asymmetries in I_{FB}. These authors explore a variety of DC biases to prove that the system's symmetry breaking is intrinsic and not a result of the simulator circuit's irregularity.

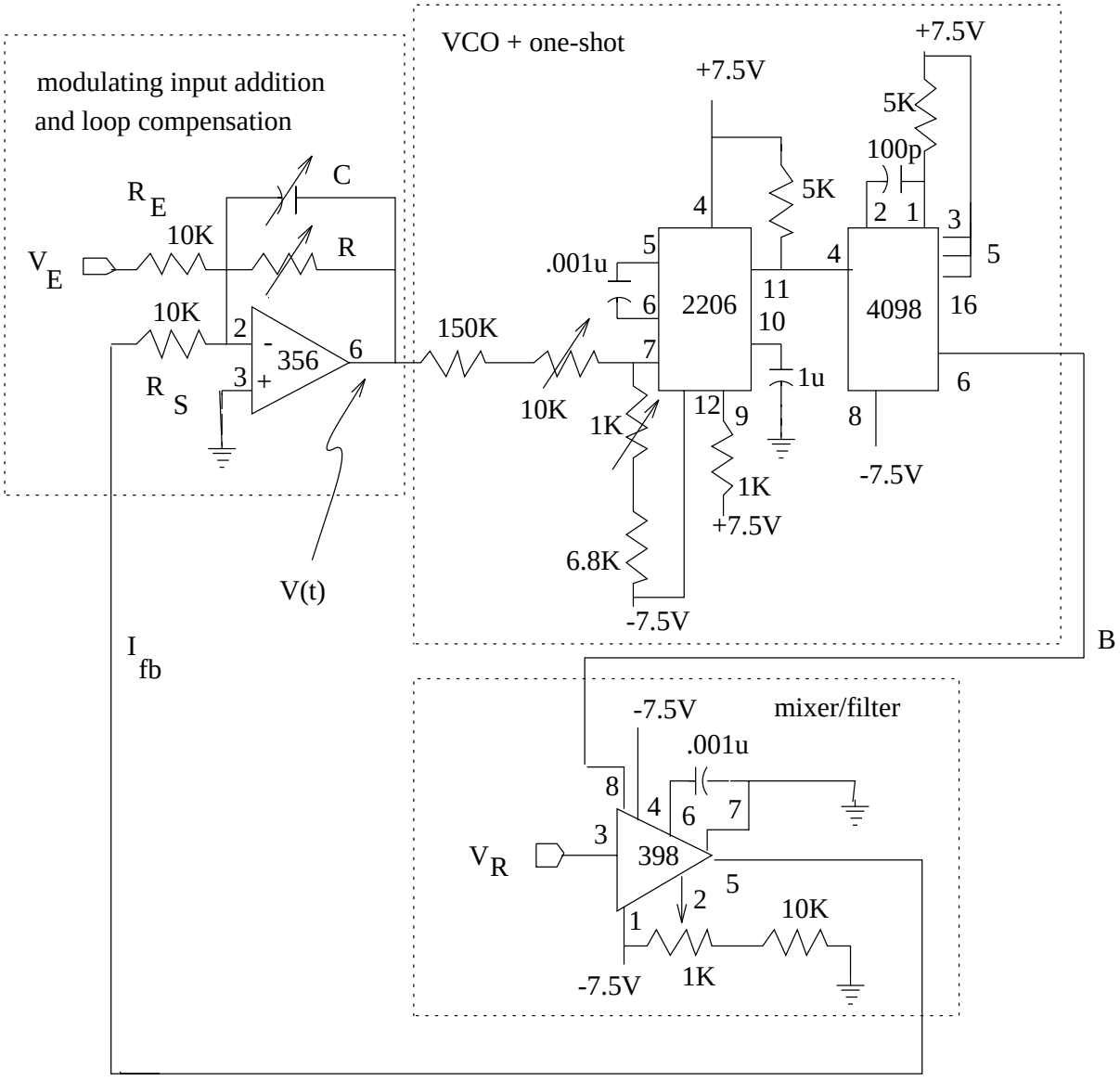


Figure B.1: Schematic of the phase-locked loop circuit

Bibliography

[1] H. Abelson. The Bifurcation Interpreter: A step towards the automatic analysis of dynamical systems. *International Journal of Computers and Mathematics with Applications*, 20:13, 1990.

[2] H. Abelson, A. A. Berlin, J. Katzenelson, W. H. McAllister, G. J. Rozas, and G. J. Sussman. The Supercomputer Toolkit and its applications. Technical Report AIM 1249, MIT Artificial Intelligence Lab, July 1990.

[3] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, G. J. Sussman, and K. Yip. Intelligence in scientific computing. *Communications of the ACM*, June 1989.

[4] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge MA, 1985. With Julie Sussman.

[5] R. H. Abraham and C. D. Shaw. *Dynamics — The Geometry of Behavior*. Addison Wesley, Redwood City CA, 1992. Second Edition.

[6] U. M. Ascher and L. R. Petzold. Stability of computational methods for constrained dynamics systems. Technical Report CICSR-TR92-001, University of British Columbia, October 1991.

[7] K. J. Astrom and B. Wittenmark. *Adaptive Control*. Addison Wesley, Redwood City CA, 1989.

[8] C. K. Bak and N. F. Pedersen. Josephson junction analog and quasiparticle-pair current. *Applied Physics Letters*, 22:149–150, 1973.

[9] G. L. Baker and J. P. Gollub. *Chaotic Dynamics: An Introduction*. Cambridge University Press, Cambridge U.K., 1990.

[10] M. Barnsley. *Fractals Everywhere*. Academic Press, Boston, 1988.

[11] R. Bellman and R. Kalaba. *Dynamic Programming and Modern Control Theory*. Academic Press, San Diego CA, 1965.

[12] G. Bernstein, M. A. Lieberman, and A. J. Lichtenberg. Nonlinear dynamics of a digital phase-locked loop. *IEEE Transactions on Communications*, 37:1062, 1989.

[13] S. P. Boyd and C. H. Barratt. *Linear Controller Design: Limits of Performance*. Prentice-Hall, Englewood Cliffs NJ, 1991.

[14] E. Bradley. Causes and effects of chaos. Technical Report AIM 1216, MIT Artificial Intelligence Lab, December 1990.

[15] E. Bradley. A control algorithm for chaotic physical systems. In *Proceedings of the First Experimental Chaos Conference*. World Scientific, 1991.

[16] E. Bradley. Control algorithms for chaotic systems. In G. Jacob and F. Lamnabhi-Lagarrigue, editors, *Lecture Notes in Control and Information Sciences*, volume 165, pages 307–325. Springer-Verlag, Paris, December 1991.

[17] E. Bradley and F. Zhao. Phase space control system design. *IEEE Control Systems Magazine*, 13:39–46, 1993.

[18] P. J. Bryant and J. W. Miles. On a periodically forced, weakly damped pendulum. Part I: Applied torque. *Journal of the Australian Mathematical Society*, 32:1–22, 1990.

[19] P. J. Bryant and J. W. Miles. On a periodically forced, weakly damped pendulum. Part III: Vertical forcing. *Journal of the Australian Mathematical Society*, 32:42–60, 1990.

[20] F. Celle, J. P. Gauthier, and G. Sallet. Synthesis of nonlinear observers: A harmonic analysis approach. In J. Descusse, editor, *New Trends in Nonlinear Control Theory*. Springer-Verlag, 1988.

[21] D. Claude. Everything you always wanted to know about linearization, but were afraid to ask. In M. Fliess and M. Hazewinkel, editors, *Algebraic and Geometric Methods in Nonlinear Control Theory*. D. Reidel, 1986.

[22] J. P. Crutchfield, J. D. Farmer, N. H. Packard, and R. S. Shaw. Chaos. *Scientific American*, 255:46–57, December 1986.

[23] M. de Sousa Vieira, A. J. Lichtenberg, and M. A. Lieberman. Nonlinear dynamics of self-synchronizing systems. *International Journal of Bifurcation and Chaos*, 1(3):1–9, 1991.

[24] D. F. Delchamps. Stabilizing a linear system with quantized state feedback. *IEEE Transactions on Automatic Control*, 35:916, 1990.

[25] J. Descusse. *New Trends in Nonlinear Control Theory: Proceedings of an International Conference on Nonlinear Systems.* Springer-Verlag, New York, 1988.

[26] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems.* Benjamin/Cummings, Menlo Park CA, 1986.

[27] D. D'Humieres, M. R. Beasley, B. Huberman, and A. Libchaber. Chaotic states and routes to chaos in the forced pendulum. *Physical Review A*, 26:3483–3496, 1982.

[28] W. L. Ditto, S. N. Rauseo, and M. L. Spano. Experimental control of chaos. *Physical Review Letters*, 65:3211, 1990.

[29] J.-P. Eckmann. Roads to turbulence in dissipative dynamical systems. *Reviews of Modern Physics*, 53:643–671, 1981.

[30] M. Eisenberg. *The Kineticist's Workbench: Combining Symbolic and Numerical Methods in the Simulation of Chemical Reaction Mechanisms.* PhD thesis, MIT Artificial Intelligence Lab, June 1991. Also available as AI-TR-1306.

[31] J. Ford and M. Ilg. Eigenfunctions, eigenvalues, and time evolution of finite, bounded, undriven quantum systems are not chaotic. *Physical Review A*, 45:6165–6173, 1992.

[32] B. Friedland. *Control System Design: An Introduction to State-Space Methods.* McGraw Hill, New York, 1986.

[33] K. S. Fu. Learning control systems and knowledge-based control systems: An intersection of artificial intelligence and automatic control. *IEEE Transactions on Automatic Control*, AC-16(1):70–72, 1971.

[34] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* Springer-Verlag, New York, 1983.

[35] J. Gullicksen, M. de Sousa Vieira, M. A. Lieberman, R. Sherman, A. J. Lichtenberg, J. Y. Huang, W. Wonchoba, M. Steinberg, and P. Khoury. Secure communications by synchronization to a chaotic signal. In *First Experimental Chaos Conference.* World Scientific, 1991.

[36] G. H. Gunaratne, P. S. Linsay, and M. J. Vinson. Chaos beyond onset: A comparison of theory and experiment. *Physical Review Letters*, 63:1, 1989.

[37] E. G. Gwinn and R. M. Westervelt. Fractal basin boundaries and intermittency in the driven damped pendulum. *Physical Review A*, 33:4143–4155, 1986.

[38] R. W. Henry and D. E. Prober. Electronic analogs of double-junction and single-junction SQUIDS. *Reviews of Scientific Instruments*, 52:902–914, 1981.

[39] B. M. Herbst and M. J. Ablowitz. Numerically induced chaos in the nonlinear Schrodinger equation. *Physical Review Letters*, 62:2065–2068, 1989.

[40] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, San Diego CA, 1974.

[41] C. S. Hsu. *Cell-to-Cell Mapping.* Springer-Verlag, New York, 1987.

[42] B. A. Huberman and E. Lumer. Dynamics of adaptive control systems. *IEEE Transactions on Circuits and Systems*, 37:547–550, 1990.

[43] E. S. Hung. Dynamics enhanced parameter estimation in chaotic systems. S.B. thesis, MIT Artificial Intelligence Lab, June 1991.

[44] IEEE. *IEEE Workshop on Intelligent Control*, 1985.

[45] IEEE. *IEEE Symposium on Computer-Aided Control System Design*, March 1992.

[46] M. Ilic and F. K. Mak. A new class of fast nonlinear voltage controllers and their impact on improved transmission capacity. *Automatic Control Conference*, 1989.

[47] E. A. Jackson. On the control of complex dynamic systems. *Physica D*, 50:341–366, 1991.

[48] R. E. Kalman. Phase-plane analysis of automatic control systems with nonlinear gain elements. *Transactions of the AIEE*, 73:383, 1955.

[49] L. Keen. Julia sets. In *Chaos and Fractals: The Mathematics Behind the Computer Graphics.* American Mathematical Society, 1988. Proceedings of Symposia in Applied Mathematics, volume 39.

[50] M. Lee. Summarizing qualitative behavior from measurements of nonlinear circuits. Technical Report AI-TR 1125, MIT Artificial Intelligence Lab, May 1989.

[51] A. J. Lichtenberg and M. A. Lieberman. *Regular and Stochastic Motion.* Springer-Verlag, New York, 1983.

[52] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963.

[53] L. Mackey and M. C. Glass. *From Clocks to Chaos: The Rhythms of Life.* Princeton University Press, Princeton NJ, 1988.

[54] B. B. Mandelbrot. *The Fractal Geometry of Nature.* Freeman, New York, 1983.

[55] I. M. Y. Mareels and R. R. Bitmead. Nonlinear dynamics in adaptive control: Chaotic and periodic stabilization. *Automatica*, 22:641, 1986.

[56] J. E. Marsden, O. M. O'Reilly, F. J. Wicklin, and B. W. Zombro. Symmetry, stability, geometric phases, and mechanical integrators. *Nonlinear Science Today*, 1:4–21, 1991.

[57] A. W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proceedings of the 8th International Workshop on Machine Learning*, 1991.

[58] F. Ohle, A. Hübler, and M. Welge. Adaptive control of chaotic systems. Technical Report CCSR-90-13, Beckman Institute/University of Illinois at Urbana/Champaign, 1990.

[59] E. Ott, C. Grebogi, and J. A. Yorke. Controlling chaos. In *Chaos: Proceedings of a Soviet-American Conference.* American Institute of Physics, 1990.

[60] T. S. Parker and L. O. Chua. *Practical Numerical Algorithms for Chaotic Systems.* Springer-Verlag, New York, 1989.

[61] L. M. Pecora and T. L. Carroll. Synchronization in chaotic systems. *Physical Review Letters*, 64:821–824, 1990.

[62] H.-O. Peitgen and P. H. Richter. *The Beauty of Fractals.* Springer-Verlag, New York, 1986.

[63] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge U.K., 1988.

[64] J. Rees and W. Clinger. The revised^3 report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 21:37, 1986.

[65] J. K. Roberge. *Operational Amplifiers: Theory and Practice.* Wiley, New York, 1975.

[66] O. E. Rossler. An equation for continuous chaos. *Phys. Lett.*, 57A:397–398, 1976.

[67] E. P. Sacks. Automatic qualitative analysis of ordinary differential equations using piecewise-linear approximations. Technical Report AI-TR 1031, MIT Artificial Intelligence Lab, March 1988.

[68] G. N. Saridis. Toward the realization of intelligent controls. *Proceedings of the IEEE*, 67(8):1115–1133, 1979.

[69] T. Shinbrot, E. Ott, C. Grebogi, and J. A. Yorke. Using chaos to direct trajectories to targets. *Physical Review Letters*, 65:3215–3218, 1990.

[70] A. Siapas. A global approach to parameter estimation of chaotic dynamical systems. Master's thesis, MIT Artificial Intelligence Lab, June 1992. also available as MIT AI Memo 1402.

[71] J. Singer, Y.-Z. Wang, and H. H. Bau. Controlling a chaotic system. *Physical Review Letters*, 66:1123–1126, 1991.

[72] J.-J. E. Slotine. Putting physics back in control. In J. Descusse, editor, *New Trends in Nonlinear Control Theory*. Springer-Verlag, 1988.

[73] J.-J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, Englewood Cliffs NJ, 1991.

[74] J. Smith. *Modern Communication Circuits*. McGraw-Hill, New York, 1986.

[75] C. Sparrow. The Lorenz equations. In H. Haken, editor, *Synergetics: A Workshop*, pages 111–134. Springer Verlag, 1977.

[76] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer Verlag, New York, 1992. Second Edition.

[77] G. J. Sussman and J. Wisdom. Chaotic evolution of the solar system. *Science*, 257:56–62, 1992.

[78] J. M. T. Thompson and H. B. Stewart. *Nonlinear Dynamics and Chaos*. Wiley, Chichester U.K., 1986.

[79] J. Vlach and K. Singhal. *Computer Methods for Circuit Analysis and Design*. Van Nostrand Reinhold, New York, 1983.

[80] D. S. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann, San Mateo CA, 1990.

[81] R. F. Williams. The structure of Lorenz attractors. *Publications Mathematiques de l'IHES*, 50:101, 1979.

[82] P. H. Winston. *Artificial Intelligence*. Addison Wesley, Redwood City CA, 1992. Third Edition.

[83] J. Wisdom and M. Holman. Symplectic maps for the N-body problem: Stability analysis. *Astronomical Journal*, 1992.

[84] K. M.-K. Yip. KAM: Automatic planning and interpretation of numerical experiments using geometrical methods. Technical Report AI-TR 1163, MIT Artificial Intelligence Lab, August 1989.

[85] F. Zhao. Extracting and representing qualitative behaviors of complex systems in phase spaces. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Australia, August 1991. Morgan Kaufmann.

[86] F. Zhao. Phase Space Navigator: Towards automating control synthesis in phase spaces for nonlinear control systems. In *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real Time Control*, 1991. Pergamon Press.

[87] G. Zhou and J. D. Birdwell. PID autotuner design using machine learning. In *IEEE Symposium on Computer-Aided Control System Design*, 1992.