Multi-Modal Control: From Motion Description Languages to Optimal Control

A Thesis
Presented to
The Academic Faculty

by

Florent C. Delmotte

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2006

Multi-Modal Control: From Motion Description Languages to Optimal Control

Approved by:

Dr. Magnus Egerstedt, Adviser
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Erik Verriest
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yorai Wardi
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Tucker Balch
College of Computing
Georgia Institute of Technology

Dr. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved : November 13th, 2006.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Professor Magnus Egerstedt, for his guidance, encouragement, and support at all levels. I would also like to thank Professors Erik Verriest and Yorai Wardi for serving on my reading committee. I had great pleasure working with both of them on numerous occasions, and I would like to thank them for the thoughtful discussions, "brainstorm sessions", and general guidance. I would also like to acknowledge Professors Aaron Lanterman and Tucker Balch for serving on my committee. Professor Tucker Balch also kindly provided me with useful ant data for my applications.

My research was funded by the National Science Foundation and DARPA, to which I am very grateful.

I would also like to thank my fellow graduate students for their friendship and for creating an enjoyable and stimulating work environment: Tejas Mehta, Henrik Axelsson, Meng Ji, Dave Wooden, Abubakr Muhammad, Shun-Ichi Azuma, Mohamed Babaali, Stafan Bj orkenstam, Mauro Boccadoro, Dennis Ding, Johan Isaksson, Brian Smith, Eric Innis, Patrick Martin, Deryck Yeung.

Thank you also to all the friends that I made here in Atlanta. It would be too long to list them all, but I cherish the friendship of each of them.

Above all, I wish to thank all of my family, especially my parents, brother, and girlfriend for their love and continuous support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

A *multi-modal system*, as its name suggests, is a system that can switch between different modes of operation. Since multi-modal systems combine time-driven dynamics (the differential equations describing each mode) and event-driven dynamics (the discrete switching laws dictating the transitions between these modes), they belong to the wider class of *hybrid systems*. Examples of multi-modal systems, also referred as *switched systems*, include electric circuits with transistors, car gearboxes, hybrid motors, and chemical processes.
Inspired from these systems, *multi-modal control* has emerged as a successful tool for dealing with systems with high complexity (systems that we would not necessarily classify as multi-modal at first sight, e.g., autonomous robots). The control consists of 1) defining modes of operations for such systems and 2) providing a discrete switching policy between these modes. In this context, a typical control program would consist of a sequence of tasks or modes, e.g., we could ask a robot to reach an object, grab it, and bring it back. This example resembles some sort of human to human (or rather human to dog) communication, i.e., a situation where only a few instructions are used, unlike classic control theory where a control value is specified at each instant. Some recent efforts focused on formalizing how multi-modal control can generate continuous motions from symbolic input strings have resulted in the design of a powerful and promising general purpose language for hybrid systems programming: the *Motion Description Language* (MDL).

The primary goal of this thesis is to develop methods for effectively defining, selecting, and coding the elementary instructions of a multi-modal control program. The idea is to provide a framework in which robots and other dynamical systems can be controlled using automatically generated high-level, symbolic control programs. In particular, we will develop methods for extracting high-level control programs from observed behaviors and

then produce symbolic control strategies that can be executed on mobile robots to mimic the observed behaviors. This empirical data can be generated from nature (as in a group of ants or schooling fish) or from human-operated robots. From the standpoint of naturally occurring data, the short term aim of such research would be to learn from nature, but a more lofty, long term goal would be to understand naturally occurring control mechanisms based on hybrid control theory. From the human-operator standpoint, the goal would be to learn effective control strategies from example.

In this work, the efficiency of a method will be measured in terms of the *complexity* and *performance* of specific control programs. Typically, we will look to minimize the number of bits transmitted, while guaranteeing that the system meets its specifications (robustness, stability, reachability, etc). The insistence on low-complexity programs is originally motivated by communication constraints on the computer control of semi-autonomous systems, but also by our belief that, as complex as they may look, natural systems indeed use short motion schemes with few basic behaviors. The attention is first focused on the design of such short-length, few-distinct-modes mode sequences within the MDL framework in Part 1. The multi-modal control problem is then addressed using an optimal control approach in Part 2, where a variety of problems are solved in order to improve performance. In particular, given a mode sequence, the question of deciding when the system should switch from one mode to another to achieve some stability and reachability requirements is studied.

1.2 Background

Here we give a brief literature review of work closely related to our research. We identify three main topics: Motion Description Languages, System Identification and Multi-Modal Estimation, and Optimal Control of Switched Autonomous Systems. It is assumed that the notions of Hybrid Systems and Multi-Modal Control are already understood, but for more information, the reader can refer to [18, 81].

1.2.1 Motion Description Languages

We first recall the definition of a formal language. Given a finite set, or *alphabet*, $\mathcal{A}$, whose elements are referred to as *letters*, by $\mathcal{A}^\star$ we understand the set of all strings, or *words*, of

finite length over $\mathcal{A}$, with the binary operation of concatenation defined on $\mathcal{A}^\star$. Relative to this operation, $\mathcal{A}^\star$ is a semigroup, and if we include the empty string in $\mathcal{A}^\star$ it becomes a monoid, i.e., a semigroup with an identity, and a *formal language* is a subset of the free monoid over a finite alphabet. (See for example [48] for an introduction to this subject.)

The use of a language-based approach to motion control emerged about twenty years ago. In a 1988 paper, R.W. Brockett first introduced "Motion Description Languages" as an attempt to answer the need for a general purpose computer control of movement [17]. In this framework, a computer controlled mechanism called an *MDL device* is sent a sequence of triples $(u_1(.), k_1(.), T_1)...(u_r(.), k_r, (.), T_r)$ where $\forall i \in \{1, ..., r\}$, $u_i()$ is an open-loop control, $k_i()$ is a closed-loop control, and T_i is the amount of time over which both $u_i()$ and $k_i()$ should be used. To illustrate this, consider a mechanism whose dynamics is given by

$$\dot{x}(t) = f(x(t)) + G(x(t))v(t), \tag{1}$$

$$y(t) = h(x(t)). \tag{2}$$

The evolution of the corresponding MDL device is as follows: if at time T_0, the input control string $(u_1, k_1, T_1)(u_2, k_2, T_2)(u_3, k_3, T_3)$ is received, then the state x will evolve according to

$$\begin{aligned}
\dot{x}(t) = &\quad f(x(t)) + G(x(t))(u_1 + k_1(y(t))) \qquad T_0 \leq t < T_0 + T_1, \\
\dot{x}(t) = &\quad f(x(t)) + G(x(t))(u_2 + k_2(y(t))) \qquad T_0 + T_1 \leq t < T_0 + T_1 + T_2, \\
\dot{x}(t) = &\quad f(x(t)) + G(x(t))(u_3 + k_3(y(t))) \qquad T_0 + T_1 + T_2 \leq t < T_0 + T_1 + T_2 + T_3.
\end{aligned}$$

The triples $(u_i(), k_i(), T_i)$ are referred to as *modal segments*[1] because they define modes of control over a segment of time. The analogy with a language is that these modal segments can be interpreted as letters, drawn from of finite collection, or *motion alphabet*, and concatenated to form control programs (words). Thus, a Motion Description Language (MDL) is a formal language defined over a motion alphabet.

An important result in [17] is that the restriction to a finite family of affine modal segments (particular modes such that $v = u_i + k_i(y) = a + M(y - b)$ for some a, b, and M with

[1] Many different appellations have been used since, and for the rest of this proposal, the reader should not get confused by the quantity of terms such as *modes*, *atoms*, *behaviors*, *letters*, *primitives* as they all refer to this same notion.

appropriate dimensions) does not limit the expressiveness of the interpreter/mechanism. In other words, such segments can be used to follow arbitrarily closely any state trajectory the original system can produce if controlled by a continuous v. Of course, in most cases, the original curve can ideally be reproduced only when the cycle period of the discrete version of Equation (1) goes to zero and the length of the mode sequence r goes to infinity. The paper, although implicitly, already introduces the issue of a trade-off between *complexity* of the mode string and *expressiveness* of the system.

This question of complexity of programs using MDLs was later studied by Egerstedt et. al. in [30, 36]. The most important result for us is that the use of feedback mappings within the framework of MDLs can significantly reduce the complexity of control programs. In other words, it is shown how the availability of sensory information can reduce the length of control procedures. This result is then applied to robot navigation in [33], where it is deduced that landmark-based navigation through a series of intermediary goals is preferable from a complexity point of view, and it is shown how the sensors' resolution should be chosen to minimize complexity.

Finally, the MDL model first introduced by Brockett has evolved over the last two decades. Slightly different versions of MDLs have been proposed, but they all share the common feature that the individual atoms, concatenated together to form the control program, can be characterized by control-interrupt pairs. The work in papers [65, 66, 67] completed the first version by adding interrupt mappings, leading to a version now known as the "extended MDL" or MDLe. The syntax is the following: a kinetic state machine governed by a differential equation of the form $\dot{x} = f(x)+G(x)u$ and $y = h(x)$ ($x \in \mathbb{R}^n$, $y \in \mathbb{R}^k$, $u \in \mathbb{R}^p$) is controlled by a mode sequence whose atoms are now evanescent fields defined on space-time. More specifically, an atom is a triple $\sigma = (k(.), \xi(.), T)$ where k maps $\mathbb{R}^+ \times \mathbb{R}^n$ to $\mathbb{R}^n$, ξ is an interrupt function mapping $\mathbb{R}^k$ the output of k sensors to $\{0, 1\}$, and $T \in \mathbb{R}^+$ is the lifetime of the mode. The mode is active until the function ξ jumps from 0 to 1 or until T seconds elapse, whichever happens first. Note that with the choice of $u = k(x, t)$, the new model allows for both open-loop and closed-loop control within a mode. With the introduction of the interrupt functions, it now also allows for event-triggered interrupts. An

interesting work based on this extended language conducted at the University of Maryland can be found in [53] where the authors draw the outline of a workable, unified, and reusable framework for the design of a control software that integrates MDLe.

1.2.2 System Identification and Multi-Modal Estimation

The problem of inferring the nature of a system from a mere observation is old, and finds its roots in classic philosophy with Plato's allegory of the cave. A more modern version in system theory is the problem of System Identification, defined by Zadeh in 1962 as "the determination on the basis of input and output, of a system within a specified class of systems, to which the system under test is equivalent" [92]. Since then, the large interest for such a problem and its countless applications has generated a multitude of methods and algorithms, to the point that even MATLAB includes a "System Identification Toolbox" where the user can build and evaluate linear models of dynamic systems from measured input-output data. For a thorough explanation of the theory of System Identification and a detailed review of available methods, see for example [63] and [57].

Also, a keen interest for the system identification of time-varying systems has given rise to the so-called Adaptive System Identification. The basic idea consists of an on-line version of the off-line classic methods (see e.g., [68]), where the data is first filtered by a moving time window. This simple idea has given rise to efficient algorithms for slowly varying systems. However, in hybrid systems, the change in system parameters is often so abrupt that classical identification methods and their adaptive extensions are not well suited.

The problem of "Multi-Modal System Identification" started at least 35 years ago with [1]. In this paper, the authors consider the problem of state estimation of a linear discrete-time system operating in Markov dependent switching environments. Hence, it is not quite the system itself that was first considered to be "multi-modal," but rather the external disturbances influencing the system equations and the measurement equations. More recently, it is with the emergence of Hybrid Systems theory that such similar problems have found new breath. Over the last decade, numerous methods have been proposed to solve the problem of Hybrid System Identification, of which an exhaustive review can be found in [80] and

[56]. To name a few, and give the reader a fine selection of corresponding references, we find the clustering-based procedure [39, 40], the bounded-error procedure [9, 10], the Bayesian procedure [55, 5], the algebraic procedure [86, 85], and the mixed integer programming procedure [75]. All these methods propose to identify piecewise autoregressive exogenous (PWARX) models of the form $y(k) = f(x(k)) + e(k)$ where $e(k)$ is the error term and the PWA mapping f is defined as $f(x) = [x' \ 1]\,\theta_i$ if $x \in \chi_i, \ \ (i = 1, ..., s)$. In this last equation, $x(k)$ is a *vector of regressors* defined as $x(k) = [y(k-1)...y(k-n_a)u'(k-1)...u'(k-n_b)]'$ with $y \in R$, $u \in R^m$, $\theta_i \in R^{n+1}$ is a *parameter vector* (or mode) with $n = n_a + n_b$. Finally, the bounded regressor space χ is partitioned into s convex polyhedral *regions* $\{\chi_i\}_{i=1}^s$. The identification problem consists of, given a data set $\{(x(k), y(k))\}_{k=1}^N$, determining the parameter vectors $\{\theta_i\}_{i=1}^s$ and the regions of the polyhedral space $\{\chi_i\}_{i=1}^s$ that best fit the data. All methods include the following three steps: 1) the estimation of the s parameter vectors, 2) the classification of the data points, and 3) the estimation of regions. All methods tackle the first two steps in a different way. The last step is performed using Multicategory Robust Linear Programming [13] for estimating separating hyperplanes that minimize the number of misclassified data points. Finally, each procedure presents different features and drawbacks and, depending on the situations (noisy data, unknown model orders, no a priori physical insight, etc.), one procedure may be preferred to the others.

What sets the work apart is that the switched system identification is done with information theoretic concerns in mind. The assumption that unknown systems may be driven by low complexity programs, an idea that can be supported by the simplicity of the mathematical schemes nature presents us, has in fact not been used as an argument in hybrid system identification. Also, we assume the dynamics is known and we identify control laws and interrupts rather than the matrices of a linear model.

1.2.3 Optimal Control of Switched Autonomous Systems

As mentioned, the second part of this thesis addresses the multi-modal control problem in an optimal control setting. A number of problems are considered, where a system's dynamics switch accordingly to a pre-specified mode sequence. Within each mode, the system under

consideration is autonomous and the task consists of finding an optimal switching policy, i.e., conditions for when the system should switch from one mode to another, minimizing a given performance criterion.

Switched Autonomous Systems arise in a variety of applications, including situations where a control module has to switch its attention among a number of subsystems [50, 62, 72, 87] or collect data sequentially from a number of sensor sources [19, 37, 51]. They also appear in landmark robot navigation, where a sensor-based navigation plan is specified through a series of intermediary goals [32, 43, 60].

In general, there has been a mounting interest in optimal control of switched systems, where the control variable consists of a proper switching law as well as an input function (see [16, 20, 45, 76, 79, 88, 90]). Switched Autonomous Systems, which constitute a subclass of systems where the input function is absent, have also received a lot of attention over the last few years. Different approaches were considered, depending on whether the systems were linear [62, 11, 42, 41] or non-linear [91, 89], discrete-time [62, 12] or continuous-time [11, 91, 89], and whether the interrupts were time-triggered [91, 89] or event-triggered [42, 41, 15]. For example, in [42, 41], the authors consider an event-driven continuous-time linear system and present a numerical algorithm for computing the switching regions on the state space. In the case of time-triggered continuous-time linear systems, Bemporad et al. proposed a method for finding both the optimal switching instants and the sequence of operating modes. Of particular interest is the work by Xu and Antsaklis in [91, 89] where optimal time switches are derived for general non-linear systems. In that work, a formula for the gradient of a performance criterion is derived and applied in various nonlinear programming algorithms. A similar approach will be considered in Part 2, where we will develop a formula, simpler than the one in [91, 89], for the gradient of the cost functional, and use it in conjunction with a gradient-descent algorithm.

PART I

Mode Recovery

Recovering hybrid representations from the observation of systems that, at a first glance, do not present clearly distinguishable modes of operation is a challenge. The design of such models could help understand the basic behaviors of these complex systems (e.g., insects) and therefore predict them, but it could also inspire the design and control of robust semi-autonomous systems (e.g., navigating robots).

This first part of the thesis tackles this problem of extracting high-level representations of observed systems. The approach is the following. Chapters 2 and 3 focus on the recovery of low-complexity control programs from data, within the MDL framework. The idea (developed in [25, 23, 24]) is to identify a succession of deterministic feedback mappings and interrupts that can reproduce some observed data. Different methods are proposed, and tested in an application involving ants [35]. Chapter 4 shows how such recovered control programs can, in turn, be simulated to mimic observed behaviors and/or used for controlling real systems. A study of the relationship between the complexity of the control programs and the performance of the controlled system is also provided. In Chapter 5, the control programs are used to provide hybrid (possibly stochastic) automata representations of the observed systems. Finally, all these methods have been unified in MODEbox (in [28]), a MATLAB software package presented in Chapter 6.

CHAPTER II

MODE RECOVERY

This chapter serves as an introduction to the problem of mode recovery from observations. First, we show how the problem can be addressed within a particular MDL framework where modes consist of feedback control-interrupt pairs. Here, the observation data takes the form of a quantized input/output string, and the mode recovery task consists of finding *consistent* control programs, i.e., mode sequences that can generate this data string. Moreover, the resulting control programs are viewed as having an information theoretic content, in the sense that they can be coded quite effectively [21]. In this context, one can ask questions concerning minimum *complexity* programs, given a particular control task. After carefully defining consistency, as well as a suitable complexity measure, we will formulate the problem of low-complexity mode recovery. The actual methods developed for recovering control programs (mode sequences) will be presented in Chapter 3.

2.1 The Model

Particular choices of Motion Description Languages (MDL) become meaningful only when the language is defined relative to the physical device that is to be controlled. One such physical device is the so-called *quantized input-output machine*, given by $M = (U, X, Y, f, h)$, where U is a finite set of admissible inputs, Y is a finite set of outputs, $X \subset \mathbb{R}^n$ is the state space of the system, $f : X \times U \to X$ defines the system evolution, and $h : X \to Y$ is a measurable output function, such that the evolution of the machine is given by $\dot{x} = f(x, u),\ y = h(x)$. The assumption that the input and output spaces U and Y are finite is legitimate if we consider that the sensors and actuators have finite resolution and finite range. Now, we are interested in providing a model for the symbolic control of this machine, using an appropriate MDL. The particular structure of a mode should allow the design of robust control programs, i.e., programs that do not alter performance, when they are applied

to systems evolving in possibly unknown and changing environments, as is the case in robot navigation. For this reason, the modes should consist of feedback mappings from the output space to the input space. Finally, and for simplicity, we choose to use a sequential control: at any time, the system should be using one and only one mode of operation. The control should then consist of a mode sequence, where each mode is described by a feedback law and an interrupt law that, when triggered, makes the system switch to the next mode. A model meeting these specifications and found in [32] is the following:

Definition 2.1.1 (Motion Description Language) *Given a quantized input-output machine M, relative to M, we let a motion description language be given by a subset of the free monoid over the set $\Sigma = U^Y \times \{0,1\}^Y$.*

In other words, we let the letters in the motion alphabet be pairs of the form (k, ξ), where $k : Y \to U$, and $\xi : Y \to \{0,1\}$. If, at time t_0, M receives the input string $(k_1, \xi_1), \ldots, (k_p, \xi_q)$, then x evolves according to

$$\begin{aligned} \dot{x} &= f(x, k_1(y)); \qquad t_0 \le t < T_1 \\ &\vdots \qquad\qquad\qquad \vdots \\ \dot{x} &= f(x, k_q(y)); \quad T_{q-1} \le t < T_q, \end{aligned}$$

where T_i denotes the time at which the interrupt ξ_i changes from 0 to 1.

Navigation Example

In order to make matters more concrete, we illustrate these ideas with a navigation example, found in [31]. What makes the control of mobile robots particularly challenging is the fact that the robots operate in unknown or partially unknown environments. Any attempt to model such a system must take this fact into account. We achieve this by letting the robot make certain observations about the environment, and we let the robot dynamics be given by

$$\begin{aligned} \dot{x} &= v, \;\; x, v \in \mathbb{R}^2 \\ y_1 &= o_d(x), \;\; y_2 = c_f(x), \end{aligned}$$

where o_d is a quantized, odometric position estimate of x, and c_f is the quantized contact force from the environment. The contact force could either be generated by tactile sensors in contact with the obstacle or by range sensors such as sonars, lasers, or IR-sensors.

Relative to this robot it is now possible to define a MDL for executing motions that drive the robot toward the goal when the robot is not in contact with an obstacle. On the other hand, when the robot is in contact with an obstacle, it seems reasonable to follow the contour of that obstacle in a clock-wise or counter clock-wise fashion, as suggested in [49]. We let the MDL be given by the free monoid over the set

$$\big\{(k,\xi)\ |k(y_1,y_2)\in\{\kappa(x_F-y_1), cR(-\pi/2)y_2\},\ \xi\in\{\xi_{GA},\xi_{OA}\}\big\}.$$

The idea here is that the goal is located at x_F, and when the robot is not in contact with an obstacle, the closed-loop mapping $k(y_1,y_2)=\kappa(x_F-y_1)$ provides a simple, proportional feedback law. When the robot is in contact with an obstacle, $k(y_1,y_2)=cR(-\pi/2)y_2$, where $c>0$, $R(\theta)$ is a rotation matrix, and the choice of $\theta=-\pi/2$ corresponds to a clockwise negotiation of the obstacle.

In other words, the multi-modal control sequence used for reaching x_F while negotiating obstacles is thus an element in the set $\sigma_{GA}\cdot(\sigma_{OA}\cdot\sigma_{GA})^\star$, where GA and OA denotes "goal-attraction" and "obstacle-avoidance" respectively, and where $a^\star=\{\emptyset,a,aa,aaa,...\}$, with $\emptyset$ denoting the empty word. The individual modes $\sigma_{GA}=(k_{GA},\xi_{GA})$ and $\sigma_{OA}=(k_{OA},\xi_{OA})$ are furthermore given by

$$\begin{cases} k_{GA}(y_1,y_2)=\kappa(x_F-y_1) \\ \xi_{GA}(y_1,y_2)=\begin{cases} 0 \text{ if } \langle y_2, x_F-y_1\rangle\geq 0 \\ 1 \text{ otherwise} \end{cases} \end{cases}$$

$$\begin{cases} k_{OA}(y_1,y_2)=cR(-\pi/2)y_2 \\ \xi_{OA}(y_1,y_2)=\begin{cases} 0 \text{ if } \langle y_2, x_F-y_1\rangle<0 \text{ or } \angle(x_F-y_1,y_2)<0 \\ 1 \text{ otherwise.} \end{cases} \end{cases}$$

Here $\angle(\alpha,\beta)$ denotes the angle between the vectors α and β. An example of using this multi-modal control sequence is shown in Figure 1.

Now, in [36], a finite automata version of the quantized input-output machine was introduced, called a *Free-Running, Feedback Automaton* (FRFA), in order to arrive at an abstract model of, for example, a landmark-based navigation system for mobile robots [34, 52]. If we let X,U,Y be finite sets, and let $\delta\in X^{X\times U}$, $\gamma\in Y^X$, then we can identify

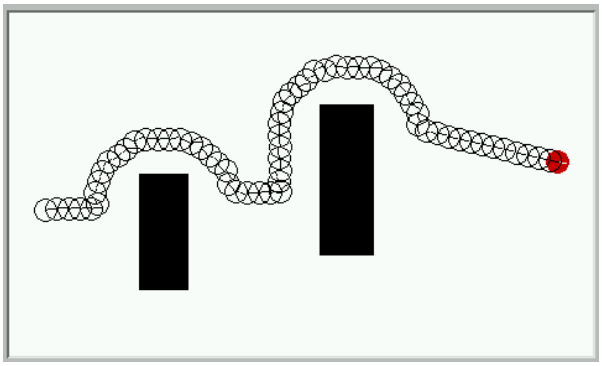


Figure 1: Basic Robot Navigation Example Using MDL.

$(X, Y, U, \delta, \gamma)$ with an *output automaton*, whose operation is given by $x(q+1) = \delta(x(q), u(q))$ and $y(q) = \gamma(x(q))$.

However, to let finite automata read strings of control modes, the model must be modified in such a way that instruction processing is akin to the way in which differential equations "process" piecewise constant inputs. The idea is to let the automaton read an input from a given alphabet, and then advance the state of the automaton repeatedly (free-running property) without reading any new inputs until an interrupt is triggered. Additional structure is furthermore imposed on the input set to allow for feedback signals to be used. Hence a FRFA is a free-running automaton whose input alphabet has the structure $\Sigma = K \times \Xi$, where, as before, $K = U^Y$ and $\Xi = \{0, 1\}^Y$. Therefore, the input to a FRFA is a pair (k, ξ), where $k : Y \to U$ and $\xi : Y \to \{0, 1\}$.

Definition 2.1.2 (Free-Running, Feedback Automaton [36]) *Let X, Y, U be finite sets and let $\delta : X \times U \to X$, $\gamma : X \to Y$ be given functions. Let $\Sigma = K \times \Xi$, where $K = U^Y$ and $\Xi = \{0, 1\}^Y$. We say that $(X, \Sigma, Y, \delta, \gamma)$ is a free-running, feedback automaton whose evolution equation is*

$$x(q+1) = \delta(x(q), k_{p(q)}(y(q))) \tag{3}$$

$$y(q) = \gamma(x(k)) \tag{4}$$

$$p(q+1) = p(q) + \xi_{p(q)}(y(q)), \tag{5}$$

given the input string $(k_1, \xi_1) \cdots (k_L, \xi_L) \in \Sigma^\star$.

In this definition, Equations (3) and (4) describe the state and output evolutions, while Equation (5) tracks the position of the active mode within the mode sequence.

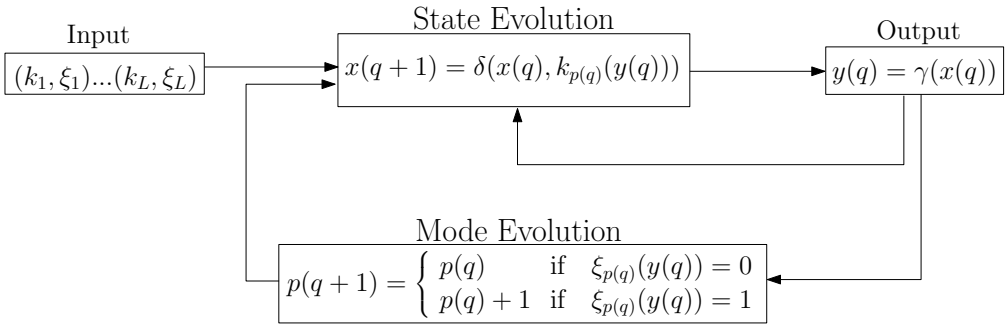


Figure 2: Free-Running Feedback Automaton.

Note that we do not consider any communication constraint regarding the size of the communication bus. At every time increment, all data between sensors, sensor processors, control processors, and actuators is assumed to be fully transmitted without errors. The author refers to an interesting work in [19, 50] where limited communication constrains the controller to choose which of the inputs to update at every cycle. We also assume no delays caused by a slow data processing or slow communications.

2.2 String Consistency

In the introduction to this chapter, we described the mode recovery problem as the problem of finding *consistent* control programs, i.e., mode sequences that can generate a given data string. Now that a particular and appropriate model has been chosen, this notion of *consistency* can be defined more specifically:

Definition 2.2.1 *Given an input-output string*

$$S = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} y(1) & y(2) & ... & y(N) \\ u(1) & u(2) & ... & u(N) \end{bmatrix} \in (\mathbb{Y} \times \mathbb{U})^N,$$

we say that a mode sequence $\sigma \in \Sigma^*$ is consistent with S if there exists a FRFA controlled by σ that produces S. In other words, $\sigma = \sigma_{m(1)}\sigma_{m(2)} \ldots \sigma_{m(L)}$, where $\forall j \in \{1, \ldots, L\}, \quad \sigma_{m(j)} \triangleq$

$(k_{m(j)}, \xi_{m(j)}) \in \Sigma_{total}$, *is consistent with* S *when:*

$$k_{m(p(i))}(y(i)) = u(i) \qquad i = 1, \ldots, N \tag{6}$$

$$p(i+1) = p(i) + \xi_{m(p(i))}(y(i)) \qquad i = 1, \ldots, N-1 \tag{7}$$

In this definition,

- N is the length of the input-output string, i.e., the number of data points.
- L is the length of the mode sequence.
- p is a mapping from $\{1, ..., N\}$ to $\{1, ..., L\}$, where $p(i)$ is the position (within the mode sequence) of the mode which is active when the i^{th} data point is read. Hence, p is a non-decreasing step function initiated at $p(1) = 1$.
- m is a mapping from $\{1, ..., L\}$ to $\{1, ..., M\}$, where $m(j)$ is the index of the mode in j^{th} position in the mode sequence. This notation assumes that all the distinct modes within the mode sequence are labelled σ_1, σ_2,...,σ_M, where M is the total number of distinct modes. E.For example, for a mode sequence $\sigma = \sigma_1\sigma_2\sigma_1\sigma_1\sigma_3$, m is a mapping from $\{1, ..., 5\}$ to $\{1, 2, 3\}$, with $m(1) = 1$, $m(2) = 2$, $m(3) = 1$, $m(4) = 1$, $m(5) = 3$. Note that when $M < L$, m is non-injective, meaning that at least one mode appears more than once in the mode sequence.

With this notation, $m(p(i))$ corresponds to the active mode at time i. Equation (6) ensures that the feedback mapping of the active mode at time i correctly maps the current output $y(i)$ to the current input $u(i)$[1]. Equation (7) ensures that if an interrupt is triggered at time i, the next mode in the mode sequence becomes the new active mode at time $i+1$. If no interrupt is triggered, the active mode at i remains the same at $i+1$.

The process of finding a mode sequence consistent with an output/input string is what we call *mode recovery* or *mode reconstruction*. Given an output/input string S, we denote by $\mathcal{C}_S \subset \Sigma^*$ the set of all mode sequences consistent with S. As the following theorem

[1]Note that we make the choice of deterministic mappings (i.e., an exact matching between outputs and inputs). In Section 5.1, we will see how these mappings can be turned into more realistic stochastic mappings.

states, $\mathcal{C}_S$ is non-empty, which means it is always possible to find at least one mode sequence consistent with S.

Theorem 2.2.1 *Given an input-output string* $S = ((y(1), u(1)), ..., (y(N), u(N))) \in (\mathbb{Y} \times \mathbb{U})^N$*, the set* $\mathcal{C}_S$ *of mode sequences consistent with* S *is non-empty.*

Proof: Consider the mode sequence $\sigma = \sigma_1\sigma_2...\sigma_N$ such that $\forall i \in \{1, ..., N\}$, $\sigma_i = (k_i, \xi_i)$ with $k_i(y(i)) = u(i)$ and $\xi_i(y(i)) = 1$. It is easy to show, by induction, that for this particular mode sequence, $p(i) = i$ and $m(p(i)) = m(i) = i$, $\forall i \in \{1, ..., N\}$. Consequently, Equations (6)-(7) for consistency are verified, and $\sigma \in \mathcal{C}_S$. ■

In general, the number of mode sequences consistent with a given input-output string is quite large (it grows exponentially with the number N of data points) but finite. With this in mind, our goal is to identify mode sequences minimizing some complexity measure on the set $\mathcal{C}_S$ of all consistent mode sequences.

2.3 Complexity Measures

Because of the finite resolution and range of sensors and actuators and/or the required quantization of such signals before computer processing, it is not a restriction to assume that the input and output sets $\mathbb{U}$ and $\mathbb{Y}$ are finite. Hence, it follows that the number of possible modes is also finite. For example, in the case where modes are of the form (k, ξ) where k is the feedback mapping from the output set $\mathbb{Y}$ to the input set $\mathbb{U}$ and ξ is the interrupt mapping from $\mathbb{Y}$ to $\{0, 1\}$, the number of possible modes is $2^{|\mathbb{Y}|}|\mathbb{U}|^{|\mathbb{Y}|}$, where the operator $|.|$ gives the number of elements in a set. A naive approach for coding such modes would be to assign the same number of bits to each mode. By doing this, each mode would be represented using $\lceil |\mathbb{Y}|(1 + log_2(|\mathbb{U}|)) \rceil$ bits. This coding scheme can only be optimal if all the modes have the same probability. Such assumption would go against the motivation and beliefs of the author that complex motions can be interpreted as the juxtaposition of some particular behaviors and that systems only use a few modes of operation. In such cases where the distribution of modes is all but uniform, Shannon's celebrated theorem

[77] can be invoked for the design of an optimal coding scheme. Indeed, modes that are more frequent should be coded using fewer bits. Here, the minimal expected code length l satisfies:

$$\mathcal{H}(\mathcal{A}) \leq l \leq \mathcal{H}(\mathcal{A}) + 1, \tag{8}$$

where $\mathcal{A}$ is the alphabet, here a subset of $\mathbb{U}^{\mathbb{Y}} \times \{0,1\}^{\mathbb{Y}}$ (the set of all possible mappings $k : \mathbb{Y} \to \mathbb{Y}$ and $\xi : \{0,1\} \to \mathbb{Y}$), and $\mathcal{H}(.)$ is the function entropy, defined by:

$$\mathcal{H}(\mathcal{A}) = -\sum_{a \in \mathcal{A}} p(a) \log_2(p(a)), \tag{9}$$

where $p(a)$ is the probability of the mode a in $\mathcal{A}$.

Along these lines, a measure for the complexity[2] of a given mode sequence $\sigma \in \Sigma^*$ is its *specification complexity*, defined as the expected number of bits required for encoding σ when an optimal coding scheme is used:

Definition 2.3.1 ***(Specification Complexity)*** *Given a finite alphabet Σ and a probability distribution p over Σ, we say that a mode sequence $\sigma \in \Sigma^*$ has specification complexity*

$$\mathcal{S}(\sigma, \Sigma) \triangleq |\sigma| \mathcal{H}(\Sigma).$$

To implement this complexity measure, we need a probability distribution over the alphabet Σ. In our case, where a mode sequence σ is recovered from an original output/input string, a probability distribution can be established by identifying the modes that compose σ, e.g if $\sigma = \sigma_1\sigma_1\sigma_2\sigma_1\sigma_3$ is recovered, then we can let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ and the corresponding probabilities are $p(\sigma_1) = 3/5, p(\sigma_2) = 1/5, p(\sigma_3) = 1/5$. In such a case where Σ and a probability distribution p are entirely built after a mode recovery, the specification complexity has the particularity that it only depends on σ. We give it the name of *empirical specification complexity*:

[2] We acknowledge the existence of alternative paradigms, such as the Kolmogorov complexity [61] and Rissanen's minimum description length [74].

Definition 2.3.2 ***(Empirical Specification Complexity)*** *Given a mode sequence* $\sigma \in \Sigma^*$*, we say that* σ *has empirical specification complexity*

$$\mathcal{S}^e(\sigma) \triangleq |\sigma|\mathcal{H}(\sigma) = -\sum_{i=1}^{M(\sigma)} \lambda_i(\sigma) \log_2 \frac{\lambda_i(\sigma)}{|\sigma|} \tag{10}$$

In this definition, $M(\sigma)$ is the number of distinct modes in σ and $\lambda_i(\sigma)$ is the number of occurrences of mode σ_i in σ. For the example where $\sigma = \sigma_1\sigma_1\sigma_2\sigma_1\sigma_3$ is recovered, the empirical specification complexity is $\mathcal{S}^e(\sigma) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{1}{5}\log_2\frac{1}{5} - \frac{1}{5}\log_2\frac{1}{5} \approx 1.37$ *bits*.

Instead of the base-2 logarithm, we may prefer using the natural logarithm $\log_e$ (or simply log). The reason is that its derivative $\frac{d}{dx}\log x = \frac{1}{x}$ does not carry extra terms as is the case with $\log_2$ where $\frac{d}{dx}\log_2 x = \frac{1}{x\log 2}$. When log is preferred, the specification complexity is measured in *nats*. The conversion *nats*/*bits* is performed using the formula $\log_2 x = \frac{\log x}{\log 2}$. In our example $\sigma = \sigma_1\sigma_1\sigma_2\sigma_1\sigma_3$, the empirical specification complexity measured in *nats* would be $1.32 \log 2 \approx 0.95$ *nats*. From this point forward, unless otherwise specified, empirical specification complexities will be measured in *nats*. Also, for simplicity, we may refer to "empirical specification complexity" as simply "specification complexity." In both cases, we will just use the simpler notation $\mathcal{S}$, whether or not the probability distribution is established empirically.

2.4 Problem Formulation

Now that we have defined a suitable measure of complexity, we are able to formulate the optimal mode recovery problem as:

Problem 2.4.1 *(**Minimum Specification Complexity** $\mathbf{P}_{\mathcal{S}}(\mathcal{C}_S)$): Given an input-output string $S = (\boldsymbol{y}, \boldsymbol{u}) \in (\mathbb{Y} \times \mathbb{U})^N$, find the* simplest *mode sequence consistent with S. In other words, find $\sigma_{\mathcal{S}^*}$ that solves:*

$$\min_{\sigma \in \mathcal{C}_S} \mathcal{S}(\sigma) \Leftrightarrow \begin{cases} \min_{\sigma \in \Sigma^*} \mathcal{S}(\sigma) \quad \text{subject to} \\ \begin{cases} p(i+1) = p(i) + \xi_{m(p(i))}(y(i)) & i = 1, \ldots, N-1 \\ k_{m(p(i))}(y(i)) = u(i) & i = 1, \ldots, N. \end{cases} \end{cases} \qquad (P_{\mathcal{S}}(\mathcal{C}_S))$$

We say that $\sigma_{\mathcal{S}^*} \in sol(P_{\mathcal{S}}(\mathcal{C}_S))$ if $\sigma_{\mathcal{S}^*}$ solves $P_{\mathcal{S}}(\mathcal{C}_S)$. The fact that $\mathcal{C}_S$ is non-empty (Theorem 2.2.1) and finite, and that $\mathcal{S}(.)$ is, as we will see later, bounded above on $P_{\mathcal{S}}(\mathcal{C}_S)$, ensures that the specification complexity can be minimized, and attains its minimum. In other words, $P_{\mathcal{S}}(\mathcal{C}_S)$ is solvable, or $sol(P_{\mathcal{S}}(\mathcal{C}_S))$ is non-empty. However, the uniqueness of a solution cannot be guaranteed. We will let $\mathcal{S}^*$ denote the unique specification complexity of the solutions to $P_{\mathcal{S}}(\mathcal{C}_S)$.

The problem of minimizing the specification complexity is not easy to address. Well known properties in information theory can be derived from definition 2.3.1. Roughly, they state that low complexity data strings are achieved when:

- the length of the data string is small.
- the size of the alphabet (i.e. the number of distinct symbols within the data string) is small.
- the probability distribution over the alphabet is as unequally distributed as possible.

The minimum specification complexity is reached when a certain equilibrium between these three conditions is achieved. Finding such an equilibrium for a mode sequence that also satisfies Equations (6)-(7) for consistency is baffling. However, if we consider the length

$L(\sigma)$ of a mode sequence σ, and its number of distinct modes $M(\sigma)$, the easily established property

$$\mathcal{S}(\sigma) \leq L(\sigma) \log M(\sigma) \tag{11}$$

allows us to focus our efforts on two more tractable problems[3]: 1) minimizing $L(\sigma)$ and 2) minimizing $M(\sigma)$. If finding $\mathcal{S}^*$ is too hard, we can at least try to approach it. By finding mode sequences with either minimum length or minimum number of distinct modes, we should generate mode sequences with low specification complexity. These two problems, referred to as 'Min$L$' and 'Min$M$' are solved in the next chapter.

[3]As we will see in the next chapter, these two problems are not independent.

CHAPTER III

LOW-COMPLEXITY MODE RECOVERY

In this chapter, algorithms are derived for solving the minimum-length (Min$L$) and minimum-alphabet (Min$M$) mode recovery problems. However, in both of these algorithms, the minimization of one variable is achieved at the expense of the other. More precisely, σ_{L^*}, which minimizes L, also maximizes M. Similarly, σ_{M^*}, which minimizes M, also maximizes L. This issue is resolved by creating two complementary algorithms for the reduction of $M(\sigma_{L^*})$ and $L(\sigma_{M^*})$. In other words, for every solution σ_{L^*}, we can find another solution σ'_{L^*} with same length but fewer distinct modes ($M(\sigma'_{L^*}) \leq M(\sigma_{L^*})$). Similarly, for every solution σ_{M^*}, we can find a shorter solution σ'_{M^*} with same number of distinct modes ($L(\sigma'_{M^*}) \leq L(\sigma_{M^*})$). The chapter ends with a comparative study of the different recovery methods in an application where mode sequences are recovered from the observation of 10 ants in a tank.

3.1 Min$L$ Mode Recovery

Here we present the solution to the minimum-length recovery problem, as first discussed in [7]. The problem formulation is as follows:

Problem 3.1.1 *(**Min$L$-min Mode Recovery** $\mathbf{P}_L(\mathcal{C}_S)$): Given an input-output string $S = (\boldsymbol{y}, \boldsymbol{u}) \in (\mathbb{Y} \times \mathbb{U})^N$, find the* shortest *mode sequence consistent with S. In other words, find σ_{L^*} that solves:*

$$\min_{\sigma \in \mathcal{C}_S} L(\sigma) \Leftrightarrow \left\{ \begin{array}{l} \min_{\sigma \in \Sigma^*} L(\sigma) \quad \text{subject to} \\ \left\{ \begin{array}{ll} p(i+1) = p(i) + \xi_{m(p(i))}(y(i)) & i = 1, \ldots, N-1 \\ k_{m(p(i))}(y(i)) = u(i) & i = 1, \ldots, N. \end{array} \right. \end{array} \right. \qquad (P_L(\mathcal{C}_S))$$

We say that $\sigma_{L^*} \in sol(P_L(\mathcal{C}_S))$ if σ_{L^*} solves $P_L(S)$. The fact that $\mathcal{C}_S$ is non-empty (Theorem

2.2.1) and finite, and that $L(.)$ is bounded (since $\forall \sigma \in \mathcal{C}_S$, $L(\sigma) \leq N$) ensures that $L(.)$ can can be minimized and attains its minimum. In other words, $P_L(\mathcal{C}_S)$ is solvable, or $sol(P_L(\mathcal{C}_S))$ is non-empty. However, the uniqueness of a solution cannot be guaranteed. We will let L^* denote the unique length of the solutions to $P_L(\mathcal{C}_S)$.

In [7], the authors first show that the search can be restricted to a subset $\hat{\Sigma}^*$ of Σ^* where the interrupt mappings trigger for only one value of the output y. In other words, for every solution σ_{L^*} with alphabet $\Sigma = \mathbb{U}^{\mathbb{Y}} \times \{0,1\}^{\mathbb{Y}}$, there exists a solution $\hat{\sigma}_{L^*} \in \hat{\Sigma}^*$ with alphabet $\hat{\Sigma} = \mathbb{U}^{\mathbb{Y}} \times \hat{\Xi}$, where

$$\hat{\Xi} = \{\xi : \mathbb{Y} \to \{0,1\} \mid \xi(y) = 1 \text{ for exactly one } y \in \mathbb{Y}\} \subset \{0,1\}^{\mathbb{Y}}.$$

The problem is then solved on $\hat{\Sigma}^*$ using dynamic programming, where a *cost-to-go* is propagated backward along the input-output string using Bellman's equation:

$$\mathcal{V}_i(\xi) = \min_{\xi' \in \hat{\Xi}} \{\mathcal{C}_i(\xi, \xi') + \mathcal{V}_{i+1}(\xi')\}.$$

Here, $\mathcal{V}_i(\xi)$ is the minimum number of modes required for producing a mode sequence consistent with $((y(i), u(i)), ..., (y(N), u(N)))$ and such that the active mode at i uses interrupt ξ. Also, $\mathcal{C}_i(\xi, \xi')$ is the transition cost associated with using ξ at time i, and ξ' at time $i+1$. This transition cost is computed as follows:

- when ξ is the interrupt triggering on $y(i)$, $\mathcal{C}_i(\xi, \xi') = 1$, since a mode switch increases the number of modes by one.
- when ξ does not trigger on $y(i)$ but $\xi \neq \xi'$, $\mathcal{C}_i(\xi, \xi') = \infty$, since it is inconsistent to switch mode when no interrupt is triggered.
- when ξ does not trigger on $y(i)$ and $\xi = \xi'$, we need to check if the absence of a mode switch produces an inconsistency with respect to the data. To do so, we need to check if setting $k_\xi(y(i))$ to $u(i)$ contradicts a previous use of the active mode. If so, we have an inconsistency, which can be reflected by setting $\mathcal{C}_i(\xi, \xi) = \infty$. Otherwise, we can stay in the same mode and set $\mathcal{C}_i(\xi, \xi') = 0$.

With this setting, solving $P_L(\mathcal{C}_S)$ is equivalent to solving

$$\begin{cases} \min_{\xi\in\hat{\Xi}} \mathcal{V}_1(\xi) \quad \textit{subject to} \\ \begin{cases} \mathcal{V}_N(\xi) = 1, \ \forall \xi \in \hat{\Xi}, \\ \mathcal{V}_i(\xi) = \min_{\xi'\in\hat{\Xi}}\{\mathcal{C}_i(\xi,\xi') + \mathcal{V}_{i+1}(\xi')\} \qquad i = 1, ..., N-1. \end{cases} \end{cases}$$

An example found in [7] is reproduced in Figure 3, where mode sequences of minimum length are recovered from the input-output string

$$S = \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 & 0 & 1 \\ 2 & 0 & 0 & 2 & 1 & 0 \end{bmatrix}.$$

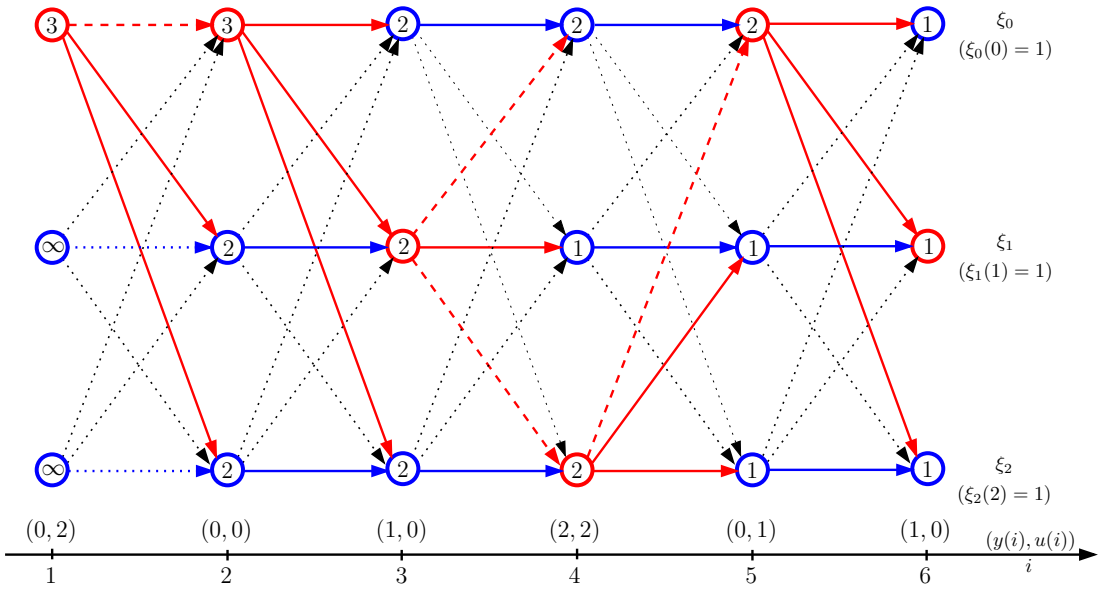


Figure 3: Min$L$ Mode Recovery using Dynamic Programming (example).

The graph is constructed from right to left to reflect the propagation of the costs-to-go backward in time. The numbers in the circles represent the costs-to-go. Dotted arrows represent inconsistent transitions (infinite transition cost). These are black when a mode switch occurs in spite of an interrupt function being triggered or blue when the absence of a mode switch leads to a feedback mapping inconsistency. All other arrows represent possible transitions. These are red when a mode switch occurs (transition cost equal to one) or blue otherwise (no transition cost). Finally, the arrows are solid on an optimal path and

dashed on a suboptimal one. The graph shows three possible optimal paths with minimum cost-to-go $\mathcal{V}_1 = 3$. They correspond to three mode sequences with minimum length $L^* = 3$.

3.2 MinM Mode Recovery

We now present an algorithm for solving the minimum-alphabet recovery problem. The problem formulation is as follows:

Problem 3.2.1 *(**MinM Mode Recovery** $\mathbf{P}_M(\mathcal{C}_S)$): Given an input-output string $S = (\boldsymbol{y}, \boldsymbol{u}) \in (\mathbb{Y} \times \mathbb{U})^N$, find the mode sequence with smallest alphabet (i.e. minimum number of distinct modes) consistent with S. In other words, find σ_{M^*} that solves:*

$$\min_{\sigma \in \mathcal{C}_S} M(\sigma) \Leftrightarrow \begin{cases} \min_{\sigma \in \Sigma^*} M(\sigma) \quad \text{subject to} \\ \begin{cases} p(i+1) = p(i) + \xi_{m(p(i))}(y(i)) & i = 1, \ldots, N-1 \\ k_{m(p(i))}(y(i)) = u(i) & i = 1, \ldots, N. \end{cases} \end{cases} \qquad (P_M(\mathcal{C}_S))$$

We say that $\sigma_{M^*} \in sol(P_M(\mathcal{C}_S))$ if σ_{M^*} solves $P_M(\mathcal{C}_S)$. The fact that $\mathcal{C}_S$ is non-empty (Theorem 2.2.1) and finite, and that $M(.)$ is bounded above (since $\forall \sigma \in \mathcal{C}_S$, $M(\sigma) \leq L(\sigma) \leq N$), ensures that $M(.)$ can be minimized and attains its minimum. In other words, $P_M(\mathcal{C}_S)$ is solvable, or $sol(P_M(\mathcal{C}_S))$ is non-empty. However, the uniqueness of a solution cannot be guaranteed. We will let M^* denote the unique length of the solutions to $P_M(\mathcal{C}_S)$.

Again, we solve this problem by restricting the search to a subset of $\mathcal{C}_S$, in particular, the set of Always Interrupt Sequences (AIS) consistent with S. As their name suggests, AIS are mode sequences that trigger an interrupt at every time increment. In other words, we say that $\sigma = \sigma_{m(1)}\sigma_{m(2)}...\sigma_{m(L)} \in \Sigma^*$ is an AIS when $\forall j \in \{1, ..., L\}$, $\sigma_{m(j)} = (k_{m(j)}, \xi_{m(j)})$ with $\forall y \in \mathbb{Y}$, $\xi_{m(j)}(y) = 1$. The AIS form a subset of Σ^* with alphabet $\Sigma_{AIS} = \mathbb{U}^{\mathbb{Y}} \times \{1\}^{\mathbb{Y}}$. Instead of $\mathcal{C}_S$, the search for a mode sequence with minimum alphabet is limited to the subset of consistent Always Interrupt Sequences: $\Sigma^*_{AIS} \cap \mathcal{C}_S \subset \mathcal{C}_S$. The problem becomes:

Problem 3.2.2 *(**MinM AIS Mode Recovery** $\mathbf{P}_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)$): Given an input-output string $S = (\boldsymbol{y}, \boldsymbol{u}) \in (\mathbb{Y} \times \mathbb{U})^N$, find the AIS with smallest alphabet (i.e. minimum number of distinct modes) consistent with S. In other words, find σ_{M^*} that solves:*

$$\min_{\sigma \in \Sigma^*_{AIS} \cap \mathcal{C}_S} M(\sigma) \Leftrightarrow \begin{cases} \min_{\sigma \in \Sigma_*} M(\sigma) \quad \text{subject to} \\ \begin{cases} \sigma_{m(j)} = (k_{m(j)}, \xi_{m(j)}) \text{ with } \forall y \in \mathbb{Y}, \xi_{m(j)}(y) = 1 & j = 1, ..., L \\ p(i+1) = p(i) + \xi_{m(p(i))}(y(i)) & i = 1, \ldots, N-1 \\ k_{m(p(i))}(y(i)) = u(i) & i = 1, \ldots, N. \end{cases} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)) \end{cases}$$

The following result simplifies the formulation:

Theorem 3.2.1 *(**Length of an AIS**) The length L of an AIS consistent with a given input-output string of length N is equal to N.*

Proof: We recall the consistency condition $p(i+1) = p(i) + \xi_{m(p(i))}(y(i))$, which describes the evolution of the position of the active mode within the mode sequence. In the context of AIS, this equation becomes $p(i+1) = p(i) + 1$, $(i = 1, ..., N-1)$. Together with $p(1) = 1$, this gives $p(i) = i$, $(i = 1, ..., N)$. Since an AIS triggers an interrupt at every time increment, the length L of the AIS is equal to the length N of the input-output string. ■

The problem now reduces to finding a minimal mapping m from $\{1, ..., N\}$ to $\{1, ..., M\}$ and feedback mappings $k_1, ..., k_M$ such that $\forall i \in \{1, ..., N\}$, $k_{m(i)}(y(i)) = u(i)$. By "minimal mapping," we mean that the size M of the image of m should be minimized. We propose the following algorithm:

Algorithm 3.2.1 *Given a mode string* $S = ((y(1), u(1)), ..., (y(N), u(N)))$*, this algorithm generates a consistent AIS* $\sigma = \sigma_{m(1)}\sigma_{m(2)}...\sigma_{m(N)}$ *using a minimum number* M *of distinct modes. The distinct modes (defined by their feedback mappings) and the active-mode mapping* m *are progressively constructed by reading the input-output string* S *from left to right. At iteration* $i \in \{1, ..., N\}$*,* $\rho(i)$ *represents the minimum number of modes used to generate a mode sequence* $\sigma = \sigma_{m(1)}...\sigma_{m(i)}$ *consistent with* $((y(1), u(1)), ..., (y(i), u(i)))$*. With this notation, the algorithm is initiated with* $\rho(0) = 0$*, i.e., no available mode. Then, for* $i = 1, ..., N$*, the algorithm does one of the following:*

- ***[case 1]*** *If there already exists a mode* $j \in \{1, ..., \rho(i-1)\}$ *such that* $k_j(y(i)) = u(i)$*, then* $m(i) = j$*. This means that mode* j *is used without modification. As no new mode is created,* $\rho(i) = \rho(i-1)$*.*
- ***[case 2]*** *If there is no mode* $j \in \{1, ..., \rho(i-1)\}$ *such that* $k_j(y(i)) = u(i)$*, but there exists one such that* $k_j(y(i))$ *is undefined, then* $k_j(y(i)) = u(i)$ *and* $m(i) = j$*. This means that mode* j *is used, but its definition is modified. As no new mode is created,* $\rho(i) = \rho(i-1)$*.*
- ***[case 3]*** *If for all modes* $j \in \{1, ..., \rho(i-1)\}$*,* $k_j(y(i))$ *is defined and* $k_j(y(i)) \neq u(i)$*, a new mode* $\sigma_{\rho i-1+1}$ *such that* $k_{\rho(i-1)+1}(y(i)) = u(i)$ *is created, and* $m(i) = \rho(i-1)+1$*. As a new mode is created,* $\rho(i) = \rho(i-1) + 1$*.*

An example is given in Figure 4, where a mode sequence consistent with a given input-output string S is recovered using Algorithm 3.2.1. At every step, the feedback mapping used to map $y(i)$ to $u(i)$ is shown within a box. The box is green in case 1, blue in case 2, and red in case 3.

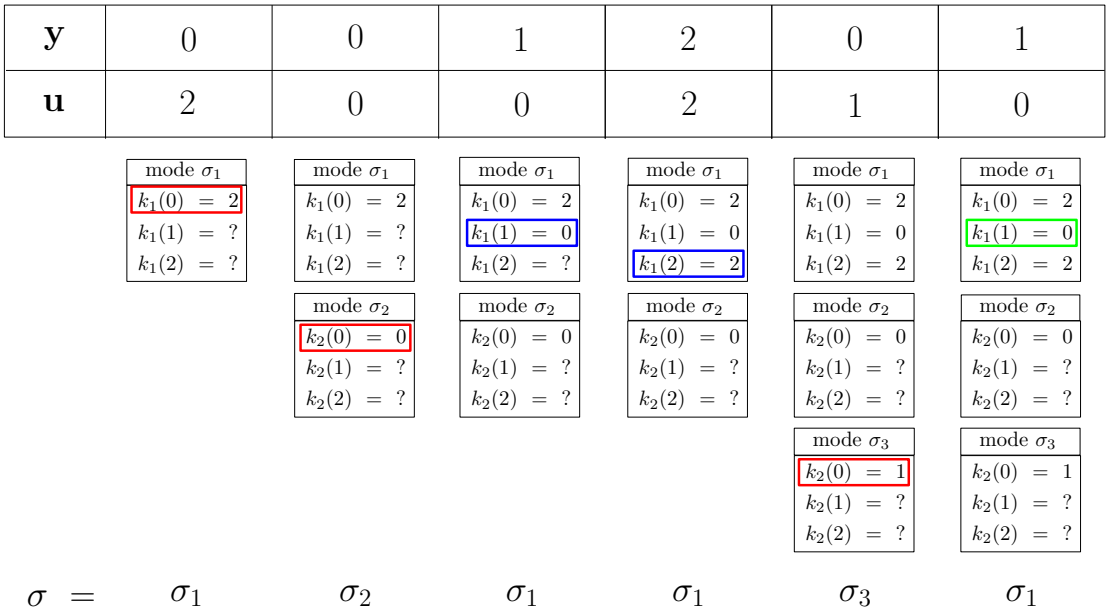


Figure 4: Min$M$ Mode Recovery using Algorithm 3.2.1 (example).

The recovered mode sequence uses three distinct modes. The reason is that the particular output value $y = 0$, must be mapped to three distinct values of the input ($u = 0$, 1, and 2). We generalize this with the following property:

Property 3.2.1 *For a given input-output string* $S = ((y(1), u(1)), ..., (y(N), u(N)))$*, Algorithm 3.2.1 generates a consistent AIS using exactly*

$$M_{AIS}(S) = \max\{card(\{u \mid (y, u) \in S\}) \mid y \in \mathbb{Y})$$

number of modes.

We now prove that Algorithm 3.2.1 indeed solves the problem at hand.

Theorem 3.2.2 *Any mode sequence* $\sigma \in \Sigma^*$ *consistent with a given input-output string* $S = ((y(1), u(1)), ..., (y(N), u(N)))$ *uses at least* $M_{AIS}(S)$ *distinct modes. In other words,*

$$\sigma \in \mathcal{C}_S \Rightarrow M(\sigma) \geq M_{AIS}(S).$$

Proof: Suppose that $\sigma \in \mathcal{C}_S$ uses $M(\sigma) < M_{AIS}(S)$ distinct modes. Denote $y_{\max}$ a value of $y \in \mathbb{Y}$ for which $M_{AIS}(S)$ is achieved. In other words, there are $M_{AIS}(S)$ distinct values of $u \in \mathbb{U}$ such that $(y_{\max}, u)$ appears in S. As $M(\sigma) < M_{AIS}(S)$, there must be two distinct couples $(y_{\max}, u(i))$ and $(y_{\max}, u(j))$ in S $(u(i) \neq u(j))$ assigned the same mode, i.e., $m(p(i)) = m(p(j))$. For these two couples, consistency requires that $k_{m(p(i))}(y_{\max}) = u(i)$ and $k_{m(p(j))}(y_{\max}) = u(j)$. However, because $k_{m(p(i))}(y_{\max}) = k_{m(p(j))}(y_{\max})$ and $u(i) \neq u(j)$, we have a contradiction. Consequently, any mode sequence consistent with S must use at least $M_{AIS}(\sigma)$ distinct modes. ■

Corollary 3.2.1 *Given an input-output string $S = ((y(1), u(1)), ..., (y(N), u(N)))$, Algorithm 3.2.1 generates a consistent AIS σ with minimum number M of distinct modes. In other words,*

$$\sigma \in sol(\mathrm{P}_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)).$$

Proof: From Property 3.2.1, $M(\sigma) = M_{AIS}$. From Theorem 3.2.2, for all $\sigma \in \Sigma^*$, $M(\sigma) \geq M_{AIS}$. Note that, since $\Sigma^*_{AIS} \cap \mathcal{C}_S \subset \Sigma^*$, it is also true on $\Sigma^*_{AIS} \cap \mathcal{C}_S$. Hence we have reached the minimum number of distinct modes. ■

Corollary 3.2.2 *Given an input-output string $S = ((y(1), u(1)), ..., (y(N), u(N)))$, Algorithm 3.2.1 generates a consistent mode sequence σ with minimum number M of distinct modes. In other words,*

$$\sigma \in sol(\mathrm{P}_M(\mathcal{C}_S)).$$

Proof: From Property 3.2.1, $M(\sigma) = M_{AIS}$. From Theorem 3.2.2, for all $\sigma \in \Sigma^*$, $M(\sigma) \geq M_{AIS}$. Hence we have reached the minimum number of distinct modes. ■

Algorithm 3.2.1 provides one element in $sol(P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S))$. However, there may be many more consistent AIS using the same (minimum) number of distinct modes M^*. These differ by how the distinct input-output data pairs are assigned to the M^* available modes. An interesting problem would be to find, among all possibilities, one minimizing the entropy, i.e., solve $P_{\mathcal{H}}(sol(P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)))$. Note that, since all elements in $sol(P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S))$ have the same length $L = N$ (Property 3.2.1), the specification complexity would also be minimized, i.e., $P_{\mathcal{H}}(sol(P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)))$=$P_{\mathcal{S}}(sol(P_M(\Sigma^*_{AIS} \cap \mathcal{C}_S)))$. The following algorithm

is presented to solve this problem. It uses the fact that for a word of given length and given alphabet, the entropy is minimized when the distribution of the alphabet is as unequally distributed as possible.

Algorithm 3.2.2 *Given an input-output string* $S = ((y(1), u(1)), ..., (y(N), u(N)))$*, this algorithm generates, among all consistent Always Interrupt Sequences with minimum alphabet* (M)*, one with minimum entropy* $\mathcal{H}$ *and specification complexity* $\mathcal{S}$*. This mode sequence is constructed as follows:*

For each $y \in \mathbb{Y}$*,*

- *find all distinct* $u \in \mathbb{U}$ *such that the pair* (y, u) *appears in* S*, and sort them by decreasing* $r(y, u)$*, where* $r(y, u)$ *is the number of times the pair* (y, u) *appears in* S*. We get a set* $\{u_1, u_2, ...u_n\}$ *with* $n \leq M_{AIS}$*, and such that* $r(y, u_1) \geq r(y, u_2) \geq ... \geq r(y, u_n)$*.*
- *for* $j = 1, ...n$*, set* $k_j(y) = u_j$ *and use mode* $\sigma_j = (k_j, \xi_j)$ *whenever* (y, u_j) *appears, i.e.,* $m(i) = j$*,* $\forall i$ *such that* $(y(i), u(i)) = (y, u_j)$*.*

3.3 Further Reductions

3.3.1 The $M \sim S$ Duality, or Why a Further Reduction is Necessary

As we already stated in Section 2.4, the problem of finding a consistent mode sequence with minimum specification complexity $\mathcal{S}^*$ is not easy. Instead, we designed two algorithms for producing consistent mode sequences with minimum length L^* or minimum number of distinct modes M^*. The motivation behind this alternative was that minimizing either term of the specification complexity upper bound $L \log M$ would generate mode sequences with significantly low specification complexity. However, in general, M and L are inversely related. This duality is an inherent property of formal languages: when the size of the alphabet increases, the average word length decreases (and vice versa). For example, numbers written in the decimal system are shorter than their binary representations. In solving the Min$L$ and Min$M$ problems, the minimization of L or M was made possible after tricks

were introduced to release some of the consistency constraints. But the use of such tricks comes at a high price. Indeed,

- when minimizing M, the use of modes that always interrupt results in a mode sequence with length equal to the length N of the I/O string. Hence, L is maximized.
- when minimizing L, the dynamic programming approach puts a cost on the action of switching from one mode to another. This cost is the same whether or not the next mode is a new mode or one that has already been used. With no examination of whether similar modes were used or not, the method in [7] generates mode sequences with, a priori, all modes being distinct. Hence, M is maximized and equals L^*.

One should now understand how, among all mode sequences consistent with a given I/O string, the solutions to the Min$L$ and Min$M$ problems are two extreme opposites. Solutions to the Min$\mathcal{S}$ problem, i.e., mode sequences with minimum specification complexity $\mathcal{S}^*$, lie somewhere in between, where a better balance between L and $\log M$ is achieved.
All the relations just mentioned are summarized in Table 1, where σ_{L^*} is a sequence with minimum length L^* generated by the algorithm in [7], σ_{M^*} is a sequence with minimum number of distinct modes M^* generated by algorithm 3.2.2, and $\sigma_{\mathcal{S}^*}$ is a sequence with minimum specification complexity $\mathcal{S}^*$.

To achieve our goal of producing low-complexity mode sequences (and ideally, approach $\mathcal{S}^*$), the mode sequences σ_{L^*} and σ_{M^*} need to be transformed so that their weak points, respectively, $M(\sigma_{L^*})$ and $L(\sigma_{M^*})$, are reduced.

3.3.2 Some Transformations and Their Effect on $\mathcal{H}$ and $\mathcal{S}$

Here, we consider how basic transformations of a mode sequence, such as adding, removing, or replacing modes, affect its entropy $\mathcal{H}$ and its specification complexity $\mathcal{S}$. In this subsection, mode sequences are only described by their modal distribution, i.e., a mode sequence is a collection $\{\lambda_i\}_{i=1}^{M}$, where $\lambda_i > 0$ is the number of occurrences of mode i, and M is the number of distinct modes within the mode sequence. The ordering of the modes, as

Table 1: Relations Between L, M, and $\mathcal{S}$.

mode sequence	σ_{L^*}		$\sigma_{\mathcal{S}^*}$		σ_{M^*}
length L	$L(\sigma_{L^*}) = L^*$	$\leq$	$L(\sigma_{\mathcal{S}^*})$	$\leq$	$L(\sigma_{M^*}) = N$
distinct modes M	$M(\sigma_{L^*}) = L^*$	$\geq$	$M(\sigma_{\mathcal{S}^*})$	$\geq$	$M(\sigma_{M^*}) = M^*$
upper bound $L \log M$	$L^* \log L^*$		$L \log M$		$N \log M^*$
specification complexity $\mathcal{S}$	$\mathcal{S}(\sigma_{L^*})$	$\geq$	$\mathcal{S}(\sigma_{\mathcal{S}^*}) = \mathcal{S}^*$	$\leq$	$\mathcal{S}(\sigma_{M^*})$

well as the consistency to a given I/O string, are intentionally neglected. Our goal is to identify what kind of transformations could be used to reduce the complexity of a given mode sequence. How such methods should be designed to preserve consistency to an I/O string will be addressed later.

Theorem 3.3.1 *Switching p occurrences of a mode j ($p \in [1, \lambda_j]$) to another mode k reduces both the entropy and the specification complexity if and only if $\lambda_k + p > \lambda_j$.*

Proof: Only modes j and k contribute to a change in entropy:

$$\begin{aligned}
\Delta\mathcal{H} &= \mathcal{H}(..., \lambda_j - p, \lambda_k + p, ...) - \mathcal{H}(..., \lambda_j, \lambda_k, ...) \\
&= -\frac{\lambda_j - p}{n} \log \frac{\lambda_j - p}{n} - \frac{\lambda_k + p}{n} \log \frac{\lambda_k + p}{n} + \frac{\lambda_j}{n} \log \frac{\lambda_j}{n} + \frac{\lambda_k}{n} \log \frac{\lambda_k}{n} \\
&= \frac{1}{n}\Big[(\lambda_j - p)\log(\lambda_j - p) - (\lambda_k + p)\log(\lambda_k + p) + \lambda_j \log \lambda_j + \lambda_k \log \lambda_k\Big] \\
&= \frac{1}{n}\Big[[x \log x]_{\lambda_j - p}^{\lambda_j} - [x \log x]_{\lambda_k}^{\lambda_k + p}\Big].
\end{aligned}$$

The derivative of $x \log x$ being monotonically increasing, we conclude that $\Delta\mathcal{H} < 0 \Leftrightarrow [x \log x]_{\lambda_j - p}^{\lambda_j} < [x \log x]_{\lambda_k}^{\lambda_k + p} \Leftrightarrow \lambda_k + p > \lambda_j$. Since $\Delta\mathcal{S} = n\Delta\mathcal{H}$, the same result holds for the specification complexity. ■

Corollary 3.3.1 *Merging two modes j and k reduces both the entropy and the specification complexity of a mode sequence.*

Proof: When merging two modes, all λ_j occurrences of a mode j are switched to another mode k. The condition $\lambda_k + p = \lambda_k + \lambda_j > \lambda_j$ being satisfied, we conclude, from the previous theorem, that the entropy and specification complexity are both reduced. ■

Also, when merging two modes, the memory space used to store one of these modes is saved. To stress the fact that the size of the motion alphabet is reduced, we will call this operation of merging modes "alphabet reduction."

Corollary 3.3.2 *Creating a new mode, and shifting p occurrences of a mode j to it, increases both the entropy and the specification complexity of a mode sequence.*

Proof: The role of a new mode can be played by a $M{+}1^{\text{th}}$ silent mode with $\lambda_{M+1} = 0$. Its presence does not modify the length of the original mode sequence (as $n = \sum_{i=1}^{M} \lambda_i = \sum_{i=1}^{M+1} \lambda_i$), nor its entropy (as $\frac{\lambda_{M+1}}{n} \log \frac{\lambda_{M+1}}{n} \to 0$). When creating a new mode, p occurrences of a mode j are switched to mode $M{+}1$, where p is any integer between 1 and $\lambda_j - 1$ (the case $p = \lambda_j$ would just result in switching the labels of the two modes). The condition in Theorem 3.3.1, $\lambda_{M+1} + p = p > \lambda_j$, is not satisfied. We conclude that entropy and specification complexity are both increased. ■

Theorem 3.3.2 *Removing occurrences of a mode j from a mode sequence*

i) reduces its entropy if $\sum_{i\neq j} \lambda_i \log \lambda_i \geq \sum_{i\neq j} \lambda_i \log \lambda_j$,

ii) always reduces its specification complexity.

Proof: i) We compute the partial derivative of $\mathcal{H}(\lambda_1, \lambda_2, ..., \lambda_M)$ with respect to λ_j. The effect of a variation in λ_j on the mode sequence length is reflected through $n = \sum_{i=1}^{M} \lambda_i = a + \lambda_j$ where $a = \sum_{i\neq j}^{M} \lambda_i > 0$ is constant.

$$\begin{aligned}
\frac{\partial \mathcal{H}}{\partial \lambda_j}(\lambda_1, ..., \lambda_j, ..., \lambda_M) &= \frac{\partial}{\partial \lambda_j}\Big[-\sum_{i\neq j}^{M} \frac{\lambda_i}{a+\lambda_j} \log \frac{\lambda_i}{a+\lambda_j} + \frac{\lambda_j}{a+\lambda_j} \log \frac{\lambda_j}{a+\lambda_j} \Big] \\
&= \sum_{i\neq j}^{M} \Big[\frac{\lambda_i}{(a+\lambda_j)^2} \log \frac{\lambda_i}{a+\lambda_j} + \frac{\lambda_i}{(a+\lambda_j)^2} \Big] - \frac{a}{(a+\lambda_j)^2} \log \frac{\lambda_j}{a+\lambda_j} - \frac{a}{(a+\lambda_j)^2} \\
&= \frac{1}{(a+\lambda_j)^2} \Big[\sum_{i\neq j}^{M} \lambda_i \log \lambda_i - \sum_{i\neq j}^{M} \lambda_i \log(a+\lambda_j) + \sum_{i\neq j}^{M} \lambda_i - a \log \lambda_j + a \log(a+\lambda_j) - a \Big] \\
&= \frac{1}{(a+\lambda_j)^2} \Big[\sum_{i\neq j}^{M} \lambda_i \log \lambda_i - a \log \lambda_j \Big].
\end{aligned}$$

Suppose that p occurrences of mode j are removed from a mode sequence ($p = 1, ..., \lambda_j$). A sufficient condition for $\Delta\mathcal{H} = \mathcal{H}(\lambda_1, ..., \lambda_j - p, ..., \lambda_M) - \mathcal{H}(\lambda_1, ..., \lambda_j, ..., \lambda_M) = -\int_{\lambda_j - p}^{\lambda_j} \frac{\partial \mathcal{H}}{\partial \lambda_j}(\lambda_1, ..., s, ..., \lambda_M) ds$ to be negative is that $\frac{\partial \mathcal{H}}{\partial \lambda_j}(\lambda_1, ..., s, ..., \lambda_M) > 0$ for all s in $[\lambda_j - p, \lambda_j]$. Noting that $\frac{1}{(a+s)^2}$ is always positive, and that $-a \log s$ is decreasing with s, a weaker sufficient condition is that $\frac{\partial \mathcal{H}}{\partial \lambda_j}(\lambda_1, ..., s, ..., \lambda_M) > 0$ at $s = \lambda_j$, i.e $\sum_{i \neq j}^{M} \lambda_i \log \lambda_i > a \log \lambda_j$.

ii) Similarly,

$$\begin{aligned}
\frac{\partial \mathcal{S}}{\partial \lambda_j}(\lambda_1, ..., \lambda_j, ..., \lambda_M) &= \frac{\partial}{\partial \lambda_j}\Big[-\sum_{i \neq j}^{M} \lambda_i \log \frac{\lambda_i}{a + \lambda_j} - \lambda_j \log \frac{\lambda_j}{a + \lambda_j} \Big] \\
&= -\sum_{i \neq j}^{M} \frac{\lambda_i}{a + \lambda_j} - \log \frac{\lambda_j}{a + \lambda_j} - \frac{a}{a + \lambda_j} \\
&= \log \frac{a + \lambda_j}{\lambda_j}.
\end{aligned}$$

This derivative being positive for any λ_j, we conclude that $\Delta\mathcal{S} = \mathcal{S}(\lambda_1, ..., \lambda_j - p, ..., \lambda_M) - \mathcal{S}(\lambda_1, ..., \lambda_j, ..., \lambda_M) = -\int_{\lambda_j - p}^{\lambda_j} \frac{\partial \mathcal{S}}{\partial \lambda_j}(\lambda_1, ..., s, ..., \lambda_M) ds < 0$. In other words, removing any p occurrences of a mode j will always decrease the specification complexity of the mode sequence. ■

3.3.3 Efficient Methods for the Specification Complexity Reduction

In the previous subsection, we considered three elementary transformations: adding, removing, or switching occurrences of a mode. These transformations were carefully defined so that they could deal with the destruction of a mode or the creation of a new one. Although more complex methods can be introduced, they would be redundant, as they can always be written as a concatenation of these three methods. One can also consider the case where only one occurrence of a mode is added, removed, or switched, and we will refer to these situations as *local transformations*. In most situations, when the inequalities in Table 1 are strict, i.e., $L^* < L(\sigma_{\mathcal{S}^*}) < L(\sigma_{M^*})$ and $M(\sigma_{L^*}) < M(\sigma_{\mathcal{S}^*}) < M^*$, the use of these local transformations in a descent-like manner cannot be used to converge from σ_{L^*} or σ_{M^*} to $\sigma_{\mathcal{S}^*}$. The reason is that

- since $L^* < L(\sigma_{\mathcal{S}^*})$, reaching $\sigma_{\mathcal{S}^*}$ from σ_{L^*} would require, at least once, the addition of an occurrence, which, according to Theorem 3.3.2, would locally increase the specification complexity.

- since $M^* < M(\sigma_{\mathcal{S}^*})$, reaching $\sigma_{\mathcal{S}^*}$ from σ_{M^*} would require, at least once, the creation of a mode, which, according to Corollary 3.3.2, would locally increase the specification

complexity.

Although the minimum complexity $\sigma_{\mathcal{S}^*}$ cannot be reached by the local transformations, a significant reduction in specification complexity could be achieved by reducing

- the alphabet size $M(\sigma_{L^*})$ of the minimum-length mode sequence generated by the algorithm in [7]. By merging modes while keeping the length of the mode sequence constant, Theorem 3.3.1 confirms a decrease in specification complexity.

- the length $L(\sigma_{M^*})$ of the minimum-alphabet mode sequence generated by Algorithm 3.2.2. By removing occurrences of certain modes (i.e. reducing the global length of the mode sequence), Theorem 3.3.2 also confirms a decrease in specification complexity.

In the following, we consider these two problems of minimizing (or at least reducing) $M(\sigma_{L^*})$ and $L(\sigma_{M^*})$. Difficulties lie in using transformations that do not affect the consistency of the mode sequences. Depending on the complexity of these two problems, optimal or suboptimal algorithms will be provided.

3.3.3.1 Minimizing $M(\sigma_{L^*})$

The problem of finding a minimum-length mode sequence consistent with a given input-output string was solved in Section 3.1. The Dynamic Programming algorithm in [7] provides a solution σ_{L^*}, where all the modes are a priori distinct. By "a priori," we stress the fact that every mode in the mode sequence was assumed to be a new mode, since no particular interest was paid to the size of the alphabet. Here, we remedy this issue by identifying similar modes and merging them, hence reducing the alphabet size $M(\sigma_{L^*})$, while keeping the length L^* constant.

First, we must define when two (or more) modes can be merged. With abuse of language, we will say that two or more modes are *mergeable* when they can be merged. We will also call *mergeability* the established relation between mergeable modes.

Definition 3.3.1 *Two modes $\sigma_i = (k_i, \xi_i)$ and $\sigma_j = (k_j, \xi_j)$ are said mergeable when their feedback mappings and interrupt functions are equal wherever their domains intersect, i.e., when $\forall y \in \mathbb{Y}$, if $k_i(y)$ and $k_j(y)$ are defined (which implies that $\xi_i(y)$ and $\xi_j(y)$ are also defined), then $k_i(y) = k_j(y)$ and $\xi_i(y) = \xi_j(y)$.*

Note that mergeability is not a transitive relation: if three modes i, j, and k are such that i and j are mergeable, and j and k are mergeable, we cannot conclude that i and k are mergeable. For this reason, we add the following definition:

Definition 3.3.2 *We say that n modes ($n \geq 3$) are mergeable when they are pairwise mergeable.*

With these definitions, it should be obvious that the action of merging two or more modes will preserve the consistency of a mode sequence if and only if the modes are mergeable.

Now, given a mode sequence $\sigma_{L^*} = \sigma_1\sigma_2...\sigma_L$, our goal is to find a mapping ID such that the resulting sequence $\sigma_{ID(1)}\sigma_{ID(2)}...\sigma_{ID(L)}$ uses a minimum of distinct modes. If ID maps $\{1, ..., L\}$ to $\{1, ..., k\}$, then the goal is to minimize k. An important requirement for consistency is that any two modes assigned the same ID (hence merged) must be mergeable, i.e., $\forall(\sigma_i, \sigma_j)$, $ID(i) = ID(j) \Rightarrow (\sigma_i, \sigma_j)$ *mergeable*. Equivalently, all modes assigned a same ID must form a group of mergeable modes.

This problem can be nicely translated into a similar problem in graph theory. Using a non-directed graph $G = (V, E)$, we represent each mode σ_i within σ_{L^*} by a vertex $v_i \in V$, and we let an edge $e_{ij} \in E$ connecting two vertices v_i and v_j represent the fact that the two modes σ_i and σ_j are mergeable. With this representation, a group of mergeable modes is a set $S \subset V$ of vertices such that for any two vertices v_i and v_j in S, there is an edge $e_{ij} \in E$ connecting the two vertices. Hence, a group of mergeable modes is what we refer to as a clique, a set of vertices that induces a complete subgraph of G. The problem of finding a minimum mapping ID such that all modes assigned a same ID are mergeable translates

here to what is known as the *Minimum Clique Partition* problem:

Problem 3.3.1 ***(Minimum Clique Partition)****: Given a non-oriented graph* $G = (V, E)$*, find a minimum collection of disjoint cliques* $K_1, K_2, ...K_n$ *that form a partition of* V *(i.e.* $K_1 \cup K_2 \cup ... \cup K_n = V$*).*

This is a well-known problem in computational complexity theory. Also referred to as the *Clique Cover*, this problem appears in the famous list of 21 NP-complete problems presented by Richard Karp in his 1972 paper [59]. NP-complete problems form a particularly difficult class of problems. As of today, and despite a $1 million dollar reward by the Clay Mathematics Institute in Cambridge[1], nobody has yet been able to prove if NP-complete problems can or cannot be solved by polynomial-bounded algorithms. For this reason, there exists no algorithm to date that can guarantee to terminate in polynomial time. Faster algorithms can only provide suboptimal solutions.

We provide here a suboptimal algorithm for the *Minimum Clique Partition* problem.

Algorithm 3.3.1 ***(Suboptimal Clique Partition)****: Given a non-oriented graph* $G = (V, E)$*, this algorithm builds a suboptimal clique partition of* V*. At each iteration step* i*, a new clique* K_i *is added to the partition. The algorithm goes as follows:*

while V *is not totally covered, i.e., while* $K_1 \cup K_2 \cup ... \cup K_{i-1} \neq V$*, do the following:*

- *randomly pick an element* v_i *in the set of uncovered vertices* $W_i = V \setminus (K_1 \cup K_2 \cup ... \cup K_{i-1})$*.*
- *randomly find a maximal clique* $K_i \in W_i$ *containing* v_i*.*
- *add* K_i *to the partition.*

[1] Visit http://www.claymath.org/millennium/ for a listing of the seven "Millennium Problems."

In this algorithm, a maximal clique is defined as a clique that is not a subset of a larger clique. At each iteration, a maximal clique containing a vertex v_i is found by selecting elements in W_i in random order and adding them to the clique if they are mergeable with every element in the clique. When all elements in W_i have been considered, the clique is maximal. But since the elements of w_i were selected randomly, we cannot guarantee that the resulting clique is the largest maximal clique containing v_i. An example in Figure 5 shows two situations where the algorithm, at a step i, picks the largest or a smaller maximal clique. In both pictures, the algorithm tries to find a maximal clique $K_i \in W_i$ containing σ_2 (blue vertex). Depending on whether or not the algorithm checks if σ_1 or σ_3 can be added to K_i before σ_4, the largest maximal clique $K_i = \{\sigma_1, \sigma_2, \sigma_3\}$ is selected (left picture), or the suboptimal one $K_i = \{\sigma_2, \sigma_4\}$ (right picture).

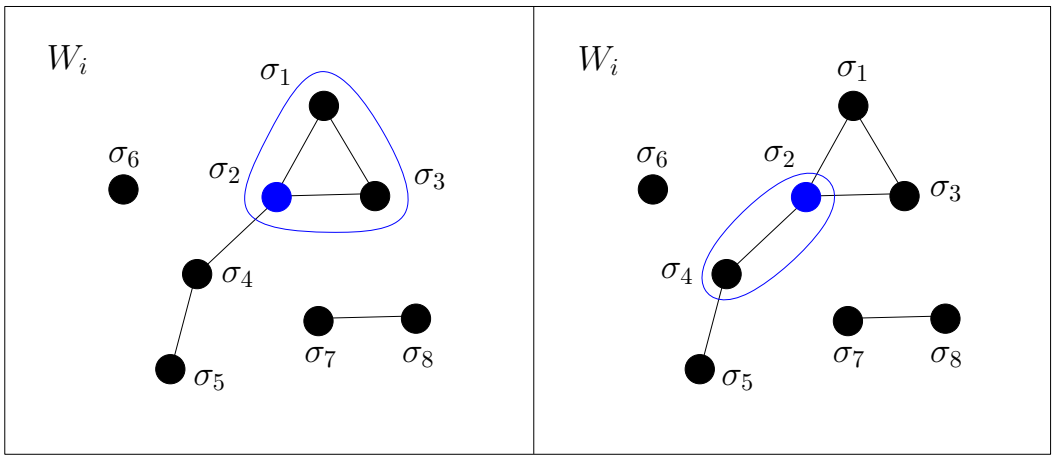


Figure 5: Maximal Cliques Picking in Algorithm 3.3.1 (example).

By always adding a maximal clique to the partition, the total number of cliques covering V remains small. However, because of the heuristic nature of this algorithm, we cannot guarantee that an optimal solution to the *Clique Partition* problem is reached. Depending on which vertices are chosen, and in which order, the algorithm can provide a significant number of suboptimal clique partitions, including the optimal ones. Figure 6 shows all possible outcomes (after symmetry considerations) of the algorithm on a simple example. Each of the six possible outcomes $\{c_i\}_{i=1}^{6}$ is represented, along with its probability $p(c_i)$ and the number of cliques $M(c_i)$. For this particular example, we see that the algorithm

would return an optimal solution ($M = 3$) with probability $p(c_2) + p(c_3) + p(c_4) = \frac{1}{2}$.

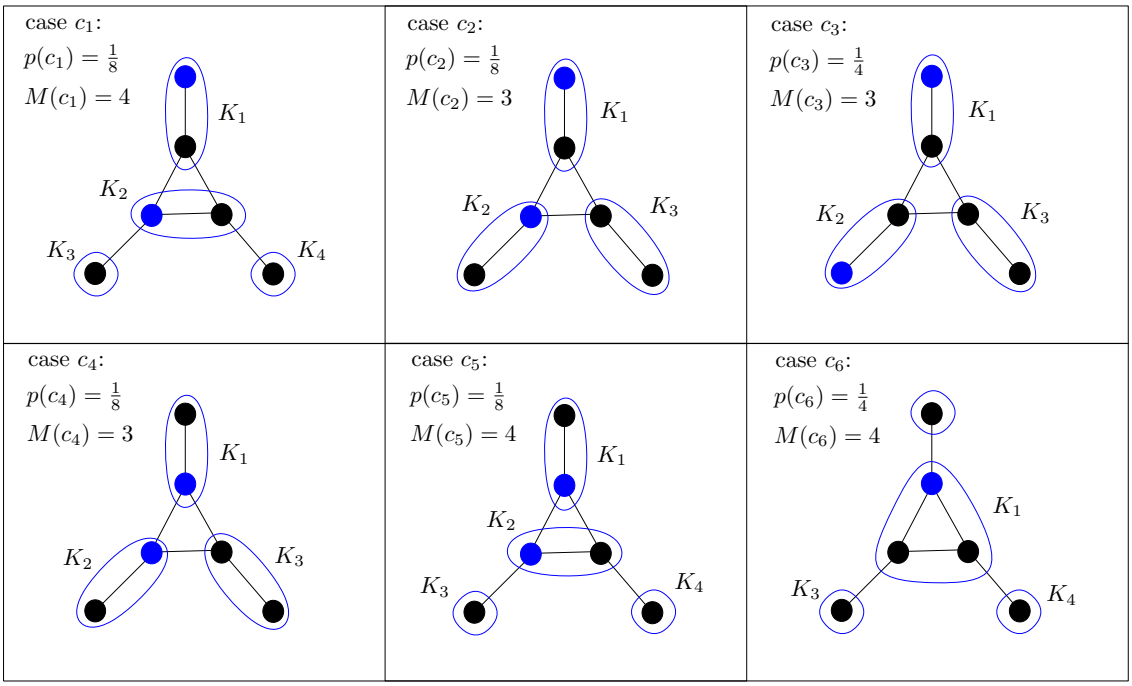


Figure 6: Possible Outcomes of the Suboptimal Clique Partition Algorithm 3.3.1 (example).

The example in Figure 6 is also interesting, as it shows that picking the largest clique (case c_6) would not guarantee an optimal solution. The problem of finding the largest clique in a graph is another NP-complete problem in Richard Karp's list [59]. Modifying our algorithm by making it search for the largest clique at every iteration would thus result in a much slower algorithm, with no guarantee of better results.

Regarding complexity, the number of operations needed for Algorithm 3.3.1 to terminate is bounded above by $\mathcal{O}(|\mathbb{Y}|L^{*2})$. Indeed, in the worst-case scenario, where all L^* modes cannot be merged to any other, the first vertex is compared to the $L^* - 1$ other vertices, the second vertex is compared to the $L^* - 2$ remaining vertices, etc. This leads to a maximum number of $\sum_{i=1}^{L^*}(i-1) = \frac{L^*(L^*-1)}{2} = \mathcal{O}(L^{*2})$ comparisons. Each of these compares the mappings k and ξ of two modes, which may require up to $2|\mathbb{Y}|$ operations. Consequently, the total number of operations required is $\mathcal{O}(|\mathbb{Y}|L^{*2})$. Algorithm 3.3.1 provides a suboptimal solution to the problem of minimizing $M(L^*)$ with satisfying results and execution time.

3.3.3.2 Minimizing $L(\sigma_{M^*})$

The problem of finding a minimum-alphabet mode sequence consistent with a given input-output string was solved in Section 3.2. Algorithms 3.2.1 and 3.2.2 provide solutions σ_{M^*} to the problem. However, whereas M is indeed minimized, the length L of the resulting mode sequences is maximum, equal to the length N of the input-output string. Here, we address this issue by removing as many mode occurrences as possible, hence reducing the length $L(\sigma_{M^*})$ while keeping the alphabet size M^* constant.

First, we recall that the two algorithms presented earlier generate Always Interrupt Sequences. Hence, by definition, the interrupt function of each mode triggers whatever the value of the output $y \in \mathbb{Y}$, i.e., $\xi_m(y) = 1$ $(m = 1, ...M^*)$. The idea is to modify the interrupt function ξ of each distinct mode and make it equal to zero (i.e. no mode change) whenever possible. Ideally, a mode sequence like $\sigma = \sigma_1\sigma_1\sigma_2\sigma_2\sigma_2\sigma_1\sigma_1\sigma_1\sigma_2\sigma_2$ could then be reduced to $\sigma = \sigma_1\sigma_2\sigma_1\sigma_2$. To conserve consistency with the input-output string, we need to carefully identify when interrupt functions can or cannot be modified. We suggest the following algorithm:

Algorithm 3.3.2 *($L(\sigma_{M^*})$-Reduction) Given an Always Interrupt Sequence (AIS) $\sigma_{M^*} = \sigma_{m(1)}, \sigma_{m(2)}, ...\sigma_{m(N)}$ consistent with an input-output sequence $S = (y(1), u(1)), ..., (y(N), u(N))$, we reduce the length of σ_{M^*} as follows:*
For all modes $m = 1, ..., M^*$,

- *For all $y \in \mathbb{Y}$, if there exist a time $i \in 1, ...N$ such that $y(i) = y$, $m(i) = m$ and $m(i+1) \neq m$, keep $\xi_m(y) = 1$.*
- *Change all the other values of $\xi_m(y)$ to zero.*

This algorithm looks for mode switches from one mode to another distinct mode. These interrupts are the only ones that need to be kept to preserve consistency. By turning off all other interrupts to zero, the length of the mode sequence can be significantly reduced. Obviously, the resulting mode sequence is not an AIS anymore. But since no new mode

is added, it also uses the minimum number M^* of distinct modes, i.e., it also belongs to $sol(P_M(\mathcal{C}(S)))$.

Example. Consider the following input-output string and a consistent Always Interrupt Sequence σ_{M*} using $M^* = 2$ distinct modes:

y	1	0	2	0	2	2	1	2	2	0	1	2
u	0	1	1	0	0	0	1	1	0	1	1	1
σ_{M*}	σ_1	σ_2	σ_2	σ_1	σ_1	σ_1	σ_2	σ_2	σ_1	σ_2	σ_2	σ_2
		↑		↑			↑		↑	↑		

The arrows under the table show switches from one mode to a distinct one. The corresponding interrupts $\xi_1(1) = 1$, $\xi_2(2) = 1$ and $\xi_1(2) = 1$ must be kept. All other interrupts $\xi_1(0)$, $\xi_2(0)$ and $\xi_2(1)$ can be changed to zero. By doing so, the mode switches occurring at instants $i = 2, 4, 7, 10,$ and 11 are suppressed, resulting in a shorter mode sequence $\sigma_{M*} = \sigma_1\sigma_2\sigma_1\sigma_1\sigma_2\sigma_1\sigma_2$. From a mode sequence with length 12 and specification complexity 8.15 *nats*, we get a simpler one (yet still consistent) with shorter length 7 and smaller specification complexity 4.78 *nats*.

3.4 Application: What Are the Ants Doing?

In this example, we observe 10 roaming ants (*Aphaenogaster cockerelli*) in a tank, and we intend to extract motion behaviors from some collected data. Different mode sequences are generated using the methods presented in this chapter, and the performance of each algorithm is compared to the others. The recovered mode sequences are then used to control a collection of Free-Running Feedback Automata. In addition to being a fun subject of study, ants are a rich source of inspiration in many robotic applications because of their fascinating skills in navigation, exploration, and organization.

3.4.1 Experimental Setting

The experimental setting is as follows: 10 ants are placed in a tank with a camera mounted on top, as seen in Figure 7. A 48-second movie is recorded and a vision-based tracking software computes the position (x, y) and orientation θ of each ant every 33 ms.

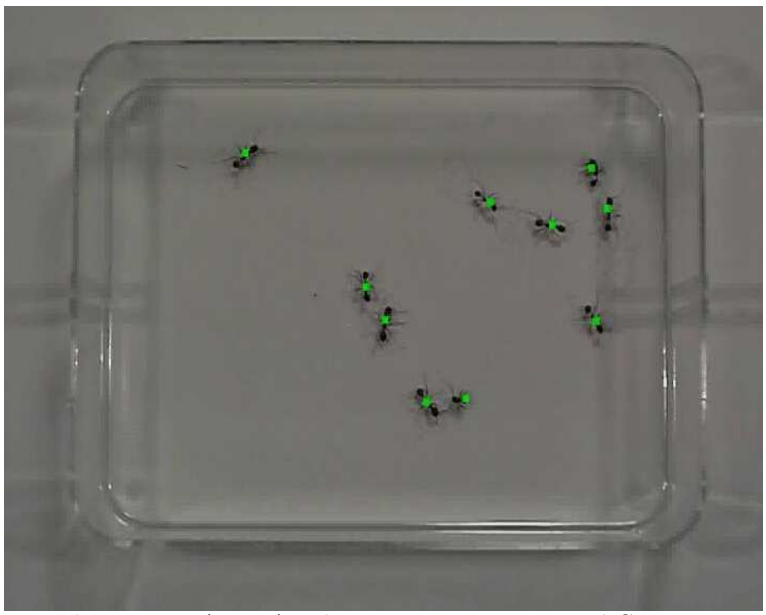


Figure 7: Ants Application: Experimental Setting.

This experimental data is kindly provided by Tucker Balch and Frank Dellaert at the Georgia Institute of Technology BORG Lab[2].

[2]http://borg.cc.gatech.edu.

3.4.2 Data Processing

A first task consists of processing the available data (position and orientation of each ant) and turning it into a quantized input-output string. Here, the choice of relevant input and output variables is crucial if we want to capture true ant-like behaviors. For example, the proximity of an obstacle or another ant probably plays an important role in an ant's decision process. The room temperature is most likely irrelevant within some bounds. To be successful, our mode recovery approach hence requires some physical insight into the underlying data-generating process. Assuming relevant input and output variables have been selected, their quantization can also play an important role in the recovery of true ant-like behaviors. This aspect of the problem will be studied in Chapter 4. For now, a quantized I/O string is computed for each ant by considering the following inputs and outputs:

- the input at time k is a two-dimensional point $u(k) = (u_1(k), u_2(k))$, where
  - $u_1(k)$ is the quantized angular velocity of the ant,
  - $u_2(k)$ is the quantized translational velocity of the ant,
- the output at time k is a three-dimensional point $y(k) = (y_1(k), y_2(k), y_3(k))$, where
  - $y_1(k)$ is the quantized angle to the closest obstacle in an egocentric referential,
  - $y_2(k)$ is the quantized distance to the closest obstacle,
  - $y_3(k)$ is the quantized angle to the closest goal in an egocentric referential.

In these definitions, an *obstacle* is either a point on the tank wall or an *already visited* ant within the visual scope of the ant, and a *goal* is an ant that has not been visited recently. Ant behaviorists usually describe *encounters* as the occurrence of two ants approaching each other and experiencing a brief antennal contact. Since our data does not provide enough information for identifying when such events occur, we say that two ants have visited each other once they have been within a small distance ϵ of each other. Figure 8 depicts the

visual scope (blue), obstacles (red), and goals (green) seen by each ant at the same instant shown in Figure 7.

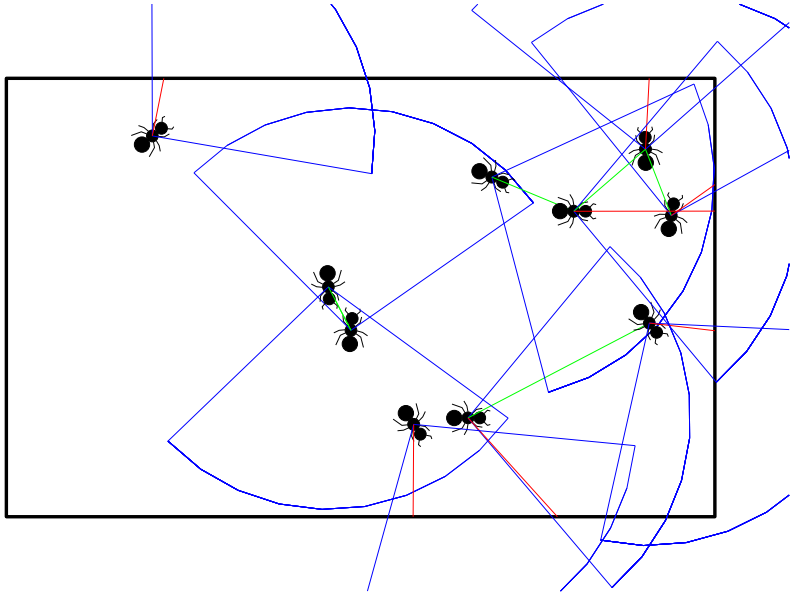


Figure 8: Ants Application: I/O String Generation.

In this example, each signal, $u_1(k), u_2(k), y_1(k), y_2(k)$, and $y_3(k)$, is quantized using five possible values. This gives 25 possible inputs and 125 possible outputs.

3.4.3 Comparison of Mode Recovery Methods

For each ant, three mode sequences, consistent with the input-output string, are first recovered:

- σ_1 is a mode sequence with minimum length. It is recovered using the algorithm developed in [7] and reproduced in section 3.1.

- σ_2 is a mode sequence with minimum alphabet. It is an Always Interrupt Sequence recovered using Algorithm 3.2.1.

- σ_3 is another Always Interrupt Sequence with minimum alphabet, but also has minimum entropy (on the set of AIS only). It is recovered using Algorithm 3.2.2.

Each of these mode sequences is then transformed, bringing the total number of mode sequences to six (per ant):

- σ_1' is obtained from σ_1 by reducing the number of distinct modes (alphabet size), using Algorithm 3.3.1.

- σ_2' is obtained from σ_2 by reducing the length of the mode sequence, using Algorithm 3.3.2.

- σ_3' is obtained from σ_3 by reducing the length of the mode sequence, using Algorithm 3.3.2.

Results, including string length L, number of distinct modes M, entropy $\mathcal{H}$, and specification complexity $\mathcal{S}$ for each of the six recovered mode sequences and for each of the 10 ants are given in Table 2.

In this table, results within a starred box stress the fact that the enclosed variable is optimal:

- the length L is minimum for σ_1 and σ_1',

- the alphabet size M is minimum for σ_2, σ_2', σ_3, and σ_3'.

- the entropy $\mathcal{H}$ is minimum for σ_3 (but here, only on the set of Min$M$ consistent Always Interrupt Sequences).

The relation signs '$>$', '$=$' and '$<$' show how the variables L, M, $\mathcal{H}$, and $\mathcal{S}$ are affected by the L-reduction and M-reduction transformations:

- in the case of σ_1, the M-reduction reduces M while keeping L constant. This alphabet reduction is followed by a reduction both in $\mathcal{H}$, and $\mathcal{S}$, as proven in Theorem 3.3.1.

- in the case of σ_2 and σ_3, the L-reduction reduces L while keeping M constant. This length reduction is followed by an increase in $\mathcal{H}$ but a decrease in $\mathcal{S}$. Whereas the complexity reduction is proven in theorem 3.3.2, the same theorem shows that the increase in entropy cannot be generalized.

Table 2: Ants Application: Comparison of Low-Complexity Mode Recovery Procedures.

	$\sigma_1 \xrightarrow{M\text{-reduction}} \sigma'_1$			$\sigma_2 \xrightarrow{L\text{-reduction}} \sigma'_2$			$\sigma_3 \xrightarrow{L\text{-reduction}} \sigma'_3$		ant#
L	66	=	66 *	145	>	138	145	137	1
	103		103	145		138	145	140	2
	120		120	145		142	145	141	3
	60		60	145		119	145	120	4
	110		110	145		142	145	141	5
	90		90	145		140	145	141	6
	113		113	145		142	145	143	7
	110		110	145		140	145	141	8
	118		118	145		140	145	142	9
	100		100	145		139	145	144	10
M	66	>	16	11	=	11	11	11 *	1
	103		26	16		16	16	16	2
	120		26	21		21	21	21	3
	60		19	14		14	14	14	4
	110		30	20		20	20	20	5
	90		26	17		17	17	17	6
	113		25	19		19	19	19	7
	110		32	22		22	22	22	8
	118		25	20		20	20	20	9
	100		28	17		17	17	17	10
$\mathcal{H}$	6.04	>	3.19	2.38	<	2.41	2.01 *	2.08	1
	6.69		3.37	3.31		3.32	2.82	2.86	2
	6.91		3.99	3.86		3.88	3.63	3.67	3
	5.91		4.09	2.93		3.18	2.59	2.87	4
	6.78		4.40	3.85		3.85	3.72	3.72	5
	6.49		4.19	3.55		3.54	3.34	3.35	6
	6.82		4.15	3.79		3.80	3.54	3.56	7
	6.78		4.50	4.00		4.01	3.86	3.86	8
	6.88		4.09	3.87		3.89	3.64	3.67	9
	6.64		4.15	3.49		3.51	3.45	3.45	10
$\mathcal{S}$	398.9	>	210.4	344.5	>	332.1	291.1	285.2	1
	688.7		347.6	480.3		457.6	409.4	400.2	2
	828.8		479.0	560.0		550.3	526.3	516.8	3
	354.4		245.4	424.7		378.4	376.2	344.7	4
	745.9		483.7	557.6		547.1	539.0	525.1	5
	584.3		377.4	515.0		495.2	483.8	472.0	6
	770.7		469.0	549.2		540.2	512.6	508.7	7
	745.9		495.4	580.4		561.8	559.9	544.5	8
	812.2		482.2	561.5		544.1	527.5	520.6	9
	664.4		414.8	506.2		488.1	499.5	496.4	10

The most important results concern the specification complexity $\mathcal{S}$. Before reduction, the Always Interrupt Sequences σ_2 and σ_3 have a lower complexity than the minimum-length mode sequence σ_1. On average, the 10 mode sequences σ_1 are respectively 28% and 39% more complex than their σ_2 and σ_3 equivalents. Also, the fact that σ_3 sequences are on average 7% less complex than σ_2 sequences justifies the extra effort spent in minimizing the entropy over the set of Min$M$ AIS. After reduction, the tendency is reversed as σ'_1 performs

on average 20% and 14% better than σ_2' and σ_3'. Whereas the M-reduction of σ_1 reduces the number of modes by 74% and the complexity by 39% on average, the L-reduction of σ_2 and σ_3, although optimal, does not reduce the length by more than 5% and the complexity by more than 4% on average.

The choice of a particular recovery method depends on the targeted application. One might prefer producing short control programs from data or longer programs using fewer modes. In both cases, the algorithms developed in this chapter result in low-complexity control programs. If the specification complexity is the only performance index considered, then finding the shortest consistent mode sequence using the Dynamic Programming Algorithm in [7] and then reducing its alphabet using Algorithm 3.3.1 seems to be a better choice. In other applications, the size of the alphabet may play a greater role. Note how, in Definition 2.3.1, the complexity measure assumes that the alphabet is already embedded in the system receiving the control program. We can easily imagine applications where, in addition to the control program, the mode definitions would also have to be sent to a robot through a communication channel. In such applications, a more appropriate complexity measure would favor control programs with small alphabet size, and mode sequences such as σ_2' and σ_3' would be preferred.

CHAPTER IV

EXECUTION AND EXPRESSIVENESS

In this chapter, we show how control programs such as the ones recovered in Chapter 3 can be effectively executed on real systems or in a computer simulation. We also investigate how the behaviors of the original system and the simulated one can be compared and speculate on the reasons behind any dissimilarity. In particular, we analyze the effect of different quantization methods.

4.1 From Mode Sequences to Fully Executable Control Programs

In Figure 9, we recall how, within the Motion Description Languages framework, a Free-Running Feedback Automaton is controlled by a mode sequence.

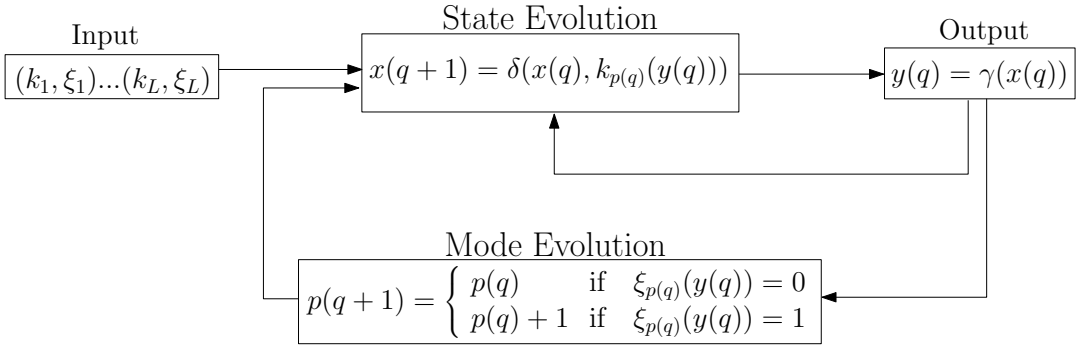


Figure 9: Free-Running Feedback Automaton.

At every time-step q, the control input applied to the system is $k_{p(q)}(y(q))$, the value of the feedback for the current mode $p(q)$ and current value of the output $y(q)$. In our approach, where modes are built by mapping finite portions of I/O strings, it is conceivable that behaviors may not be defined on the whole output space. In other words, without some type of persistency of excitation (i.e. if the training data set is too small), we may run into situations where the control input value $k_{p(q)}(y(q))$ is undefined. As such, the mode sequences recovered in Chapter 3 cannot be directly executed on MDL-devices without a

few modifications. For example, consider the case where we recovered the mode (k, ξ) and where the available empirical data only allows us to define the domain of k and ξ as a proper subset of the total output space $\mathbb{Y}$, denoted here by Y_k or Y_ξ. From the construction of the modes, these two subsets are always identical, i.e., $Y_k = Y_\xi$. There are many ways in which one can ensure that whenever this mode is active and an output value $y \in \mathbb{Y} \setminus Y_k$ is encountered, i.e., whenever $k_{p(q)}(y(q))$ is undefined, a control input value is still computed. We list here a few of them:

- If $y(q-1) \in Y_k$ but $y(q) \notin Y_k$, we can replace $k(y(q))$ with $k(y(q-1))$ as well as let $\xi(y(q)) = 0$. As one might expect, this easy approach can sometimes produce undesirable behaviors, such as robots moving around indefinitely in a circular motion.

- If $y(q) \notin Y_k$, but $y(q) \in Y_{\tilde{k}}$ for some other mode pair $(\tilde{k}, \tilde{\xi})$ in the recovered mode sequence, we can let $k(y(p))$ be given by the most recurrent input symbol $\tilde{u} \in U$ such that $\tilde{k}(y(q)) = \tilde{u}$. This method works as long as $y(q)$ belongs to the domain of at least one mode in the sequence. If this is not the case, additional choices must be made.

- If $y(q)$ does not belong to the domain of any of the modes in the sequence, we can introduce a norm on Y, and pick $\tilde{y}$ instead of $y(q)$, where $\tilde{y}$ minimizes $\|y(q) - \tilde{y}\|_{\mathbb{Y}}$ subject to the constraint that $\tilde{y}$ belongs to the domain for at least one mode in the sequence.

Note that all of these choices are heuristic in the sense that there is no fundamental reason for choosing one over the other. Rather, they should be thought of as tools for going from recovered mode strings to executable control programs.

An example of a computer simulation is given in Figure 10. Here, 10 mode sequences recovered from the observation of ants in Section 3.4 are executed on 10 Free-Running Feedback Automata. The underlying system modeling ant locomotion is a unicycle, controlled by translational and angular velocities. The figure shows a snapshot from a 30-second movie where, for more realism, each agent is drawn as an ant with appropriate position and orientation.

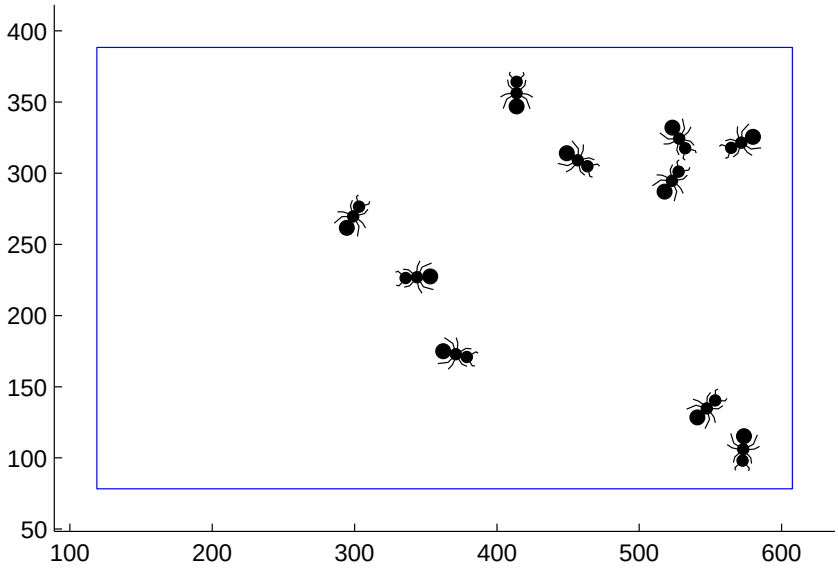


Figure 10: Ants Application: Simulating Ten Virtual Ants.

4.2 Performance

Now, assuming that recovered mode sequences can be efficiently used as control programs, we are interested in comparing the *simulated* system with the *original* system. By *performance*, we very informally understand the ability of the former to reproduce the behaviors of the latter. To make this observation more precise, one needs to provide a measure of how well the simulated system can reproduce the behavior of the original system. Of course, the choice of a particular measure should be motivated by the targeted application. For example, in the case of the ants, if we were to produce a simulation with thousands of ants for a scene in an animation movie, the simulated ants would have to pass a very subjective (qualitative) "eye"-test, meaning we would just be concerned by how "real" the virtual ants look, e.g., in the way they approach, avoid, follow each other, etc. In a robot application, we could use more objective arguments such as "did the robot bump into an obstacle or not?" and more quantitative measures, e.g., the time required to reach a target location. In most applications, the reason why it is not possible to compare two trajectories using common measures such as the mean squared error (or any norm on the space where the two

trajectories belong) is that the systems under consideration are highly reactive. Whereas this is just an assumption concerning the original (unknown) system, the statement makes no doubt for the simulated system which, by construction, exhibits feedback control. Consequently, in an unknown and/or changing environment, no two trajectories will look the same and a mean squared error measurement will have little relevance.

More insight can be brought to the problem if we consider the following three systems:

- system (a): the original system with unknown dynamics, e.g., an ant. From this system, a finite input/output string is extracted using both time sampling and signal quantization. The choice of the inputs and outputs is driven by the available controllers and sensors on the computer-controlled machine M that will be used to reproduce the original system's behavior. Of course, when the execution is restricted to a computer simulation, there is much more flexibility in the choice of the input and output features.

- system (b): the machine M, with known dynamics, controlled by the zero-order hold quantized input string measured on (a), as an open loop control. In our ant application, this could correspond to a unicycle described by $\dot{x}(t) = v(t)\cos\theta(t)$, $\dot{y}(t) = v(t)\sin\theta(t)$ and $\dot{\theta}(t) = \omega(t)$, controlled by $v(t) = u_1(k)$ and $\omega(t) = u_2(k)$ for $kT \leq t < kT+T$ where T is the sampling period. Here, $u_1(k)$ and $u_2(k)$ represent the quantized velocity and angular velocity (inputs) measured from the ant trajectory.

- system (c): the same machine M, but this time, controlled by a MDL sequence consistent with the input-output string measured on (a). This system has the same dynamic equations as (b) but the control here is a feedback control. For the unicycle, this means that the controls $u_1(k)$ and $u_2(k)$ are given by the output-input map of the active mode at time k.

As defined, the performance is a measure of how well the trajectory of system (c) approximates the one of (a). To achieve a perfect reproduction, we would need for (c) to perfectly reproduce (b) and for (b) to perfectly reproduce (a). For this reason, the overall performance can be thought of as the combination of two subsystems' performance.

First, for (c) to exactly track (b), it would require that the Free-Running Feedback Automaton controlling (c) produces the exact same input string as the one used to control (b). This condition is tied down by the use of feedback control in (c) and is obviously hard to achieve in a changing environment. When (c) encounters a different output value than the one computed from (a), the feedback mapping in (c) may provide a different input control value than the one used in (b), resulting in a local variation in state trajectories between (b) and (c). Because of this divergence, the output values read at the next time increment will most likely be different and probably produce distinct input values again. In different situations, system (c) may rectify its trajectory or use a completely different one. In this context, measuring performance on the quality of the tracking may not be very relevant. Obviously, different mode sequences recovered from the same input-output string should produce distinct state trajectories. The performance we consider here is really the performance of a given mode string. However, how different mode recovery methods affect performance is difficult to capture, even qualitatively.

On the other hand, the performance of system (b) at tracking (a) is easier to deal with. Since the control input to (b) is a zero-order hold quantized input string measured on (a), the quality of the fit is directly related to the quality of the sampling/quantization process applied to the input. By decreasing the sampling period T and increasing the number of quantization levels q_U, we can generate an arbitrarily good approximation of the input function, and consequently, of the state trajectory. However, the length of the input(-output) string is inversely proportional to the sampling period, and the number of bits necessary for encoding one input value increases with q_U. Hence, performance is achieved at the expense of complexity. Figure 11 illustrates this performance/complexity duality statement. For both pictures, points in the performance-complexity plane were computed from real biological data. The recorded velocity of an ant was sampled using various sample periods T and quantization levels q_U. In the left picture, the data points were computed for various q_U but for a constant sample period ($T = 33ms$). In the right picture, the data points were computed for various T but for a constant number of quantization levels ($q_U = 8$). Here, the performance was measured as the inverse of the input signal distortion

(mean squared error between the original input and the zero-order hold input string), and the complexity was measured as the number of bits required to encode the input string (i.e. proportional to $1/T \log_2(q_U)$). The units and order of the curves depend greatly on the measure choices, but what most concerns us here is their tendency.

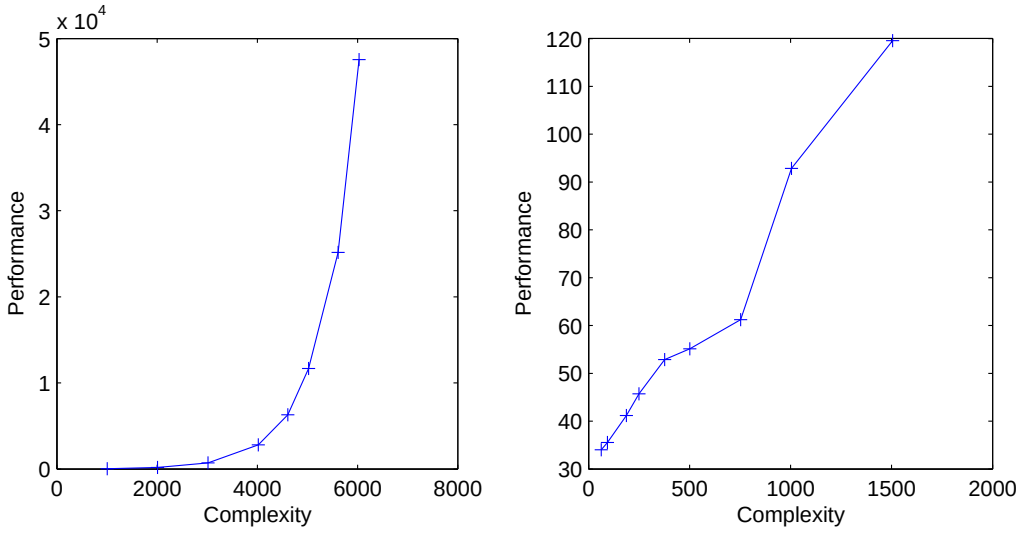


Figure 11: Complexity vs. Performance.

4.3 Quantization Methods

In this section, we present four different scalar quantization strategies and compare them in terms of their performance and complexity. First, we introduce some notations:

Definition 4.3.1 *A* scalar quantizer *is a staircase function that maps input values in* $\mathbb{R}$ *into a finite set* $Q = \{q_1, q_2, ..., q_N\}$ *of real numbers called* quantization levels. *The range of the input values is divided into* N *adjacent intervals, with boundaries at the* threshold levels $t_0, t_1, ..., t_N$. *When the input value belongs to the* i^{th} *interval* (t_{i-1}, t_i), *it is mapped (quantized) to the* i^{th} *quantization level* q_i.

Now, assuming a given string of input values $X = \{x(1), ..., x(z)\}$, the four quantizers under consideration are defined as follows:

[Q_1] uniform quantizer: the range $[x_{min}, x_{max}]$ of the input values is divided into N equal intervals. The length $\Delta = (x_{max} - x_{min})/N$ of each of these intervals is called

quantization step size. The threshold levels are $t_i = x_{min} + i\Delta$, $i = 0, ..., N$. For each interval (t_{i-1}, t_i), the corresponding quantization level q_i is the mean value of all the data points from X falling within this interval.

[Q_2] equally-distributed quantizer: the threshold levels are chosen so that each of the N intervals contains the same number of elements from X (plus or minus one for when z is not a multiple of N). To do so, the elements in X (all assumed distinct) are first sorted. Assuming k and r are, respectively, the quotient and the remainder of the division of z by N, the sorted input values are split into r groups of $k+1$, and $N-r$ groups of k. Each t_i $(i = 1, ..., N-1)$ is set at the mean value between the last value of the i^{th} group and the first value of the $(i+1)^{\text{th}}$ group. Also, t_0 is the value of the first element in the first group, i.e., x_{min}, and t_N is the value of the last element in the last group, i.e., x_{max}. Finally, the quantization level q_i $(i = 1, ..., N)$ is the mean of the values in the i^{th} group.

[Q_3] optimal-PCM quantizer: this quantizer uses quantization thresholds and quantization levels minimizing the mean squared error $MSE = \sum_{j=1}^{r}(x(j) - q(x(j)))^2$. Necessary conditions for optimality were formulated by Lloyd in 1957 [64], along with an effective algorithm for computing the optimal solution.

[Q_4] optimal-DPCM quantizer: this quantizer falls in the category of *predictive* quantizers, where an educated guess is made about the present value $x(i)$ of the signal, based on past signal transmissions $x(i-1)$, $x(i-2)$, ..., $x(i-m)$, for some positive integer m. Given a linear *predictor* $[p(1), p(2), ..., p(m)] \in \mathbb{R}^m$, the predicted value for $x(i)$ is $y(i) = p(1)x(i-1) + p(2)x(i-2) + ... + p(i)x(i-m)$. Instead of quantizing $x(i)$, the predictive quantizer quantizes the error between the predicted value $y(i)$ and the actual value $x(i)$. The integer m above is called the *predictive order*. In an optimal-DPCM, the data string X is used as a training data for computing an optimal predictor, i.e., one that will minimize the signal distortion.

The performance of these quantizers is analyzed in Figure 12 where the mean squared error, or *signal distortion*, is computed for the quantization of five distinct signals: a simple

sinusoid, a random signal with uniform probability distribution over $[-1, 1]$, two gaussian noises with zero-mean and standard deviations $1/3$ and $1/5$, and the recorded angular velocity of an ant (biological data). The time evolution and distribution of these signals (before quantization) are represented along with the measured distortion for each of the four quantizers. In order to compare the methods, a same number of quantization levels (eight) was used in every case.

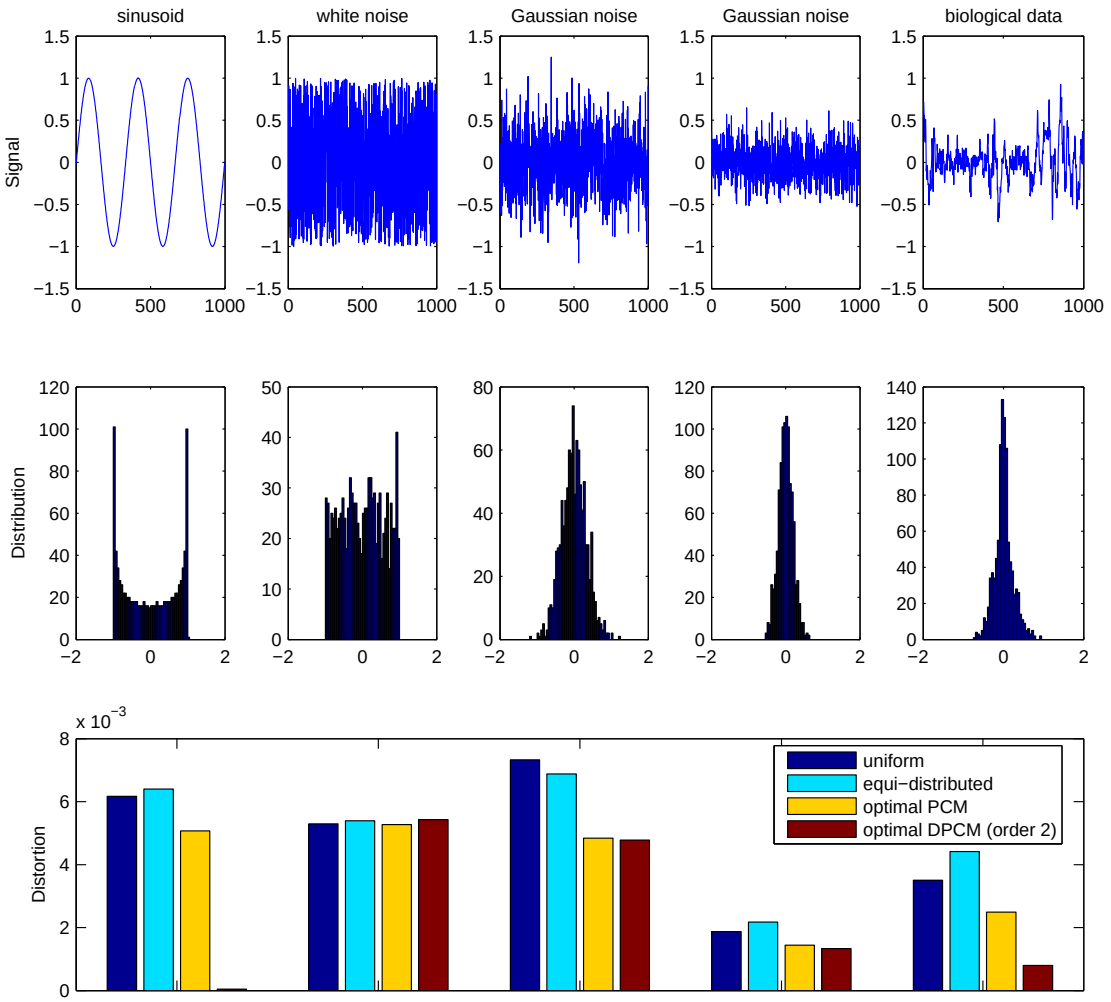


Figure 12: Performance of Four Quantization Methods on Various Signals.

In view of these results, we can conclude that, regarding distortion, the optimal PCM and optimal DPCM quantizers generally perform better. Of these two, the optimal DPCM quantizer seems to be the best suited when the data does not come from the execution of a

random variable. Also, the relative performance of the quantizers seems to be correlated to the signal distribution. In fact, as the distribution becomes more unevenly distributed, the difference in distortion between the optimal quantizers and the equi-distributed quantizer on one side, and between the equi-distributed quantizer and the uniform quantizer on the other side, increases.

Next, the four quantizers are compared in terms of complexity. The objective is to get a feeling for how different quantization methods can affect the outcome of the mode recovery process studied so far. To this end, we run the same experiment as in Section 3.4. The available ant data is quantized using each of the four quantizers. For each of these, 10 mode sequences are recovered from 10 quantized I/O strings (one for each of the 10 ants) using the "MinL-LowM" recovery method. The specification complexity (Definition 2.3.2) of each mode sequence is represented in Figure 13 against the distortion in the quantized input string ('+' signs). For each quantization method, the average specification complexity vs. average distortion is also represented (big dot).

A first remark is that the choice of a particular quantization method can significantly affect the complexity and performance of the recovered control programs. Also, Figure 13 illustrates again what was pointed out earlier: the impossibility of minimizing both quantities at the same time. The optimal PCM and DPCM, which provide the best distortion, perform badly in terms of complexity. On the other hand, the equi-distributed quantization method gives the best specification complexity, but at the expense of a poor signal distortion. To understand the differences in terms of complexity, one should interpret the quantizers as filters bringing more or less redundancy to the data. Since the underlying reason for a mode switching is the existence of two distinct input values for the same output value, a better repartition of these input and output values can reduce the probability of such situations.

To conclude, we underlined the existence of a tradeoff between the complexity of the recovered control programs of Chapter 3 and their performance when applied to real systems. We also showed how the quantization process can play an important role in this tradeoff. In particular, the choice of a quantization method, the sampling period, and the number of

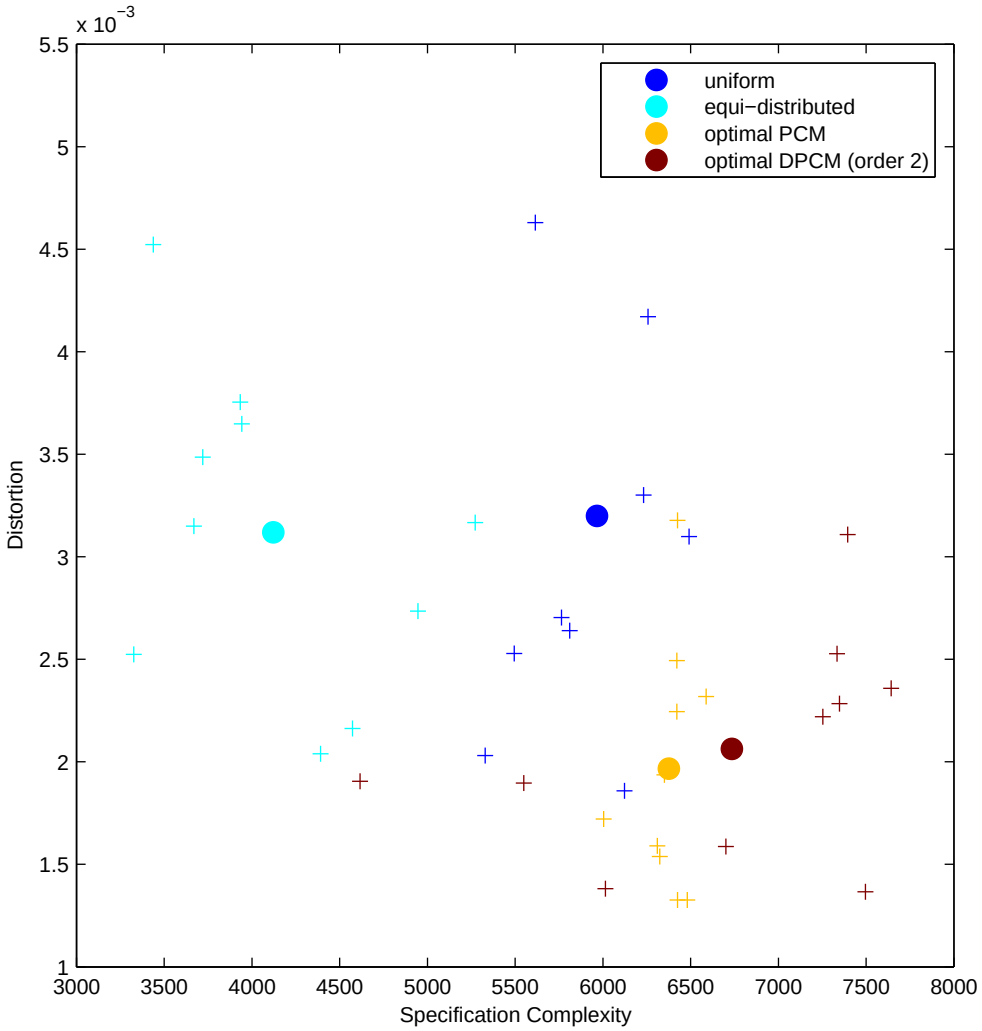


Figure 13: Performance vs Complexity Plot for Four Quantization Methods.

quantization levels can be used as tuning parameters for achieving a desired level (ratio) of complexity/performance.

CHAPTER V

FROM MODE STRINGS TO EXECUTABLE AUTOMATA

In Chapter 3, we developed methods for obtaining mode sequences from empirical data in an automated way. However, once such a mode sequence has been obtained, one would like to produce a compact representation of the underlying switching logic that describes the high-level operation of the system. In fact, mode sequences (or strings) can be thought of as sample paths generated by an underlying system, and in this chapter we set out to recover this system in a computationally tractable manner. More specifically, we assume that a mode sequence $k_{i_1}, \xi_{j_1}, k_{i_2}, \xi_{j_2}, \ldots$ is the input-output path resulting from the execution of a finite automaton, where k_{i_q} is the "output" from the underlying finite automaton in state s_q, and ξ_{j_q} is the corresponding "event" that triggers a transition from state s_q to s_{q+1}. In this chapter, we are interested in recovering this underlying automaton. Moreover, as there are possibly many such automata, we will focus our attention on trying to recover minimal automata, i.e, automata with minimum state space cardinality. Figure 14 shows an example of a mode sequence and the corresponding minimal automaton.

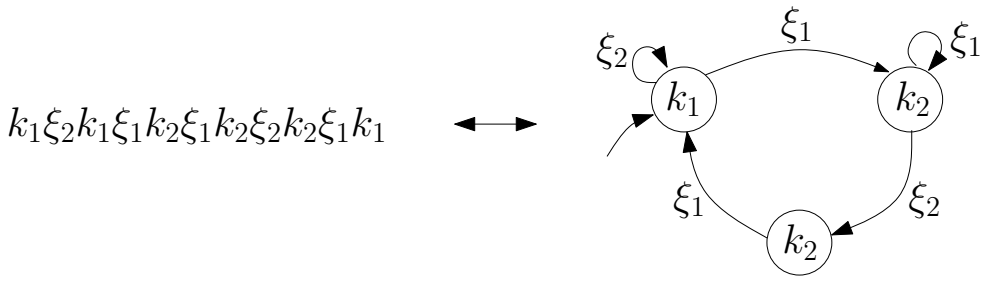


Figure 14: Example of a Mode Sequence and Corresponding Minimal Input-Output Automaton.

Note that even though the mode strings describe an execution of a hybrid, multi-modal system, the problem of capturing the lowest complexity system for describing the switching logic is in fact a problem purely in the area of finite automata theory.

The chapter's organization is as follows. In Section 5.1, a preprocessing step is applied

to the mode sequence, in which the association $k \sim \xi$ of each mode is broken. The motion alphabet is thus split into two alphabets, one for the feedback mappings and one for the interrupt functions. A new alphabet reduction algorithm is then applied to both alphabets in order to reduce the number of distinct symbols. The problem of recovering an automaton is defined in Section 5.2, together with a definition of system complexity (cardinality of the state space), which is the performance criterion that our recovered system should minimize. An explicit algorithm for finding the lowest complexity automaton that can generate the sample path is given in Section 5.3 together with a study of its computational complexity. In fact, the proposed algorithm is found to be superexponential in the worst case, and as such, it is of dubious computational relevance. Instead, lower complexity (in some cases suboptimal) algorithms are presented, and their performance is evaluated against a large-scale example.

5.1 Merging Feedback Mappings and Interrupt Functions

The conversion from mode sequences to executable I/O automata requires splitting the motion alphabet Σ into two alphabets: the input alphabet Ξ of interrupt functions and the output alphabet $\mathcal{K}$ of feedback mappings. In order to produce low complexity automata, a preliminary task consists of reducing the size of Ξ and $\mathcal{K}$ by merging combinations of elements that look "similar." In order to do so, we first need to define a measure of "similarity." Consider merging n distinct feedback mappings[1] $k_{i_1}, k_{i_2}, ..., k_{i_n}$, resulting in the creation of a "macro-feedback mapping" K_I, where we let $I = \{i_1, ..., i_n\}$. Note that for a given $y \in \mathbb{Y}$, two distinct feedback mappings may map y to two different inputs, i.e., $\exists(\alpha, \beta) \in I^2$ such that $k_\alpha(y) = u_i$ and $k_\beta(y) = u_j$ with $u_i \neq u_j$. For this reason, we choose to represent $K_I(y)$ as a random variable. In this respect, the resulting "macro-feedback mapping" K_I is a random process defined on $\mathbb{Y}$.

Now, for a given $y \in \mathbb{Y}$, the probability mass function of $K_I(y)$ can be recovered by

[1]Here we choose to merge feedback mappings but the same ideas, definitions and algorithm apply for interrupt functions.

$$p_{K_I(y)}(u) = \Pr\{K_I(y) = u\} = \frac{card\{q|y_q = y,\, u_q = u \text{ and } m(q) \in I\}}{card\{q|y_q = y \text{ and } m(q) \in I\}},$$

where $m(q)$ refers to the mode active at time q and card denotes cardinality. Next, we define the entropy of the random variable $K_I(y)$:

$$H(K_I(y)) = -\sum_{u \in \mathbb{U}} p_{K_I(y)}(u) \log(p_{K_I(y)}(u)).$$

It is easily established that this entropy is bounded by

$$0 \leq H(K_I(y)) \leq \log(n).$$

Finally, we define an entropy measure for the random process K_I. It is the normalized and weighted average of the entropies of all random variables $K_I(y)$, when y describes $\mathbb{Y}$:

$$H(K_I) = \frac{1}{\log(n)} \sum_{y \in \mathbb{Y}} p_Y(y) H(K_I(y)).$$

Note that in this definition, the output is also considered a random variable Y. Similarly, its probability mass function $p_Y : y \to p_Y(y) = \Pr\{Y = y\}$ can be estimated from the available data:

$$p_Y(y) = \frac{card\{q|y_q = y,\ \text{and } m(q) \in I\}}{card\{q|m(q) \in I\}}.$$

The total entropy $H(K_I)$ is a measure of the average uncertainty in the random process K_I. It varies from zero to one, where

- $H(K_I) = 0$ means that there is no uncertainty in K_I, i.e., for all $y \in \mathbb{Y}$, all modes in I map y to the same input value.
- $H(K_I) = 1$ means that the uncertainty in K_I is maximal, i.e., for all $y \in \mathbb{Y}$, all modes are equally active and they all map y to a distinct input value.

In other words, the two extreme values are reached when the n feedback mappings are either equal ($H(K_I) = 0$) or completely different ($H(K_I) = 1$). We propose to define a threshold value $\gamma_k \in [0, 1]$ so that if $H(K_I) \leq \gamma_k$, the feedback mappings are considered similar enough to be merged.

Now that a criterion for mergeability has been defined, the alphabet reduction problem is the same as the one defined in section 3.2. It consists of finding a minimum collection of disjoints sets K_{I_1}, K_{I_1}, ..., K_{I_M} that form a partition of $\mathcal{K}$, i.e., $\mathcal{K} = K_{I_1} \cup K_{I_2 \cup ... \cup K_{I_{M^*}}}$. Also, each set of the partition should only contain mergeable elements, i.e., $H(K_{I_i}) \leq \gamma_k$ for $i = 1, ..., M^*$. Recall that the problem is NP-complete but that heuristic suboptimal algorithms such as Algorithm 3.3.1 can provide a pretty good approximation in polynomial time. This algorithm is rewritten here using the new notation:

Alphabet Reduction Algorithm:

Given an alphabet $\mathcal{K} = \{k_{i_1}, k_{i_2}, ..., k_{i_n}\}$ and a reduction threshold γ_k, we reduce $\mathcal{K}$ in the following manner:

while $\mathcal{K}$ is not totally covered, i.e., while $K_{I_1} \cup K_{I_2} \cup ... \cup K_{I_{i-1}} \neq \mathcal{K}$, do the following:

- randomly pick an element k_i in the set of uncovered vertices $W_i = \mathcal{K} \setminus (K_{I_1} \cup K_{I_2} \cup ... \cup K_{I_{i-1}})$.
- randomly find a maximal set $K_{I_i} \in W_i$ containing k_i and such that $H(K_{I_i}) \leq \gamma_k$.
- add K_{I_i} to the partition.

Application:

Given an I/O string S and a consistent mode sequence $\sigma = \sigma_{m(1)}\sigma_{m(2)}...\sigma_{m(n)}$ where for $q = 1, ..., n$, $\sigma_{m(q)} = (k_{m(q)}, \xi_{m(q)})$ and $m(q) \in \{1, ...M(\sigma)\}$ ($M(\sigma)$ being the number of distinct modes in σ). We define two thresholds γ_ξ and γ_k and apply the Alphabet Reduction Algorithm to the input alphabet $\mathcal{E} = \{\xi_1, \xi_2, ..., \xi_{M(\sigma)}\}$ and output alphabet $\mathcal{K} = \{k_1, k_2, ..., k_{M(\sigma)}\}$.

This alphabet reduction algorithm serves many purposes. First, as mentioned earlier, this algorithm splits the motion alphabet (Σ) obtained from the earlier step into two alphabets (Ξ and $\mathcal{K}$) so that the recovered hybrid strings can be viewed as input/output strings of a hybrid automaton. Second, this process facilitates noise reduction, which occurs naturally when dealing with empirical data. Additionally, the reduction in the size of the alphabets makes the construction of automata more tractable. Finally, this process also leads to a

stochastic interpretation of the feedback and interrupt functions, which is sometimes more natural than a deterministic interpretation.

5.2 The Automata State Reduction Problem

After applying the alphabet reduction algorithm presented above, our recovered string is of the form $k_{i_1}\xi_{j_1}k_{i_2}\xi_{j_2}\cdots$. As mentioned in the introduction to this chapter, we can think of k_{i_q} as the "output" of a finite automaton in state s_q, and ξ_{j_q} as the corresponding "event" that triggers a transition from state s_q to s_{q+1}. The question then becomes, can we recover this underlying automaton? As we will see, we can always find at least one such automaton. However, this automaton is in general not unique. We will then focus our attention on trying to recover minimal automata, but first we need to establish some notation (found in [48]).

Definition 5.2.1 *An* output automaton *is a 6-tuple* $< S, \Sigma, Y, s_0, T, h >$*, where* S *is the finite set of states,* Σ *is the input alphabet,* Y *is the output alphabet,* $s_0 \in S$ *is the initial state,* $T \subseteq S \times \Sigma \times S$ *is the set of allowable transitions, and* $h : S \to Y$ *is the output function.*

(For our purposes, the input and output alphabets will be the finite set of interrupts (Ξ) and feedback laws ($\mathcal{K}$) respectively). We will write $s \xrightarrow{\sigma} s'$ as a shorthand for $(s, \sigma, s') \in T$.

Definition 5.2.2 *A* Deterministic Finite Automaton *(DFA) is an automaton where each state has a unique transition for a given symbol (if the transition is defined), i.e., if* $s \xrightarrow{\sigma} p$ *and* $s \xrightarrow{\sigma} q$ *, then* $p = q$*.*

A DFA is said to be *complete* if a transition is defined for every symbol in any given state (i.e. for any $s \in S$, $(s, \sigma, s') \in T$ for all $\sigma \in \Sigma$ and some $s' \in S$). Otherwise it is said to be *incomplete.*

Definition 5.2.3 *A* path π *as a finite alternating sequence,* $s_{i_1}\sigma_{j_1}s_{i_2}\sigma_{j_2}s_{i_2}\cdots\sigma_{j_{n-1}}s_{i_n}$*, of states and inputs, starting and ending with a state. We say that a path is* executable *on* A *if* $s_{i_1} = s_0$ *and each input transitions the state preceding it to the one following it, i.e.,* $(s_{i_q}, \sigma_{j_q}, s_{i_{q+1}}) \in T$ *for all* q*.*

Definition 5.2.4 *An* I/O path π_y *is an alternating sequence,* $y_{\ell_1}\sigma_{j_1}y_{\ell_2}\sigma_{j_2}\cdots\sigma_{j_{n-1}}y_{\ell_n}$*, of outputs and inputs, starting and ending with an output. We say that an I/O path is* executable *on A if there exists an executable path* $s_{i_1}\sigma_{j_1}s_{i_2}\sigma_{j_2}\cdots s_{j_n}$ *such that for all q,* $h(s_{i_q}) = y_{\ell_q}$.

With these definitions, the problem under consideration formulates as follows:

Problem 5.2.1 ***(Minimal Automaton Recovery)****: Given an I/O path* $\pi_y = y_{\ell_1}\sigma_{j_1}\cdots y_{\ell_n}$*, find the minimal deterministic output automaton* $A =< S, \Sigma, Y, s_0, T, h >$ *on which* π_y *is executable. Here, by "minimal," we understand the non-necessarily unique automaton with the minimum number of states.*

Note that for a given I/O path $\pi_y = y_{\ell_1}\sigma_{j_1}\cdots y_{\ell_n}$, there always exists at least one output automaton A on which π_y is executable. It is the automaton that, at each transition, jumps to a new state. This sequential output automaton has exactly n states. The set of automata that can execute π_y is thus non-empty, and there always exists a solution to the problem defined above.

In fact, this problem is related to the problem of producing minimal equivalent automata, since one could consider applying a state reduction algorithm to the sequential output-automaton derived above. Deterministic state reduction techniques with polynomial time exist in the case of completely specified DFA (see for example [14]). However, when the original DFA is not complete, which is our case with the sequential automaton, the problem is much harder, and proved to be NP-complete in [71]. Here, we show how our less general problem, the state reduction of a sequential incomplete DFA, is NP-complete.

A first task consists of finding when two states are mergeable, or here instead, when they are not mergeable. First, an obvious sufficient condition for non-mergeability of two states s_i and s_j is when their output values $h(s_i)$ and $h(s_j)$ are non-equal. A less trivial sufficient condition for the existence of non-mergeable states is the presence of what we call *inconsistent subpaths* in the I/O path.

Definition 5.2.5 *Given an I/O path, two subpaths* $y_{a_1}\sigma_{a_1}y_{a_2}\sigma_{a_2}...\sigma_{a_{n-1}}y_{a_n}$ *and*

$y_{b_1}\sigma_{b_1}y_{b_2}\sigma_{b_2}...\sigma_{b_{n-1}}y_{b_n}$ *are called* inconsistent *when* $y_{a_1} = y_{b_1}$, $\sigma_{a_1} = \sigma_{b_1}$, ..., $\sigma_{a_{n-1}} = \sigma_{b_{n-1}}$ *but* $y_{a_n} \neq y_{b_n}$.

We now state that when two subpaths are inconsistent, their two initial states with outputs y_{a_1} and y_{b_1} are non-mergeable. The reason is that, starting from these two states, the input string $\sigma_{a_1}\sigma_{a_2}...\sigma_{a_{n-1}}$ $(=\sigma_{b_1}\sigma_{b_2}...\sigma_{b_{n-1}})$ triggers two distinct output strings (which differ by the last output value only).

Assuming all non-mergeable states have been identified, a second task consists of finding how the states can be optimally merged into a minimum representation. Again, this problem can be nicely translated into a graph theory setting. We first build a graph where each state of the original sequential automaton is represented by a vertex. Then, edges are drawn between vertices whenever the corresponding states are non-mergeable. With this representation, the problem at hand is a graph coloring problem: "find a minimum color assignment such that every two adjacent (connected) vertices are painted with distinct colors." The idea is that vertices assigned a same color correspond to merged modes. By minimizing the number of colors used to fill the whole graph, we minimize the resulting number of distinct modes. Figure 15 shows how the same example as in Figure 14 translates into a graph coloring problem.

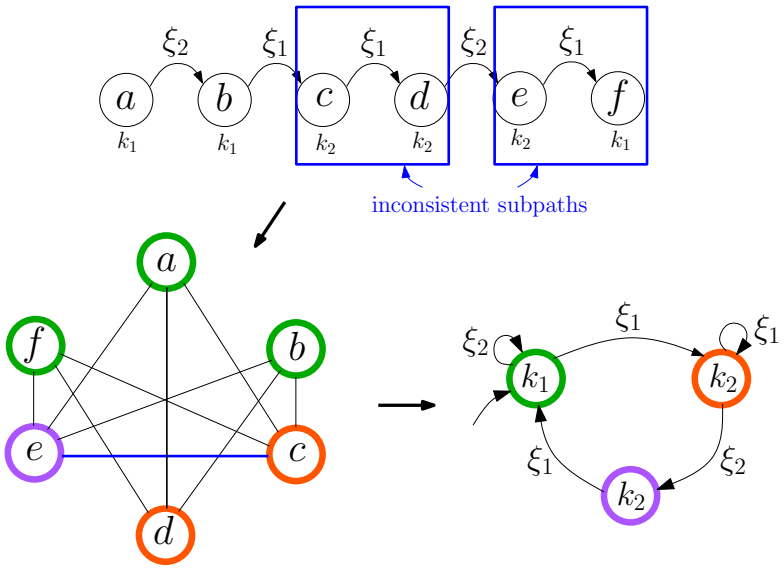


Figure 15: Minimal Input-Output Automaton Problem via Graph Coloring Problem.

In this figure, the original sequential automaton (top) is turned into a graph representation (bottom left) where the vertices correspond to the six states of the sequential automaton, and the edges represent non-mergeability between states. These are black when the states have distinct outputs (k_1 and k_2) or blue when the non-mergeability derives from the existence of inconsistent subpaths. An optimal coloring of the graph using three distinct colors is shown and the corresponding minimal automaton (bottom right) is depicted.

As seen, the state reduction of a sequential DFA can be split into two tasks: 1) the construction of a graph which requires the search for all non-mergeable states and 2) the search of an optimal coloring of the graph. While the first task can be accomplished in a number of operations that is a polynomial in the length of the original automaton, the second one is a known NP-complete problem, listed in [59]. For this reason, the whole problem is NP-complete and we cannot hope to solve it in polynomial time. There is, however, an abundance of literature pertaining to the reduction of incompletely specified automata. The various approaches for solving this problem can be categorized as either exact or heuristic based. As illustrated, the standard approach for this problem is based on an enumeration of the set of compatible states and the solution of a binate covering problem [69, 44]. A different approach for exact minimization not based on enumeration is presented in [70], while [73, 47] present heuristic based algorithms that significantly reduce run-time while obtaining correct results in most cases.

In the next section, we present different algorithms for the state reduction of an incompletely specified DFA, then compare their performance in terms of optimality and complexity (i.e. execution time).

5.3 Algorithms

5.3.1 Exhaustive Search Algorithm

In this section we describe an iterative algorithm that solves the problem of finding the smallest automaton consistent with a given I/O path $\pi_y = y_{\ell_1}\sigma_{j_1}\cdots\sigma_{j_{n-1}}y_{\ell_n}$. The set of all consistent automata is progressively constructed by reading the I/O path from the left to the right. At the end of this exhaustive search, the automaton with the fewest number of states

is the optimal solution. Each candidate automata is represented by $< S, \Sigma, Y, s_0, T, h >$ and the algorithm keeps track of this tuple as well as its current state s_{curr} at each step.

When the k^{th} iteration starts, the set of candidate automata $\mathcal{A}^{k-1}$ has been constructed from the reading of $y_{\ell_1}\sigma_{j_1}\cdots\sigma_{j_{k-1}}y_{\ell_k}$. (The algorithm is initialized at the 0th step with $\mathcal{A}^0 = \{A_0\}$, where A_0 is the single candidate automaton with $S = \{s_0\}$, $\Sigma = \varnothing$, $Y = \{y_{\ell_1}\}$, $s_0 = s_0$, $T = \varnothing$, h such that $h(s_0) = y_{\ell_1}$, and $s_{curr} = s_0$). The next input, σ_{j_k}, and output ,$y_{\ell_{k+1}}$, of the I/O path are read, and we do the following for each candidate automaton in $\mathcal{A}^{k-1}$:

Exhaustive Search Algorithm:

(a) if $\exists s'$ such that $(s_{curr}, \sigma_{j_k}, s') \in T$, then:

(a1) if $h(s') = y_{\ell_{k+1}}$, then the current state is set to s'.

(a2) if $h(s') \neq y_{\ell_{k+1}}$, then the candidate automaton is inconsistent and so, it is discarded.

(b) if $\nexists s'$ such that $(s_{curr}, \sigma_{j_k}, s') \in T$, we do two things:

(b1) we make the candidate automaton jump to a new state s_{new}: we add s_{new} to S, possibly add σ_{j_k} to Σ (if not already in that set), add $(s_{curr}, \sigma_{j_k}, s_{new})$ to T, set $h(s_{new}) = y_{\ell_{k+1}}$, and set the new current state to s_{new}.

(b2) we create a new candidate automaton for every pre-existing state s_{pre} such that $h(s_{pre}) = y_{\ell_{k+1}}$. For each of these new automata, we add $(s_{curr}, \sigma_{j_k}, s_{pre})$ to T, possibly add σ_{j_k} to Σ, and set the current state to s_{pre}.

All the modified or created candidate automata now constitute $\mathcal{A}^k$. After the last iteration, we are left with all possible consistent automata. The last task consists of choosing the one in $\mathcal{A}^n$ with lowest complexity, i.e., with the smallest number of states. Since, by construction, the algorithm produces every possible automaton that is consistent with the I/O path, it will return the optimal solution.

An example is shown in Figure 16, where all the automata consistent with the I/O path

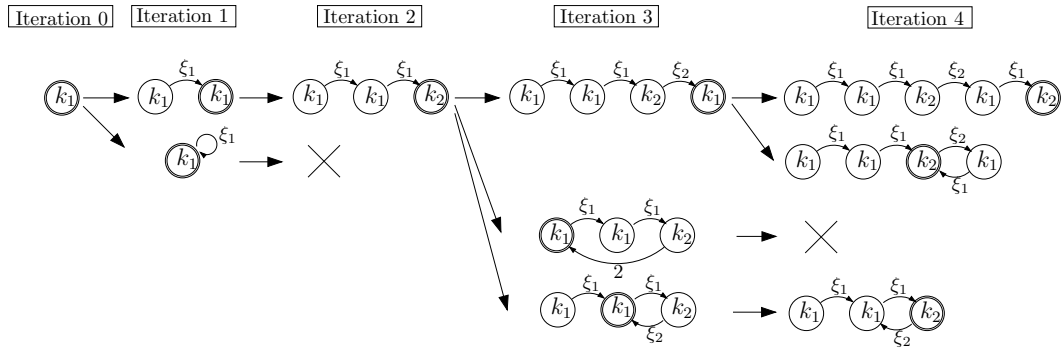


Figure 16: DFA State Reduction: Example of an Exhaustive Search (the "iteration 4" column shows all automata consistent with the I/O path $k_1\xi_1k_1\xi_1k_2\xi_2k_1\xi_1k_2$).

$k_1\xi_1k_1\xi_1k_2\xi_2k_1\xi_1k_2$ are constructed. In this diagram, a state is represented by a circle, with its output value inscribed. As the I/O path is read, the current state of each automaton is a bold circle. After the last iteration, we are left with three automata consistent with the I/O path. The optimal one is one with a minimum number of distinct states, i.e., the last one, with three states.

We now follow with a complexity analysis for this algorithm. Generally, as the length of the I/O path increases, the number of possible candidates quickly increases as well. Among all I/O paths of a given length, one of the form $y_1\sigma_1y_1\sigma_2y_1\sigma_3\cdots\sigma_{n-1}y_1$, where each input is a new input and all the outputs remain the same, constitutes a worst-case scenario, in the sense that it will generate a maximum number of candidate automata. Indeed, the use of a new input ensures that at every iteration of the algorithm, we are in case (b), the only one that creates new candidates. Also, the use of the same output maximizes the number of candidates created in (b2). Now, given such a path of length n, how many consistent candidates $z(n) = \text{card}(\mathcal{A}^n)$ are there? After iteration k, suppose that we have $z(k) = \sum_{i=1}^{k} z_i(k)$ candidates, where $z_i(k)$ is the number of candidates in $\mathcal{A}^k$ with exactly i states. Then, at $k+1$, we have $z_i(k+1) = z_{i-1}(k) + i \times z_i(k)$, where $z_{i-1}(k)$ accounts for the candidates in $\mathcal{A}^k$ with $i-1$ states incremented to i, via (b1), and $i \times z_i(k)$ accounts for the new candidates created from the candidates in $\mathcal{A}^k$ with i states, via (b2). This relation now allows us to calculate, by iteration, the total number $z(n)$ of consistent automata. This number is depicted in Figure 5.3.1, as a function of the length n of the worst-case I/O path.

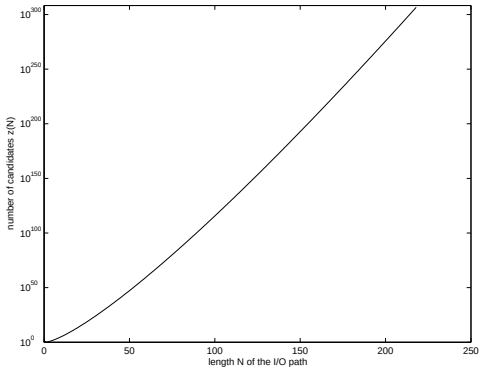


Figure 17: Exponential Complexity in the Worst-Case Scenario (semilog).

Note that on this semilog plot, the curve is superlinear, meaning that the number of automata created by the exhaustive search algorithm is a superexponential function of the length n of the I/O path. Because of this exploding complexity, the computation time of the proposed exhaustive search often becomes prohibitive. There should be no surprise that an algorithm that explicitly generates all possible candidate automata suffers from a hefty computational burden. As such, the next section presents a Pseudo-Exhaustive Search Algorithm that explicitly bounds this computational complexity.

5.3.2 Pseudo-Exhaustive Search Algorithm

In order to contain the complexity explosion of the previous algorithm, we apply two heuristic modifications to it:

- We limit the memory resources so that only a fixed number of automata M can be stored. When this number has been reached, new candidate automata created via (b2) are automatically discarded.

- We set a maximum complexity c_{max}. In (b1), if an automaton has more than c_{max} states, it is discarded.

With these modifications, the algorithm tries to answer the following question:

Q(c_{max},M): "*Given an upper bound M on the number of candidates we can store, can we find an automaton with at most c_{max} states, that is consistent with the given I/O path π_y?*".

Depending on how the algorithm terminates, the answer to Q(c_{max}, M) is:

- 'YES': when at the last iteration, we have at least one automaton.
- 'NO': when all automata are discarded because they had more than c_{max} states.
- 'I-can't-say': when all automata have been discarded, with some of them due to the fact that the maximum number of candidate automata M has been reached.

The results obtained when solving Q(c_{max}, M) are then used in a high-level iterative algorithm that increases the bounds until a solution is found. We call this algorithm the Pseudo-Exhaustive Search Algorithm:

Pseudo-Exhaustive Search Algorithm:

$c_{max} = 1$;
while Q(c_{max}, M) is not 'YES'
  if Q(c_{max}, M) = 'NO',
    increment c_{max};
  if Q(c_{max}, M) = 'I-can't-say',
    double M (memory size);
end of while
At this point, we have found an automata with optimal complexity c_{max}.

This algorithm performs well for automata with low complexity (typically ≤ 10) . However, as c_{max} increases, the amount of memory resources and the time to answer each Q(c_{max}, M) quickly become significantly large, often prohibitive. As a consequence, this algorithm is not best suited for cases with a high complexity optimal solution. For such cases, we could benefit from a time-limited version of the algorithm. The idea is to specify a time limit after which, if no solution has been found, the algorithm returns the actual value of c_{max}. This value is a lower bound on the complexity of the optimal solution and, as such, can be used to evaluate the quality of a suboptimal solution.

5.3.3 Suboptimal Algorithm

Since the exhaustive search solution to the problem can quickly become numerically intractable, we will present a suboptimal algorithm that produces a low-complexity output automaton A but not always a minimum complexity one. Similarly to the optimal algorithm, the suboptimal algorithm constructs a consistent automaton sequentially by progressively reading a given I/O path $\pi_y = y_{\ell_1}\sigma_{j_1}\cdots\sigma_{j_{n-1}}y_{\ell_n}$. However, instead of keeping up with all possible consistent automata at each step, this algorithm greedily selects one of them randomly. So at each step, we do the following:

(a) If there exists a s' such that $(s_{curr}, \sigma_{j_q}, s') \in T$ with $h(s') = y_{\ell_{q+1}}$, then update the current state as $s_{curr} = s'$.

(b) If no s' exists such that $(s_{curr}, \sigma_{j_q}, s') \in T$, then let $S_{possible} := \{s \in S \quad | \quad h(s) = y_{\ell_{q+1}}\}$. Now for each $s \in S_{possible}$, suppose the transition $(s_{curr}, \sigma_{j_q}, s)$ is added to T and check whether this causes the automation A to be inconsistent with the given I/O path π_y. To do this, we go through the I/O path and check if $(s_{i_k}, \sigma_{j_k}, s_{i_{k+1}}) \in T$ then $h(s_{i_{k+1}}) = y_{\ell_{k+1}}$ for $k = q, \ldots, n-1$. If we find that $h(s_{i_{k+1}}) \neq y_{\ell_{k+1}}$ for any k then adding the transition causes A to be inconsistent, and thus this transition should not be added. Of course if we find that $(s_{i_k}, \sigma_{j_k}, s_{i_{k+1}}) \notin T$ for some k, then A is consistent and we do not need to go through the remaining I/O path. If no inconsistency is found, then add s to the set of candidate states $S_{candidate}$.

   (b1) If the set $S_{candidate}$ is empty, then add a new state s_{new} to automaton A, with $h(s_{new}) = y_{\ell_{q+1}}$. We also add the transition $(s_{curr}, \sigma_{j_q}, s_{new})$ to the set of allowable transitions T and let $s_{curr} = s_{new}$.

   (b2) If $S_{candidate}$ is not empty, then let s be a random state selected from $S_{candidate}$ and add the transition $(s_{curr}, \sigma_{j_q}, s)$ to T. Set $s_{curr} = s$.

In step (b), we ensure that a transition to an existing state is added only if this does not cause an inconsistency with π_y, thus guaranteeing that the algorithm terminates with

an automaton A consistent with π_y at the end of iteration $q = n$. Since we only add a new state when absolutely necessary, i.e., when we have no suitable candidates, we do in fact reduce the number of states. Note that in step $(b2)$, we select the next state randomly from the set of all possible candidates. This random selection is where a possible suboptimal candidate may be chosen. We can only make the optimal choice by generating a different possible automaton for each possible candidate and continuing in this manner. However, as we saw in previous section this could quickly become numerically intractable as the number of possible automatons can explode, thus we incorporate this greedy approach to quickly find a suboptimal solution. In contrast to the high complexity of the previous algorithm, we prove that this algorithm has cubic complexity. To see this, note that at each iteration q of the algorithm, we can possibly connect to all of the previous q states and have to go through $n - q$ steps of the output path π_y to check for consistency. Hence at each iteration q, we have to perform $q(n - q)$ possible operations. Now let $f(n)$ denote the maximum number of operations necessary for this algorithm to finish given an I/O path of length n, then

$$\begin{aligned} f(n) &= \sum_{q=1}^{n} q(n-q), \text{ while} \\ f(n+1) &= \sum_{q=1}^{n+1} q(n+1-q) \\ &= \sum_{q=1}^{n+1} q(n-q) + \sum_{q=1}^{n+1} q \\ &= \sum_{q=1}^{n} q(n-q) + \sum_{q=1}^{n} q \\ &= f(n) + \frac{n(n+1)}{2}. \end{aligned}$$

Thus we conclude that this algorithm has cubic complexity, i.e. $O(n^3)$. Of course, this reduction in complexity comes at the expense of optimality.

We illustrate a situation where the algorithm may produce a suboptimal solution with a specific example. Suppose we are given $\pi_y = y_1\sigma_1 y_1\sigma_2 y_2\sigma_1 y_1\sigma_2 y_1\sigma_3 y_2\sigma_1 y_1\sigma_2 y_1\sigma_1 y_2$, then a possible automaton A at iteration i is shown in Figure 18 (a). Note here at iteration $i = 4$, we could have added transition (s_3, σ_2, s_1) instead of the transition (s_3, σ_2, s_3) since both

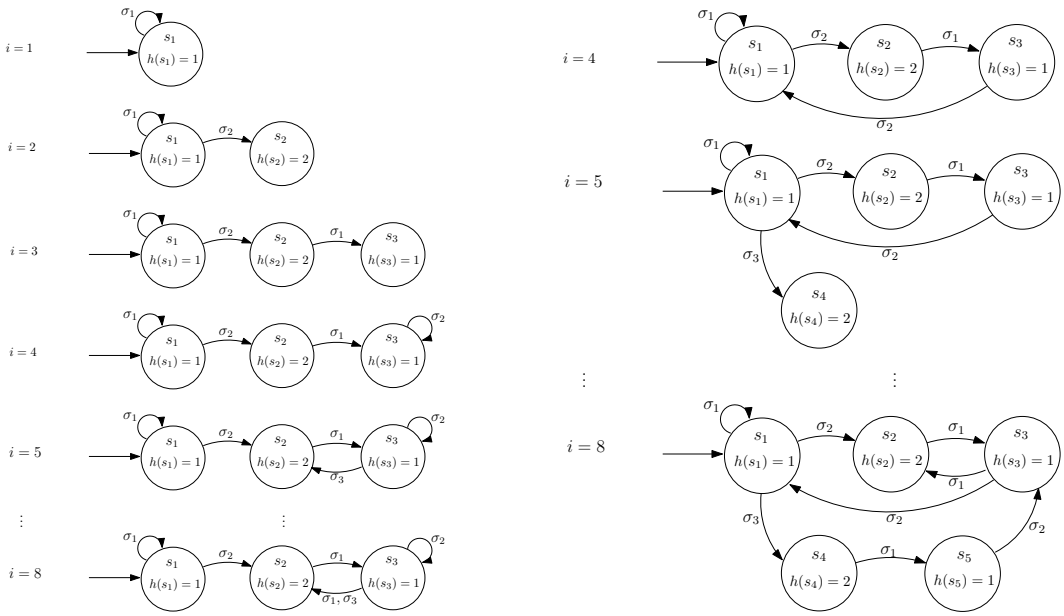


Figure 18: Different Outcomes of the Suboptimal DFA State Reduction Algorithm.

states s_1 and s_3 are possible candidates. The resulting automaton with s_1 as the next state is shown in Figure 18 (b), while Figure 18 (a) shows the optimal automaton with $|S_A| = 3$. This example illustrates how this random choice of next state from all possible candidates can possibly lead to a suboptimal solution.

5.3.4 Performance Comparison

In order to compare the performance of the three different algorithms, the following table gathers some experimental results that allow this comparison. The setup when creating this table is as follows: for each example, a random underlying automaton was created, over which a random walk was executed in order to produce an I/O path. The manner in which the entries in the table should be read is as follows:

- The "Experimental setting" reads $(\alpha, \beta, \gamma, \delta)$ where α is the number of states in the underlying automaton used for generating the string, β is the size of its input alphabet, γ the size of its output alphabet, and δ the number of jumps in the random walk (i.e the length n of the I/O path)

- The results show the computation time on a standard desktop computer (Pentium 4, CPU 2.4Ghz, 1.25GB of RAM) together with the complexity (the size of the automaton that was recovered) of the solution in parenthesis.

When no entry is given, the computation could not be performed in a reasonable amount of time. In this particular case, we terminated the algorithm after 10 minutes had elapsed.

Table 3: Performance Comparison of Three DFA State Reduction Algorithms.

Experimental setting	Algorithm 1 (Exhaustive)	Algorithm 2 (Pseudo-Exhaustive)	Algorithm 3 (Greedy)
(4,2,2,15)	0.031s (4)	0.032s (4)	0.015s (5)
(5,3,3,15)	128.4s (5)	0.032s (5)	0.015s (5)
(6,3,3,100)		5.391s (6)	0.016s (12)
(6,3,3,500)		5.769s (6)	0.078s (61)
(6,3,3,2000)		66.95s (6)	2.360s (170)
(8,3,3,2000)			4.672s (201)
(100,20,20,5000)			13.65s (299)

As can be seen from the table, Algorithms 1 and 2 give the same (optimal) complexity, which is to be expected. Moreover, Algorithm 2 outperforms Algorithm 1 in terms of computational efficiency. The third algorithm, does not return an optimal solution (expect for the second example) but, because of its cubic complexity, always terminates in predictable (polynomial) time.

CHAPTER VI

A UNIFIED SOFTWARE PACKAGE FOR MODELING AND SIMULATION

In this chapter, we present a software automated tool for extracting and simulating high-level (multi-modal) control programs from observed empirical data. The tool is MODEbox (Mode Optimization and Data Extraction = MODE), available as a MATLAB toolbox at http://gritslab.ece.gatech.edu/MODEbox.html. It was created to address the challenging problem of combining all the methods developed in the previous chapters into a single automated process that can be used for numerous applications.

In the next sections, we describe the modules of MODEbox, we then show how a very intuitive graphical user interface guides the user through every step of the process, and we conclude by running the program on an example.

6.1 Modules

Figure 19 shows how the global task of MODEbox, extracting and simulating hybrid system models from empirical data, is divided into four subtasks corresponding to the four major modules "Preprocessing", "MDL-capturing", "Automaton", "Simulation".

Overall, MODEbox takes in a string of input-output measurements, turns it into a string of symbolic input-output pairs, and produces consistent MDL strings. MODEbox then constructs a finite automaton capable of producing these MDL strings as a sample path, which can then be used as a control law to simulate similar trajectories or to control real systems. We now give a more detailed description of each of the four basic modules.

Module 1: Preprocessing

In the first block in Figure 19, a data string consisting of input-output pairs is being read by MODEbox. The assumption is that the data is generated by a dynamical system

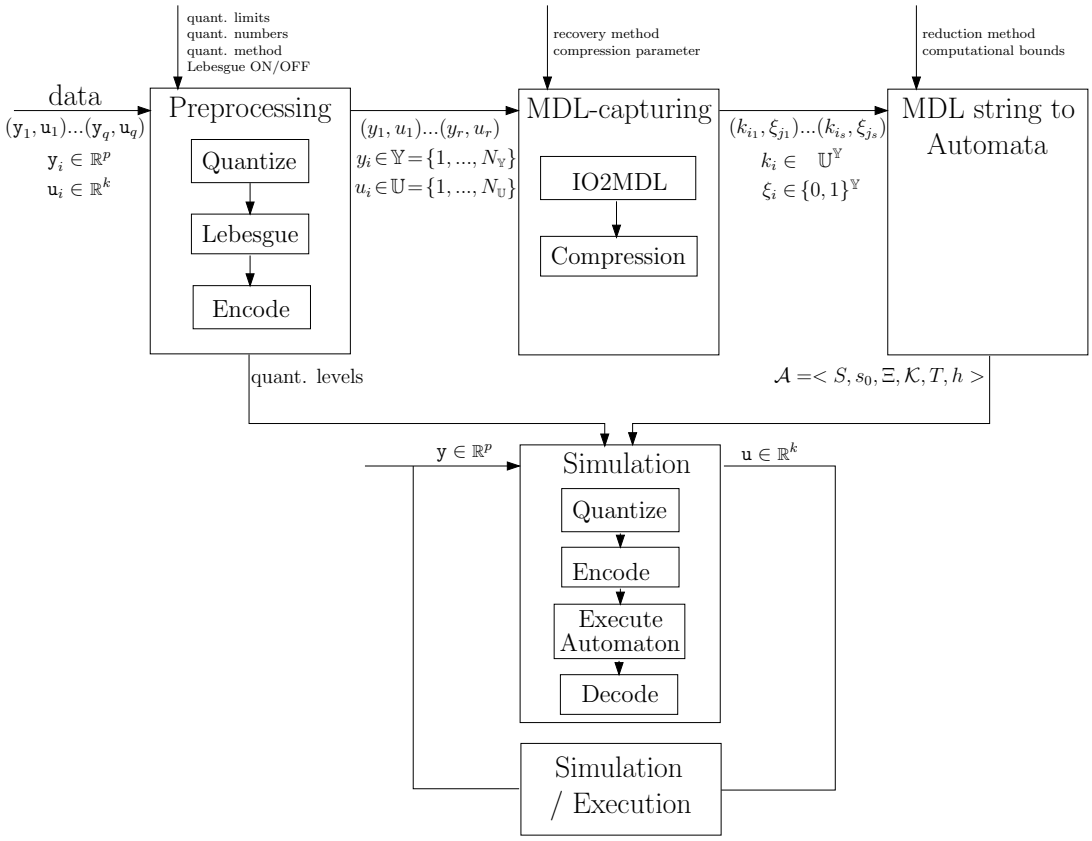


Figure 19: Overview of the MODEbox Operational Units.

$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$, $\mathbf{y}_k = h(\mathbf{x}_k)$, and the data string is given by $(\mathbf{y}_1, \mathbf{u}_1), \ldots, (\mathbf{y}_q, \mathbf{u}_q)$, where the outputs $\mathbf{y}_i \in \mathbb{R}^p$ and the inputs $\mathbf{u}_i \in \mathbb{R}^k$. Here, we use boldface to denote variables before they have been operated on by the **Preprocessing** module. In fact, this string is operated on by the **Preprocessing** module using three different, sequential components, namely `Quantize`, `Encode` and `Lebesque`. `Quantize` produces a finite precision representation of the data string, `Encode` maps the quantized data strings to symbols, and `Lebesgue` reduces the length of the data string by making sure that no consecutive, symbolic input-output pairs are the same [6]. As a result, the output of this block is a new string $(y_1, u_1), \ldots, (y_N, u_N)$, where $N \leq q$ and $y_i \in \mathbb{Y}$, $u_i \in \mathbb{U}$. The user can specify how many regions (`quant. numbers`) the quantization should produce and what quantization method to use (`quant. method`). The user can select between four quantization methods: uniform, equi-distributed, optimal PCM, and optimal DPCM. The choice of a quantization method and a number of quantization levels is motivated by whether the user is more interested in the final model's

complexity or the model's performance at approximating the original system. The user can also choose whether or not Lebesgue sampling should be employed (`Lebesgue ON/OFF`).

Module 2: MDL Capturing

The output from the **Preprocessing** module is now fed into the **MDL Capturing** module. The problem of recovering MDL strings from input-output data was introduced in chapter 2, and different strategies for minimizing certain objectives were developed in chapter 3. Although minimizing the so-called empirical specification complexity is the preferred objective, this is not easily achievable. Instead, in the `IO2MDL` block, the user can choose between one of four methods that manage this complexity: "MinL-MaxM", "MaxL-MinM", "MinL-LowM", or "LowL-MinM". Once a MDL string is produced, the result is fed to `Compression`, where "similar" feedback laws and interrupt functions are identified and combined, based on a user-specified `compression parameter` that sets the threshold (between 0 and 1) for how similar they need to be in order to be considered the same. The similarity measure is the normalized average entropy introduced in 5.1, quantifying the uncertainty in the random variable $k_i(y)$ (and respectively $\xi_i(y)$), where i can be any of the modes that under consideration. The resulting output from this module is a string $(k_{i_1}, \xi_{j_1}), \ldots, (k_{i_s}, \xi_{j_s})$, where $s \leq N$.

Module 3: MDL to Automata

The resulting MDL string can be thought of as a sample path generated on a finite automaton, where the output function $h(s) = k$ returns the feedback law the system should use in state s. Transitions in the automaton are triggered by the corresponding interrupts. If we let $\mathcal{K}$ and Ξ denote the set of feedback laws and interrupt functions respectively, this module produces a finite automaton $\mathcal{A} = \langle S, s_0, \Xi, \mathcal{K}, T, h \rangle$, where S is the state space, s_0 the initial state, $T : S \times \Xi \to S$ is the transition relation, and $h, \mathcal{K}$, and Ξ are as previously defined. Moreover, $\mathcal{A}$ should not only be such that the MDL string is a sample path of $\mathcal{A}$, but $\mathcal{A}$ should also be of low complexity, in the sense of state-space cardinality. There is an abundance of literature pertaining to this topic in terms of reducing incompletely specified finite state machines. This subject was considered in detail in Sections 5.2 and 5.3, where

algorithms for finding such automata were presented. In fact, methods for obtaining optimal automata are available, but due to the computational complexity associated with this problem (which is known to be NP-complete), the user can choose not to insist on optimality through the `reduction method` input. The two methods implemented in MODEbox include an exhaustive search algorithm that creates all automata consistent with the MDL string and a suboptimal algorithm that returns a low complexity solution. Here, the price for optimality is to be paid in terms of computation time.

Module 4: Simulation/Execution

Once an automaton $\mathcal{A}$ has been produced, the obtained hybrid control law can be simulated to mimic observed behavior and/or used for controlling real systems in the **Simulation** module. This last step in the MODEbox flow diagram represents this situation, where externally obtained measurements $y \in \mathbb{R}^p$ (either through simulation or from a real experiment) are quantized and encoded (with the same quantization levels (`quant. levels`) used in the **Preprocessing** module) to produce symbolic measurements $y \in \mathbb{Y}$. These measurements are then used for driving the finite automaton through `ExecuteAutomata`, and the corresponding control symbols are computed and decoded to produce executable control signals $u \in \mathbb{R}^k$. This is the only block in the toolbox that requires any significant user input since each simulation is application specific.

6.2 User Interface

The use of MODEbox is made easy by a very intuitive graphical user interface (GUI). This GUI, accessed by typing `MODEboxgui` in the MATLAB command window, is represented in Figure 20. As one can see, the organization of the interface explicitly uses the same modules described in the previous section. For this reason, a proper use of the GUI should not be difficult, once one understands how each module works. The two windows at the bottom display helpful information to guide the user through the different steps. While the right window plays the role of a workspace window, i.e., displays all the created variables and their format, the left window prints out report information at each module completion.

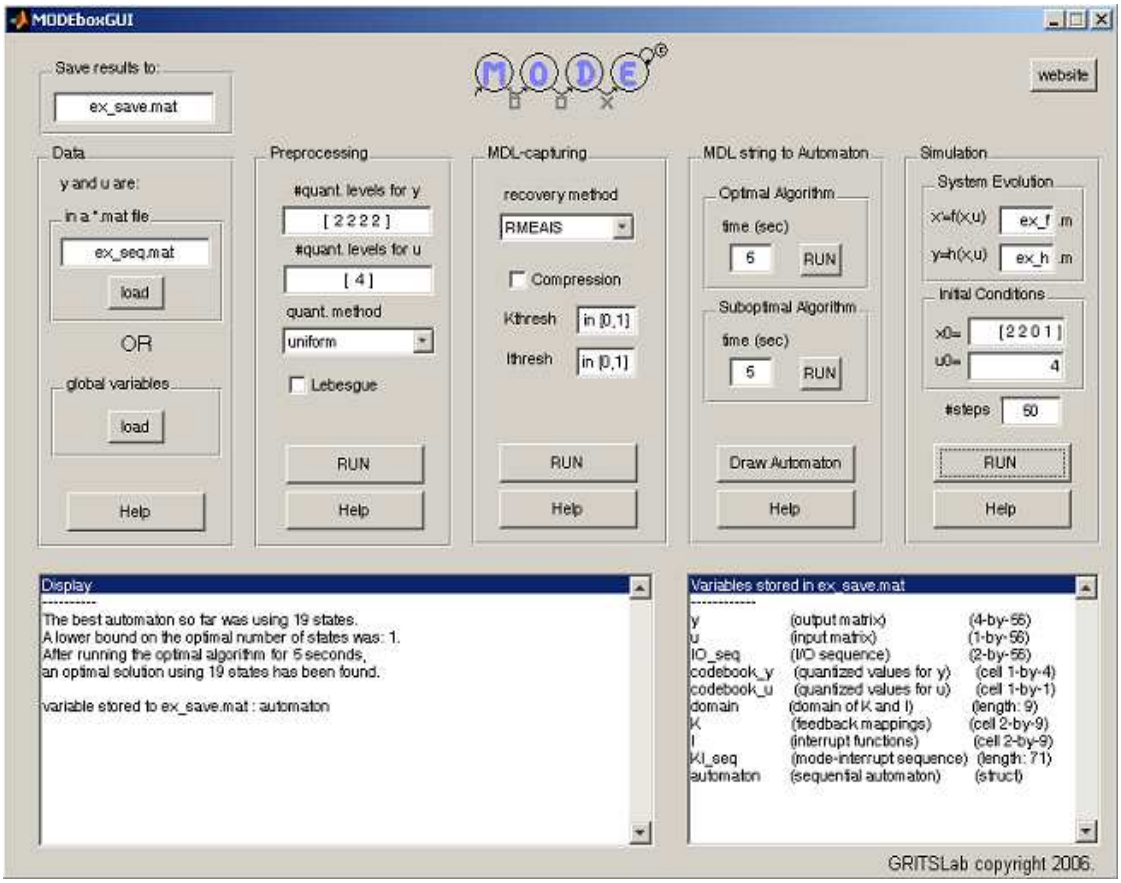


Figure 20: MODEbox Graphical User Interface.

We now describe a succession of operations for using the MODEbox GUI.

0. Save results box:

First, the user enters the path of a *.mat file where results will be saved. At any time, the "Stored variables" window at the bottom right shows which variables have been saved in this *.mat file.

1. Data box:

Next, the user loads the input-output data: u and y should be two matrices where the number of rows corresponds to the dimensions of u and y and the number of columns corresponds to the number of samples. The user can load u and y by doing one of the following:

- specifying the path of a *.mat file where the input (u) and output (y) matrices are stored.

- loading global variables u and y form the current Matlab workspace in the main window.

2. Preprocessing box:

The data is then turned into a string of symbolic input-output pairs. The user must specify:

1) the number of quantization levels used for each dimension of y and u.

2) a quantization method from the pop-up menu.

3) whether or not Lebesgue sampling should be used (check box).

3. MDL-capturing box:

Then the I/O string is turned into an alternating sequence of feedback mappings (K) and interrupt functions (I). The user must choose a recovery method from the pop-up menu. The user must choose whether compression should be done (check button) and if so, must set thresholds `Kthresh` and `Ithresh` between 0 and 1, 0 meaning that feedback mappings (respectively interrupt functions) will be merged only if they are absolutely identical, and 1 meaning that all feedback mappings (respectively interrupt functions) will be merged to the same one.

4. MDL string to Automaton box:

When the previous part is run, the alternating sequence of Ks and Is is automatically turned into a sequential output automaton. To improve the complexity (number of states) of this automaton, the user can choose to run an optimal algorithm or a suboptimal algorithm for a specific time. The algorithms can be run several times and in any order. The program always keeps the simplest automaton encountered. This automaton can be plotted by clicking the Draw Automaton button. This option requires that Simulink is installed.

5. Simulation box:

Finally, a simulation can be run by executing the recovered automaton. Here, the user needs to specify the names of two implemented functions that govern the system evolution:

(1) $x_{k+1} = f(x_k, u_k)$ and (2) $y_k = h(x_k, u_k)$. For example, in the case of the ants, these two functions could be `funicycle` and `hunicycle`. The user would then have to write two files `funicycle.m` and `hunicycle.m` for the implementation of these two functions. Finally, the simulation requires initial state and control conditions, and a number of iterations before termination.

Once the simulation is run, the input, output, and state trajectories are stored under the variable names UU, YY, and XX.

Since the MODEbox is designed to be a general tool for use with a variety of different applications, it is the responsibility of the user to implement the state evolution and output functions. For the same reason, MODEbox cannot provide a general tool for representing the simulated trajectories. For example, in the case of the ants, we wrote a function generating a movie (see Figure 7) from the state trajectories $XX = [x; y; \theta]$.

6.3 Example

To better illustrate the MODEbox operation, we consider a simple maze example. Note that this example is overly simplistic but it is merely to be thought of as a vehicle for making certain operational aspects explicit. In fact, we will carefully describe how each operational unit works concretely on the example. The files and instructions for running this example are available on the MODEbox webpage (http://gritslab.ece.gatech.edu/MODEbox.html).

Suppose data is collected from the observations of a robot going through a maze as depicted in Figure 21. This data will be given by an input/output string, where the inputs are variables relevant to the system's control decisions, and the outputs are signals possibly used to control the observed system. For this particular maze example, we choose the outputs to be the color (i.e. 0=black, 1=white) of the cell in front of the robot (y_1), on its left (y_2), behind it (y_3), and on its right (y_4). The input is the corresponding action (1-"go straight", 2-"turn left", 3-"U-turn", 4-"turn right") taken by the robot in response to the outputs.

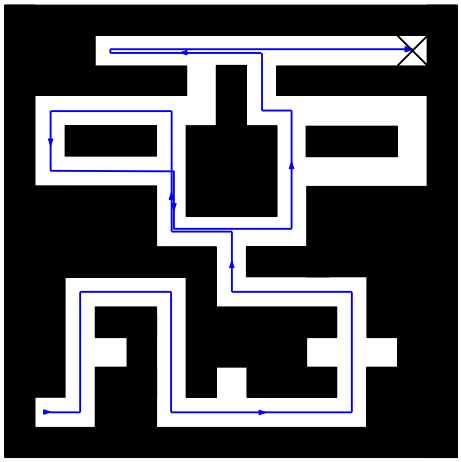


Figure 21: Observed Trajectory of a Robot Going Through a Maze (used as input data for MODEbox).

Preprocessing module:

Each data point comes from the set $Y \times U$, where $Y = \{0,1\}^4$ and $U = \{1,2,3,4\}$. Since this is already a small discrete set, we do not need to quantize the data any further. The recommended choice for the quantization levels are thus $[2,2,2,2]$ for the output and $[4]$ for the input. If so, the data is encoded into a discrete set of symbols $\mathbb{Y} \times \mathbb{U}$, where $\mathbb{Y} = \{1,2,\ldots,16\}$ and $\mathbb{U} = \{1,2,3,4\}$, resulting in an I/O string.

MDL Capturing module:

A close look at the trajectory in Figure 21 shows that the robot's behavior is almost always predictable. Indeed, the robot goes straight whenever possible, and turns left or right when it is the only possible choice. The only ambiguous situation is when the robot encounters a wall, with two openings on the left and right (situation that we will note $y_\top = (0,1,1,1)'$). In this case, we see that the robot sometimes chooses to turn left ($u = 2$) and sometimes right ($u = 4$). The above remarks help understand the outcome of each mode recovery methods, represented in Figure 22. We now analyze these results:

a) The MinL method returns a mode sequence with a minimum length of 4 modes. Were it not for the existence of distinct inputs when $y_\top$ is encountered, the system's trajectory could be represented with one single mode. Here, the first three modes interrupt when $y_\top$ is encountered, and their respective actions at this particular situation are

"turn left", "turn left", and "turn right". The interpretation of the last mode's interrupt is more subtle. In fact, this last mode interrupts when $y = (1, 0, 0, 0)'$, i.e., a situation similar to the starting position, which is not encountered on the trajectory of the system in this last mode. This means that mode 4 does not interrupt until the end of the I/O string.

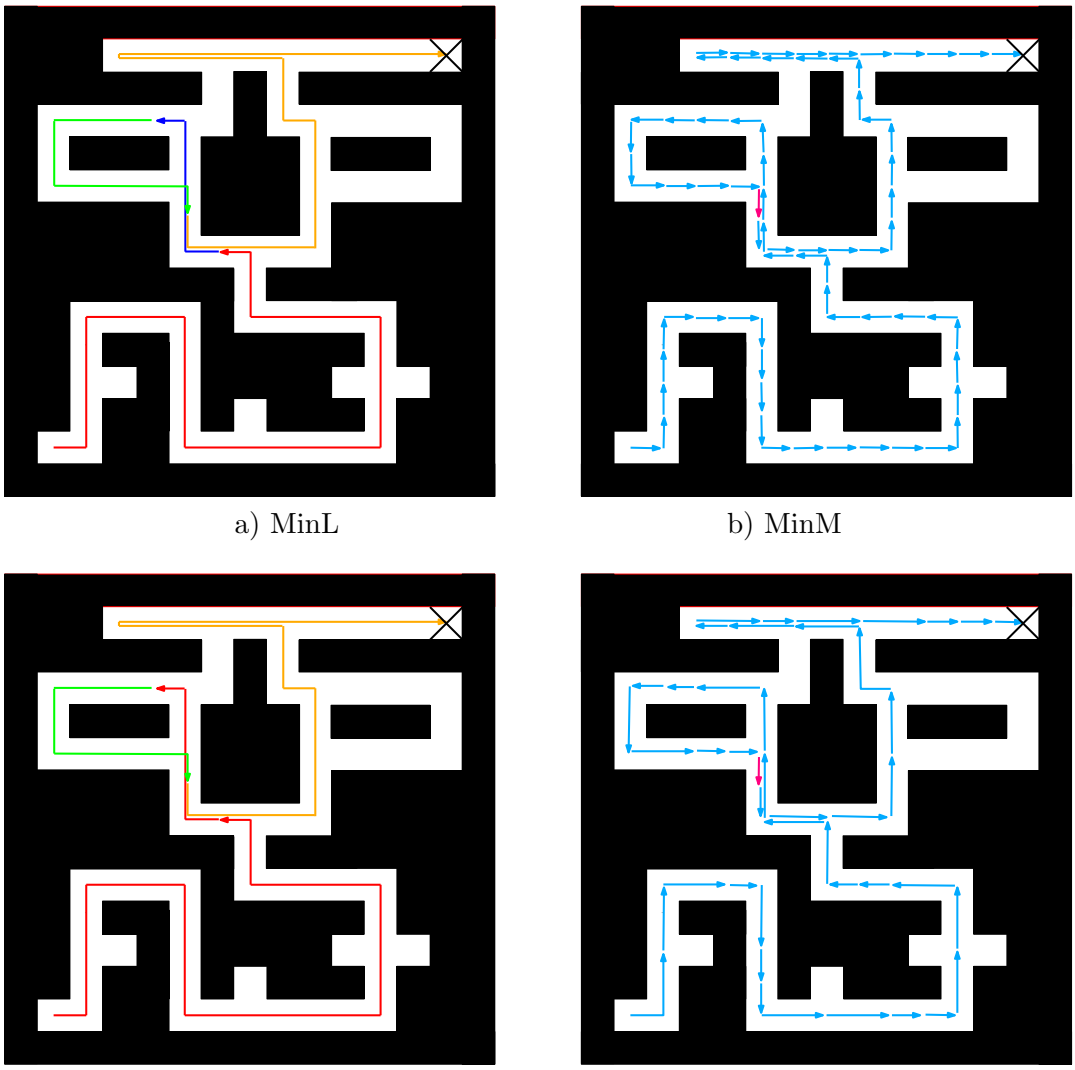
b) The MinM method returns an Always Interrupt Sequence using a minimum number of 2 distinct modes. The first one turns left when $y_\top$ is encountered, while the second one turns right. We also see that this second mode is used only once in the whole mode sequence. This results from the minimization of the entropy over the set of consistent AIS.

c) The MinL-LowM method returns a slightly better mode sequence than the MinL method, after noticing that the second and third mode are identical.

d) The MinM-LowL method returns an improved version of the mode sequence recovered with the MinM method. The interrupt of the first mode is modified so that now, it only triggers when $y = [1, 0, 1, 0]'$ (i.e. when the robot can go straight or back only). This interrupt, which corresponds to the interrupt triggered between mode 1 and mode 2 in b), is the only one that needs to be preserved for mode 1.

At this point, we assume that the user selected the mode sequence recovered from the MinL-LowM recovery method, which is the one with lowest specification complexity. In this case, the mode sequence is of the form $k_1\xi_1k_1\xi_1k_2\xi_2k_3$. Now, if the "compression" box is checked, the alphabet reduction algorithms will simplify this sequence to $k_{13}\xi_{12}k_{13}\xi_{12}k_2\xi_{12}k_{13}$, after noticing that the feedback mappings k_1 and k_3 are identical, as well as the interrupt functions ξ_1 and ξ_2. Moreover, the feedback mappings k_{13} and k_2 are very similar, in the sense that they only differ by the action taken when $y_\top$ is encountered (namely turn left or right). Consequently, if a big enough similarity threshold γ_k is selected, the two modes will be considered similar enough to be merged, resulting in the mode sequence $k_{123}\xi_{12}k_{123}\xi_{12}k_{123}\xi_{12}k_{123}$. Note that the mode k_{123} is a random process: now, when

$y_\top$ is encountered, the probability of turning right is $\frac{1}{4}$ and the probability of turning left is $\frac{3}{4}$.



a) MinL b) MinM

c) MinL-LowM d) MinM-LowL

Figure 22: Maze Example: Outcome for Each Mode Recovery Method.

MDL to Automata module:

Next, the mode sequence is turned into a minimal automaton. Figure 23 shows the optimal automaton, depending on whether the mode sequence $k_{13}\xi_{12}k_{13}\xi_{12}k_2\xi_{12}k_{13}$ (case a)) or $k_{123}\xi_{12}k_{123}\xi_{12}k_{123}\xi_{12}k_{123}$ (case b)) was recovered.

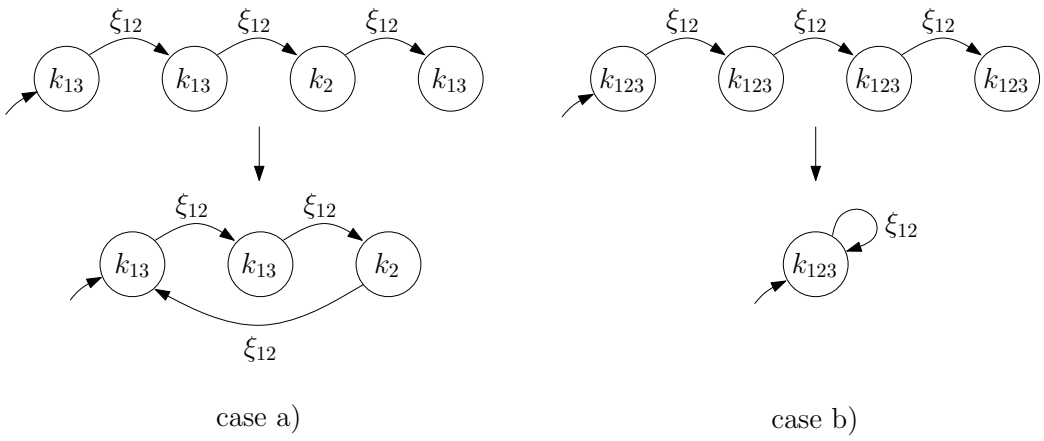


Figure 23: Maze Example: From Sequential Automata to Minimal Automata.

Simulation module:

Finally, the recovered control procedures are now applied to a robot in order to navigate through a new maze. To this end, we define two functions $f(x, u)$ and $h(x)$ reflecting the state evolution and observation of the robot to be controlled. At each time increment, the control $u_q = k_{m_q}(y_q)$ is applied, where k_{m_q} is the feedback mapping of the active mode, and y_q is the quantized version of the output $h(x_q)$. The state then evolves according to $x_{q+1} = f(x_q, u_q)$. Figure 24 shows two trajectories corresponding to the simulation of the two automata represented in Figure 23. Note how the deterministic automaton gets caught inside a loop, while the stochastic one is eventually able to find its way out.

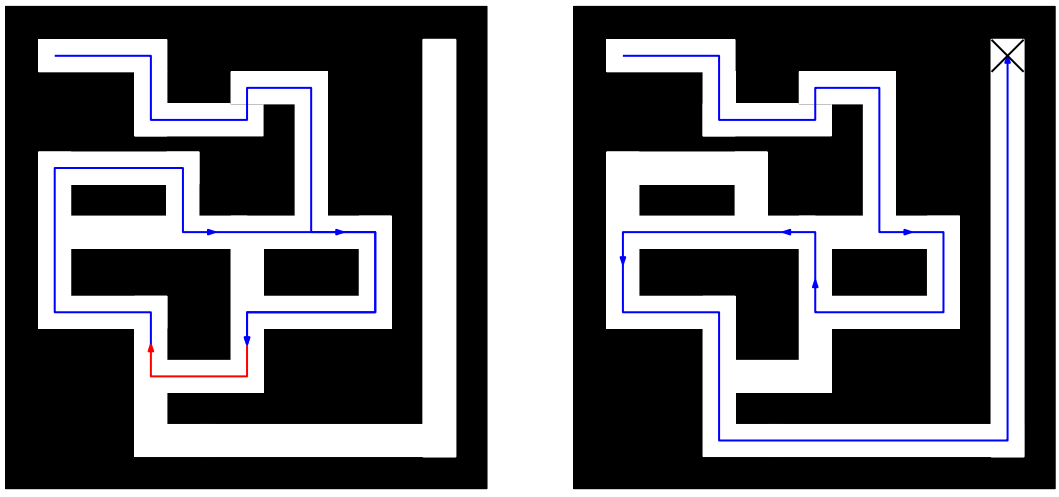


Figure 24: Maze Example: Two Simulated Trajectories.

PART II

Optimal Multi-Modal Control

The main objective of this second part is to develop rational methods for controlling complex systems by switching between different modes of operation, resulting in hybrid, multi-modal control strategies. This work ties the design of feedback controllers (Part I) with a supervisory control law that dictates the transitions between the system's operational modes. The problem is cast in an optimal control setting: given a sequence of behaviors (or modes of operation), we consider the problem of finding a switching policy that minimizes a cost functional of the system's trajectory.

This approach constitutes a compromise between two distinct approaches in robot navigation: the deliberative approach and the reactive approach [3]. As in the deliberative approach, the trajectory of the system is carefully planned out in advance, in order to minimize a performance index. As in the reactive approach, the system's motion results from a sequence of autonomous behaviors. However, our approach is neither purely deliberative nor purely reactive for two reasons: 1) the offline trajectory computation requires that all changes in the environment in which the system evolves, if any, are known in advance. This contrasts with the reactive approach which deals with robust systems in unknown and changing environment 2) except for the switching policy, the system under consideration is completely autonomous. This contrasts with the deliberative approach where systems, mostly completely controllable, are controlled using an open loop. Our goal is to apply the analytic tools of one approach (deliberative) to the control of the switched autonomous systems of the other approach (reactive). However, we must keep in mind that such methods can only be efficient in a structured environment.

The cadre of this second part also differs from the previous one in the sense that the hybrid systems that we consider are now time-triggered. In the MDL framework, these correspond to devices controlled by mode sequences $(k_1, T_1)(k_2, T_2)...(k_N, T_N)$ instead of $(k_1, \xi_1)(k_2, \xi_2)...(k_N, \xi_N)$. Here, T_1, T_2, ..., T_N are ordered elements in $\mathbb{R}$ corresponding to the switching times from one mode to the next one. The problem consists then in choosing such moments (sometimes along with other switching variables) in order to minimize a cost functional of the system's state trajectory.

The organization of this second part is as follows. Chapter 7 shows on a simple example

how the multi-modal control problem can be cast and solved in an optimal control setting. In particular, necessary conditions for optimality are derived using calculus of variation and a numerical algorithm for the design of an optimal switching policy is presented. In Chapter 8, the methods are progressively modified to comply with more complex systems and performance indices. These modifications allow us to consider a variety of applications in robot navigation, curve fitting, data compression, control of epidemics, etc [83, 84, 82, 26, 27, 29]. Finally, this part ends with the discussion of possible extensions to this work in Chapter 9.

CHAPTER VII

OPTIMAL CONTROL OF SWITCHED AUTONOMOUS SYSTEMS

This chapter serves as an introduction to the methodology used for solving a class of optimal control of switched autonomous problems. The methods are shown on a simple system (found in [38]) and more complex systems will be considered in the next chapter. The approach consists of deriving expressions for the partial derivatives of a cost function using calculus of variations. These expressions are then used with an iterative gradient descent numerical algorithm to find a set of control variables locally minimizing the cost function. The effectiveness of the approach is shown through a simple simulation example where a supervisory controller switches its attention between two subsystems.

7.1 Problem Formulation

The state evolution $x(t) \in \mathbb{R}^n$ of a switched autonomous system is given by

$$\begin{cases} x(T_0) = x_0, \\ \dot{x}(t) = f_i(x(t)), \quad t \in (T_{i-1}, T_i), \quad i = 1, ..., N, \end{cases} \tag{12}$$

where $\{f_i\}_{i=1}^N$ is a given finite sequence of continuously differentiable functions from $\mathbb{R}^n$ to $\mathbb{R}^n$. The initial state x_0 and initial and final times T_0 and T_N are also given. The control variables are the instants $T_1, ...T_{N-1}$ when the system switches from one mode to another. These switching times are to be chosen in order to minimize a given cost function

$$J = \int_{T_0}^{T_N} L(x(t))dt, \tag{13}$$

where L is a continuously differentiable function from $\mathbb{R}^n$ to $\mathbb{R}$.

Example:

A good illustration is in multi-process control, where a collection of p unstable dynamical systems are to be controlled. At every instant, the controller can only control a subset

$m < p$ of dynamical systems. To avoid optimal solutions exhibiting a Zeno phenomenon, the controller can only switch attention a finite (given) number of times. Figure 25 shows an example of such a process, where two unstable scalar systems $x_1(t)$ and $x_2(t)$ are to be controlled. We assume that the controller can only control one of the two systems at a time, so it switches attention from one to another. Using the notation $x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$, the global dynamics are:

$$\begin{aligned}
\dot{x}(t) &= \begin{pmatrix} -1 & 0 \\ 1 & 2 \end{pmatrix} x(t) = A_1 x(t), \qquad && t \in [0, T_1) \\
\dot{x}(t) &= \begin{pmatrix} 1 & 1 \\ 0 & -2 \end{pmatrix} x(t) = A_2 x(t), \qquad && t \in [T_1, T_2) \\
\dot{x}(t) &= A_1 x(t), && t \in [T_2, T_3) \\
\dot{x}(t) &= A_2 x(t), && t \in [T_3, 1],
\end{aligned}$$

with initial conditions $x(0) = [1, 0]'$. Figure 25 shows the state evolution when the switching times (the control variables) are $T_1 = 0.3$, $T_2 = 0.5$, $T_3 = 0.7$.

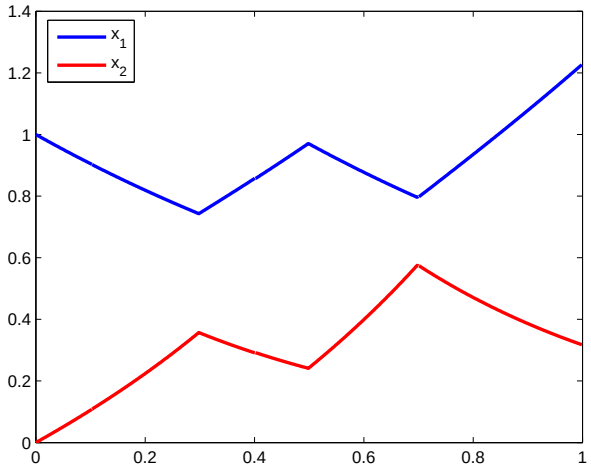


Figure 25: State Trajectory of a Switched Autonomous System (example).

7.2 Necessary Optimality Conditions

As stated, the problem is a parameter optimization problem. Solving it, as such, requires the explicit solution of the state equations, and their dependency on $T_1, T_2, ..., T_{N-1}$. In

the previous example, an explicit solution can be found, but we are interested in the more general case where such solutions are not available. We, therefore, solve the problem using classical variational methods. The idea is that analytic expressions for the partial derivatives of the cost function with respect to the control variables can be obtained by looking at the cost variation induced by an infinitely small perturbation in the control variables. A necessary condition for optimality is the cancellation of all these partial derivatives. We show here a detailed derivation.

Consider the two following systems:

- an "unperturbed system" with control variables $T = (T_1, ...T_{N-1})$, state trajectory $x(t)$, and performance index $J(T) = \int_{T_0}^{T_N} L(x(t))dt$.
- a "perturbed system" obtained from the previous one when the control variables are slightly modified. The control variables are now $T + \epsilon\theta = (T_1 + \epsilon\theta_1, ..., T_{N-1} + \epsilon\theta_{N-1})$ with $\epsilon << 1$, the new state trajectory is $x(t) + \epsilon\eta(t)$, and the new performance index is $J(T + \epsilon\theta) = \int_{T_0}^{T_N} L(x(t) + \epsilon\eta(t))dt$.

We first adjoin the dynamical equations of the system within each mode via a Lagrange multiplier λ defined on each $[T_{i-1}, T_i)$. The cost equations now write

$$\begin{aligned} J(T) &= \sum_{i=1}^{N} \int_{T_{i-1}}^{T_i} [L(x(t)) + \lambda(f_i(x(t)) - \dot{x}(t))]dt \\ J(T+\epsilon\theta) &= \sum_{i=1}^{N} \int_{T_{i-1}+\epsilon\theta_{i-1}}^{T_i+\epsilon\theta_i} [L(x(t) + \epsilon\eta(t)) + \lambda(f_i(x(t) + \epsilon\eta(t)) - (\dot{x}(t) + \epsilon\dot{\eta}(t)))]dt, \end{aligned}$$

where $\theta_0 = \theta_N = 0$. If we let $A = \sum_{i=1}^{N} \int_{\tau_{i-1}+\epsilon\theta_{i-1}}^{\tau_i+\epsilon\theta_i} [L_i(x) + \lambda(f_i(x) - \dot{x})]dt$, then

$$\begin{aligned} J(T) &= \sum_{i=1}^{N} \int_{T_{i-1}}^{T_{i-1}+\epsilon\theta_{i-1}} [L(x) + \lambda(f_i(x) - \dot{x})]dt + A - \sum_{i=1}^{N} \int_{T_i}^{T_i+\epsilon\theta_i} [L(x) + \lambda(f_i(x) - \dot{x})]dt \\ &= \sum_{i=0}^{N-1} \int_{T_i}^{T_i+\epsilon\theta_i} [L(x) + \lambda(f_{i+1}(x) - \dot{x})]dt + A - \sum_{i=1}^{N} \int_{T_i}^{T_i+\epsilon\theta_i} [L(x) + \lambda(f_i(x) - \dot{x})]dt \\ &= A + \sum_{i=1}^{N-1} \epsilon\theta_i [L(x) - L(x) + \lambda(f_{i+1}(x) - f_i(x))]|_{t=T_i^+} + o(\epsilon) \\ &= A - \sum_{i=1}^{N-1} \epsilon\theta_i \lambda(T_i^+)[f_{i+1}(x(T_i)) - f_i(x(T_i))] + o(\epsilon) \end{aligned}$$

and

$$\begin{aligned} J(T+\epsilon\theta) &= A+\epsilon\sum_{i=1}^{N}\int_{T_{i-1}}^{T_i}[\frac{\partial L_i}{\partial x}\eta+\lambda(\frac{\partial f_i}{\partial x}\eta-\dot{\eta})]dt+o(\epsilon) \\ &= A+\epsilon\sum_{i=1}^{N}\int_{T_{i-1}}^{T_i}[\frac{\partial L_i}{\partial x}+\lambda\frac{\partial f_i}{\partial x}+\dot{\lambda}]\eta\,dt-\epsilon\sum_{i=1}^{N}[\lambda\eta]_{\tau_{i-1}}^{T_i}+o(\epsilon) \end{aligned}$$

By substraction and keeping only the first-order elements, we get the directional derivative (in the θ-direction)

$$\begin{aligned} \nabla J_\theta &= \lim_{\epsilon\to 0}\frac{J(T+\epsilon\theta)-J(T)}{\epsilon} \\ &= \sum_{i=1}^{N}\int_{T_{i-1}}^{T_i}[\frac{\partial L}{\partial x}+\lambda\frac{\partial f_i}{\partial x}+\dot{\lambda}]\eta\,dt-\sum_{i=1}^{N}[\lambda\eta]_{T_{i-1}}^{T_i}+\sum_{i=1}^{N-1}\theta_i\lambda(T_i^+)\Big[f_{i+1}(x(T_i))-f_i(x(T_i))\Big]. \end{aligned}$$

At this point, we can choose λ so that all η-terms disappear. In other words, for a particular choice of the Lagrange multipliers, no variation in the state trajectory need to be computed (this is the purpose of the Calculus of Variation method, which avoids the explicit computation of the induced state variation). If we choose λ continuous on (T_0, T_N) and such that

$$\begin{cases} \dot{\lambda}(t)=-\frac{\partial L}{\partial x}(x(t))-\lambda(t)\frac{\partial f_i}{\partial x}(x(t)) \qquad t\in[T_{i-1},T_i], \quad i=1,\ldots,N \\ \lambda(T_N)=0 \end{cases}$$

We are then left with

$$\nabla J_\theta = \sum_{i=1}^{N-1}\theta_i\lambda(T_i)\Big[f_{i+1}(x(T_i))-f_i(x(T_i))\Big].$$

By identification with $\nabla J_T=\sum_{i=1}^{N-1}\frac{\partial J}{\partial T_i}\theta_i$, we get an expression for the partial derivatives of J with respect to the control variables. A necessary condition for optimality is that the directional derivative is zero in every possible direction $\theta=(\theta_1,\ldots\theta_{N-1})$, where the θ_i are independent. This is equivalent to having all the partial derivatives $\frac{\partial J}{\partial T_i}, i=1,\ldots,N-1$ equal to zero. We summarize these results in the theorem below:

Theorem 7.2.1 *The system with equations*

$$\begin{cases} x(T_0)=x_0, \\ \dot{x}(t)=f_i(x(t)), \quad t\in(T_{i-1},T_i), \quad i=1,\ldots,N, \end{cases} \tag{14}$$

makes the performance index

$$J = \int_{T_0}^{T_N} L(x(t))dt \tag{15}$$

stationary if the switching times $T_1, ..., T_{N-1}$ *are chosen as follows:*

Euler Lagrange Equations

$$\begin{cases} \dot{\lambda}(t) = -\frac{\partial L}{\partial x}(x(t)) - \lambda(t)\frac{\partial f_i}{\partial x}(x(t)) & t \in [T_{i-1}, T_i), \quad i = 1, \ldots, N \\ \lambda(T_N) = 0 \end{cases} \tag{16}$$

Optimality conditions:

$$\frac{\partial J}{\partial T_i} = \lambda(T_i)\Big[f_{i+1}(x(T_i)) - f_i(x(T_i))\Big] = 0, \qquad i = 1, ...N-1. \tag{17}$$

7.3 Numerical Algorithm

The reason why the formula derived above are particularly easy to work with is that they give us access to a very straight-forward numerical algorithm.

Algorithm 7.3.1 ***Steepest Descent Algorithm***

1) Start with an initial guess for the control vector $\overline{T} = (T_1, ...T_N - 1)$.

2) Numerically compute the state $x(t)$ *forward in time from* T_0 *to* T_N *using Equation (14).*

3) Numerically compute the co-state $\lambda(t)$ *backward in time from* T_N *to* T_0 *using Equation (16).*

4) Compute the gradient $\nabla J(\overline{T}) = (\frac{\partial J}{\partial T_1}, ..., \frac{\partial J}{\partial T_{N-1}})$ *using Equation (17)*

5) Update the control vector in the direction of negative gradient, i.e replace $\overline{T}$ *by* $\overline{T} - \gamma\nabla J(\overline{T})$, *where* $\gamma > 0$ *is the* stepsize.

6) Repeat steps 2 to 5 until a convergence criterion is met, e.g., $||\nabla J(\overline{T})|| < \epsilon$ *for some small* $\epsilon > 0$.

The fact that the same state $x(t)$ and costate $\lambda(t)$ are used to compute all the partial derivatives $\frac{\partial J}{\partial T_i}$ results in a fast gradient computation. Given a reasonable convergence

criterion and a judicious stepsize, the algorithm converges to a local minimum within a few iterations. For different initial guesses of the control variables, the algorithm may converge to different local minima. Often, an a priori understanding of the optimization problem will help determine a "clever" guess, which is likely to converge to the global minimum. In the absence of such knowledge, the common strategy is to try different executions of the algorithm for different initial guesses. The fast convergence of the algorithm particularly allows such heuristic methods.

Example:

We now show how Algorithm 7.3.1 is successfully applied to the example introduced earlier. The objective is to find switching times minimizing the performance index $J = \int_0^1 ||x(t)||^2 dt$. Starting with $T_1 = 0.3$, $T_2 = 0.50$, $T_3 = 0.8$, the algorithm quickly converges to the solution $T_1^* \approx 0.38$, $T_2^* \approx 0.51$, $T_3^* \approx 0.72$. Figures 26(a)(b)(c) show the value of the cost function J, the control variables T_1, T_2 and T_3, and the L_2-norm of J at each iteration. Figure 26(d) compares the state trajectories at the beginning (dotted) and end (solid) of the execution.

At every iteration, the new control vector was computed using the Armijo stepsize. By providing a suboptimal solution to the problem of minimizing J in the direction of $-\nabla J$, the Armijo stepsize, introduced in [4], ensures a fast convergence both in the number of iterations and the overall computation time.

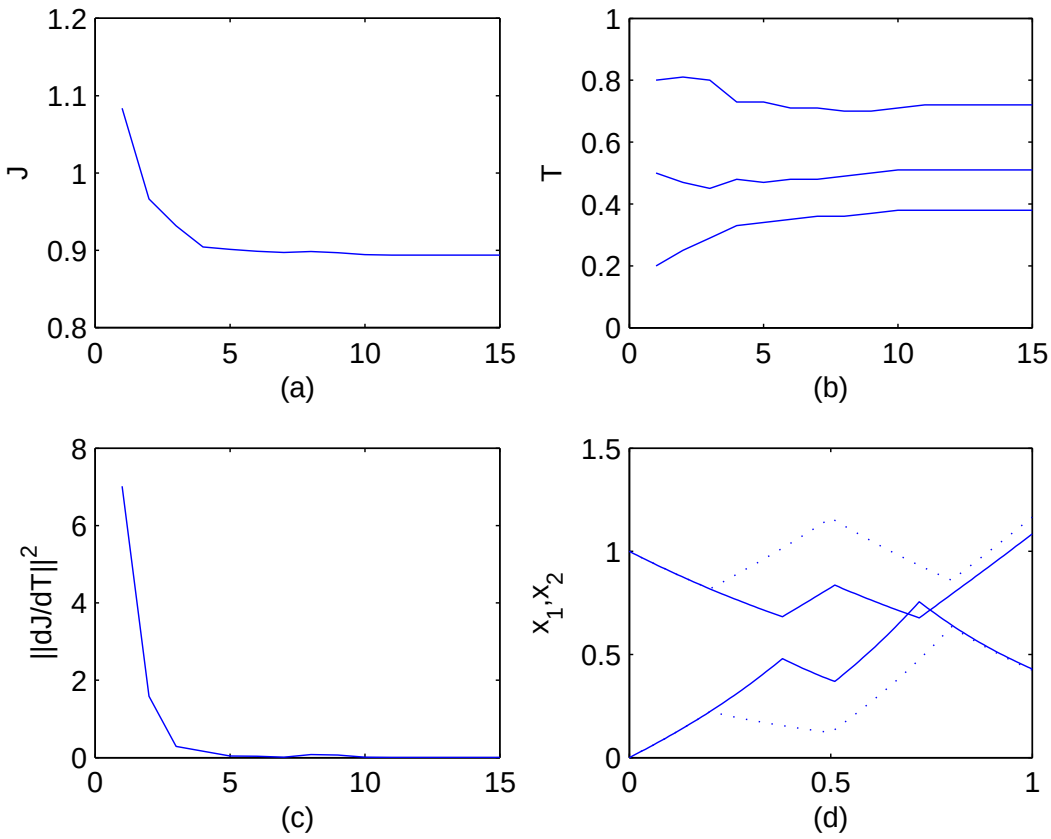


Figure 26: Optimal Switching Times Using Gradient Descent Algorithm (example).

CHAPTER VIII

APPLICATIONS AND GENERALIZATIONS

In this chapter, the method presented in Chapter 7 is used in a variety of applications. Although the approach remains the same, we progressively bring more complexity in the systems dynamics (jumps at the switching times, delays, etc) and performance indices (final cost, switching costs, distance to a given curve, etc). Accordingly, the numerous publications [26, 27, 29, 82, 83, 84] based on this work span a quite diverse number of applications including robot navigation, curve fitting, data compression, control of epidemics, etc.

8.1 Optimal Sample Time Selections for Interpolation and Smoothing

Here we are interested in the ability of the switched autonomous system to follow a given curve. This can be interpreted as maximizing the performance as defined in Chapter 4. To this end, a new cost function $J = \int_{T_0}^{T_N} L(x(t), h(t))dt$ is chosen, where $h(t)$ is the parameterized curve to approximate. For example, by setting $L(x(t), h(t)) = (x(t) - h(t))^2$, we intend to minimize the mean squared error between the original curve and its hybrid approximation. Also, to comply with the dynamics of many curve fitting models, we now let $\dot{x}(t) = f_i(x, t, T)$, where $T = [T_0, T_1, ..., T_N]$. Note how the trajectory of the generated system now explicitly depends on the switching times. By following the same development as above, we establish the theorem below

Theorem 8.1.1 *The system with equations*

$$\begin{cases} x(T_0) = x_0, \\ \dot{x}(t) = f_i(x(t), t, T), \quad t \in (T_{i-1}, T_i), \quad i = 1, ..., N, \end{cases} \tag{18}$$

makes the performance index

$$J = \int_{T_0}^{T_N} L(x(t), h(t))dt \tag{19}$$

stationary if the switching times $T_1, ..., T_{N-1}$ are chosen as follows:

Euler Lagrange Equations

$$\begin{cases} \dot{\lambda}(t) = -\frac{\partial L}{\partial x}(x(t), h(t)) - \lambda(t)\frac{\partial f_i}{\partial x}(x(t), t, T) \qquad t \in [T_{i-1}, T_i), \quad i = 1, \ldots, N \\ \lambda(T_N) = 0 \end{cases} \tag{20}$$

Optimality conditions:

$$\frac{\partial J}{\partial T_i} = \int_{T_0}^{T_N} \lambda(t)\frac{\partial f_\xi(t)}{\partial x}(x(t), t, T)dt + \lambda(T_i)\Big[f_{i+1}(x(T_i)) - f_i(x(T_i))\Big] = 0, \;\; i = 1, ..., N{-}1. \tag{21}$$

We now apply the previous theorem together with Algorithm 7.3.1 for the optimal sample selection in two curve fitting applications.

8.1.1 Linear Approximation

In this simple application, a curve is approximated by drawing straight lines between points on the curve. Assuming the function to approximate is $h : [t_0, t_f] \to \mathbb{R}$, the approximating function x is such that for $i = 1, ..., N + 1$ and $\forall t \in [\tau_{i-1}, \tau_i)$

$$x(t) = h(T_{i-1}) + (t - T_{i-1})\frac{h(T_i) - h(T_{i-1})}{T_i - T_{i-1}}.$$

The state derivative on each segment $[T_{i-1}, T_i)$ only depends on the switching times T_{i-1} and T_i

$$\dot{x}(t) = f_i(T_{i-1}, T_i) = \frac{h(T_i) - h(T_{i-1})}{T_i - T_{i-1}} \quad \text{on } [T_{i-1}, T_i).$$

We now apply the developed algorithm to the problem of determining $T_1, ..., T_{N-1}$ in order to minimize the cost function

$$J = \int_{T_0}^{T_N} (h(t) - x(t))^2 dt.$$

Figure 27 shows how the algorithm converges. The following parameters were used:

$$\begin{cases} h(t) = 5\sin(\frac{2\pi t}{300}) + 3\sin(\frac{2\pi t}{100}) + \frac{t^2}{20000} - \frac{t}{50} \\ \\ [t_0, t_f] = [0, 200], \quad N = 5 \quad \text{and} \quad \gamma = 1. \end{cases}$$

The lowest picture in Figure 27 shows how fast the algorithm converges. The optimal solution is reached after very few iterations, in spite of a "bad" initial guess and a constant step size l.

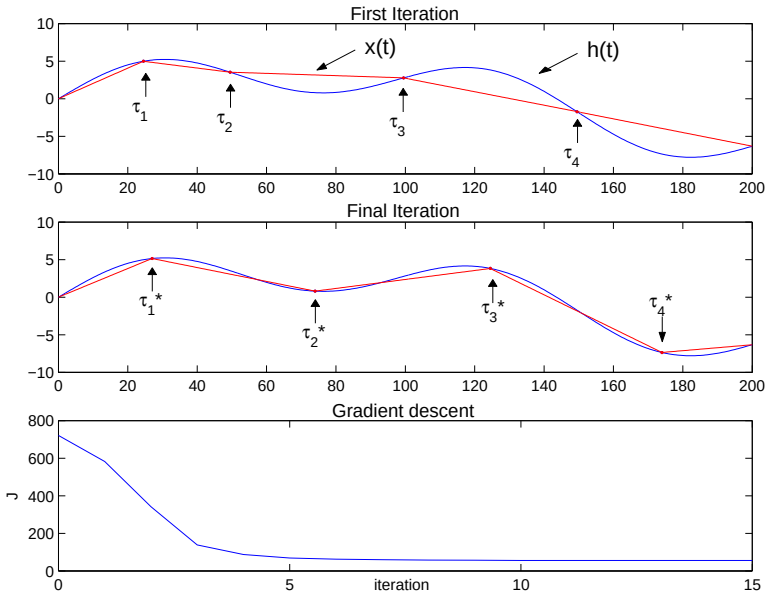


Figure 27: Optimal Linear Approximation.

8.1.2 Generalized Smoothing Splines

Generalized smoothing splines bring an optimal control flavor to data interpolation . In a recent paper [78], Sun shows how the input of a single-input single-output linear system

$$\begin{aligned} \dot{x} &= Ax + Bu,\ x \in \mathbb{R}^n,\ u \in \mathbb{R} \\ y &= Cx,\ y \in \mathbb{R}. \end{aligned} \tag{22}$$

should be chosen in order to drive the output of the system close to a given sequence of data points. These data points are given by $(T_1, \xi_1), ..., (T_{N-1}, \xi_{N-1})$ where the $\xi_i \in \mathbb{R}$ are measurements obtained from $\xi_i = h(T_i) + \epsilon_i$, given some underlying curve $h(t)$, corrupted by measurement noise ϵ_i, $i = 1, ...N-1$. The control u is chosen to minimize a cost function

$$\int_{T_0}^{T_N} u^2(t)dt + \rho \sum_{i=1}^{N-1} \omega_i (y(T_i) - \xi_i)^2. \tag{23}$$

Here, the parameters $\omega_i > 0$ allow us to choose the relative importance of interpolating each data point. The *smoothing parameter* ρ is a more interesting tuning parameter. When ρ is high, more importance is paid in closely approximating (interpolating) the ξ_i. On the contrary, when ρ is low, more importance is paid in generating a slowly varying (smooth) approximation. The effect of ρ on the interpolation/smoothing is shown in Figure 28 (borrowed from [78]), where two generalized smoothing splines are constructed for the same set

of data points, but for different values of ρ.

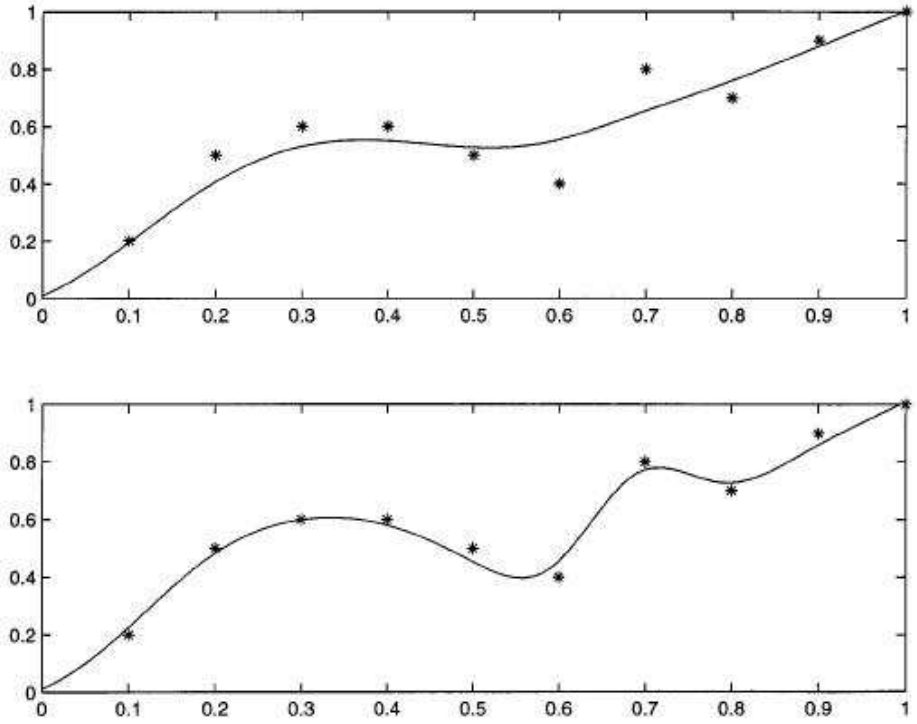


Figure 28: Generalized Smoothing Splines with Different ρ Parameter.

This method of using smoothing generalized splines has for instance been applied by [58] to the production of calligraphic Japanese characters. Three decoupled systems with scalar inputs and outputs are used. The output of the first system corresponds to the x-position of the brush, the second to its y-position, and the third to the thickness of the stroke. The idea is to let the data points encode these variables at strategically selected points, with quite remarkable results. However, it is not clear how exactly these points should be selected. Here, we can apply the results of theorem 8.1.1 for the optimal sample selection.

Given the dynamics in Equation (22), the (unique) optimal solution to the problem in Equation (23) was in [78] found to be

$$u(t) = \gamma(t)^T (I + \mathcal{W}\Gamma)^{-1} \mathcal{W}\xi,$$

where

$$\mathcal{W}=\begin{pmatrix} \omega_1 & 0 & \cdots & 0 \\ 0 & \omega_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega_N \end{pmatrix},\ \xi=\begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{pmatrix}=\begin{pmatrix} h(\tau_1) \\ h(\tau_2) \\ \vdots \\ h(\tau_N) \end{pmatrix},$$

$$\gamma(t)=\begin{pmatrix} \gamma_1(t) \\ \gamma_2(t) \\ \vdots \\ \gamma_N(t) \end{pmatrix},\ \gamma_i(t)=\begin{cases} Ce^{A(\tau_i-t)}B & \text{if } t\le\tau_i \\ 0 & \text{otherwise,} \end{cases}$$

and where the Grammian Γ is given by

$$\Gamma=\int_{t_0}^{t_f}\gamma(s)\gamma(s)^T ds\in\mathbb{R}^{N\times N}.$$

Note that the definition of the basis functions $\gamma_i(t)$ implies that u may be discontinuous at τ_i. In fact, we could define a new set of basis functions

$$\zeta_i(t)=\begin{pmatrix} 0 \\ \vdots \\ 0 \\ Ce^{A(\tau_i-t)}B \\ Ce^{A(\tau_{i+1}-t)}B \\ \vdots \\ Ce^{A(\tau_N-t)}B \end{pmatrix},\ t\in[\tau_{i-1},\tau_i),\ i=1,\ldots,N,$$

with $\zeta_{N+1}=0$. Hence we have the new system

$$\begin{aligned} \dot{x} &= Ax+Bu \\ &= Ax+B\zeta_i^T(t,\tau)(I+\mathcal{W}\Gamma(\tau))^{-1}\mathcal{W}\xi(\tau) \\ &\triangleq f_i(x,t,\tau), t\in[\tau_{i-1},\tau_i) \end{aligned}$$

that is of the prescribed form. With regards to this system, we choose the following performance index:

$$J=\int_{T_0}^{T_N}(Cx(t)-h(t))^2dt \quad \Big(i.e., L(x(t),h(t))=(Cx(t)-h(t))^2\Big).$$

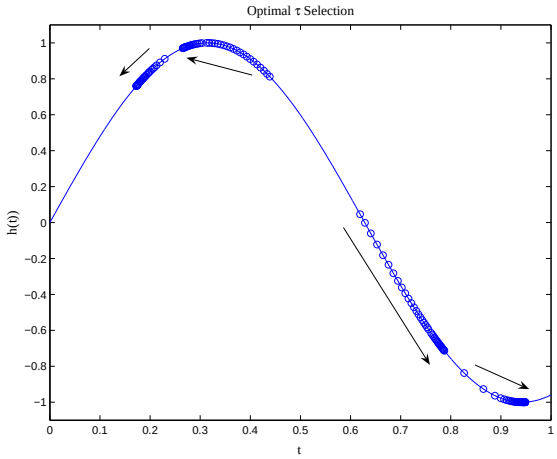


Figure 29: Evolution of the Sample Times when Creating Smoothing Splines for the Underlying Curve $h(t) = \sin(5t)$.

Numerical Simulation

We apply this method to the system

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \; B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, C = (1, 0, 0),$$

which gives the standard cubic smoothing spline.

Results from applying the gradient descent method using the Armijo step-size over 40 iterations are shown in Figures 29-31. In that example, the underlying curve was given by $h(t) = \sin(5t)$, and four sample times where selected with $\omega_i = 1$, $i = 1, \ldots, 4$.

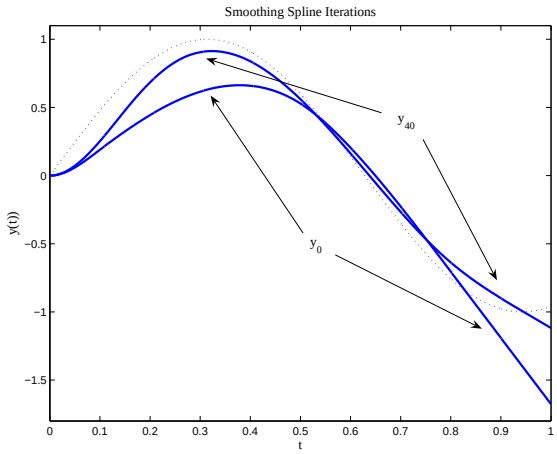


Figure 30: Smoothing Splines at the First and Last (40th) Iterations.

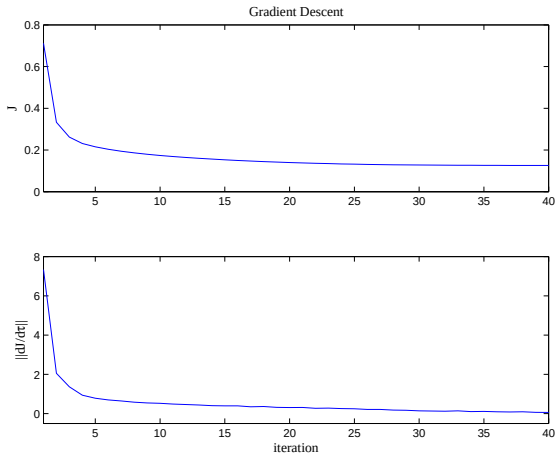


Figure 31: Evolutions of $J(\tau(k))$ and $\|dJ/d\tau(\tau(k))\|$.

8.2 Optimal Control of Switched Delay Systems

In this section, we extend the results of Theorem 7.2.1 to systems with delays. Without loss of generality, we consider systems having a single crisp delay (point delay) τ and with separable continuous modes defined by the functional differential equations:

$$\dot{x}(t) = f_i(x(t)) + g_i(x(t-\tau)) \tag{24}$$

We first study the effect of a perturbation in the switching time ($T_i \rightarrow T_i + \epsilon\theta_i$). It should be noted that the following analysis (in particular, Theorem 8.2.1, Tables 4 and 5, and their interpretation) is the work of Professor Erik Verriest [82].

Definition 8.2.1 *A function y is said to be C_k at t_0 if the k^{th} derivative of y is continuous at t_0, but the $k+1^{st}$ is not.*

Assume that a single controlled switch occurs at time T_i, switching from mode i to $i+1$. Also, assume that the state variable x is continuous at every mode switch. This makes $\dot{x}(t)$ discontinuous at T_i. Consequently, $x(t)$ and $f(x(t))$ have a 'kink' (i.e. are non-differentiable). From the continuity assumption, $x(t)$ and $f(x(t))$ are C_0 at T_i. But then $x(t-\tau)$ and $g(x(t-\tau))$ are C_0 at $T_i + \tau$. In turn, this implies that $\dot{x}(t)$ is C_0 at $T_i + \tau$, inducing again C_1 behavior in $x(t)$ and $f(x(t))$ at $T_i + \tau$ and C_1 behavior in $x(t-\tau)$ and $g(x(t-\tau))$ at time $T_i + 2\tau$, and so on. We summarize the chain of events in the following table:

Table 4: Delay System Behavior Induced by a Continuous Mode Switch at T_i.

	T_i	$T_i+\tau$	$T_i+2\tau$	...	$T_i+k\tau$
$\dot{x}(t)$	jump	C_0	C_1	...	C_{k-1}
$x(t)$	C_0	C_1	C_2	...	C_k
$f(x(t))$	C_0	C_1	C_2	...	C_k
$x(t-\tau)$		C_0	C_1	...	C_{k-1}
$g(x(t-\tau))$		C_0	C_1	...	C_{k-1}

We now establish a simple result:

Theorem 8.2.1 *If y is C_k at t_0, then the variation of y induced by the perturbation $t_0 \rightarrow t_0 + \epsilon\theta$ is of order $k+1$ in $\epsilon\theta$.*

Proof: We note that the variation involves only the effect due to the shift of the switch of behavior from t_0 to $t_0 + \theta$. Since this is only visible in the $(k+1)$-st derivative,

$$\Delta_{t_0} y \triangleq y_+(t_0^+) - y_-(t_0^-) \approxeq [y_+^{(k+1)} - y_-^{(k+1)}] \frac{(\epsilon\theta)^{k+1}}{(k+1)!}.$$

■

In particular, if y is a continuous integrand, the first order variation of its integral (as in the performance index) will not involve 'future' additional contributions due to the propagation effect of the delay. Such additional contributions do occur however, if the integrand has jumps, as with impulsive control. Table 5 shows indeed that, in the case of impulsive control, x is C_0 at $T_i + \tau$. The first order variation of its integral will thus involve additional contributions at all instants $T_i + \tau$.

Table 5: Delay System Behavior Induced by an Impulsive Mode Switch at T_i.

	T_i	$T_i + \tau$	$T_i + 2\tau$	...	$T_i + k\tau$
$\dot{x}(t)$	jump	jump	C_0	...	C_{k-2}
$x(t)$	jump	C_0	C_1	...	C_{k-1}
$f(x(t))$	jump	C_0	C_1	...	C_{k-1}
$x(t-\tau)$		jump	C_1	...	C_{k-2}
$g(x(t-\tau))$		jump	C_1	...	C_{k-2}

In the case of an impulsive control (studied in the next section), if a consecutive switch T_{i+1} happens before $T_i + \tau$, another $\epsilon\theta$ perturbation will be induced before all $\epsilon\theta$ perturbations induced by the previous switch at T_i are generated. Therefore, the bookkeeping of all perturbation terms may get quite complicated, in this more general case, especially in view of the fact that all possibilities (of relative positions of switching instants) need to be taken into account. To avoid such bookkeeping, we will require that $\forall i$, $T_{i+1} > T_i + \tau$. In other words, we will assume that the systems considered all have a refractory period, in the sense that once an action is taken, it takes a non-infinitesimal amount of time (here equal to the delay τ) before a subsequent action can be taken. Refractory periods are ubiquitous in physiological systems (e.g., neural spike propagation), electrical systems (e.g., time to recharge a capacitor), and manufacturing systems (e.g., refurbishing, restocking) to name a few. Refractory periods also provide a safeguard towards unwanted high frequency switching.

In the case of a continuous switch (studied in this section), the immediate effect of a variation in a single switch T_i is represented in Figure 32. Here, the induced perturbation

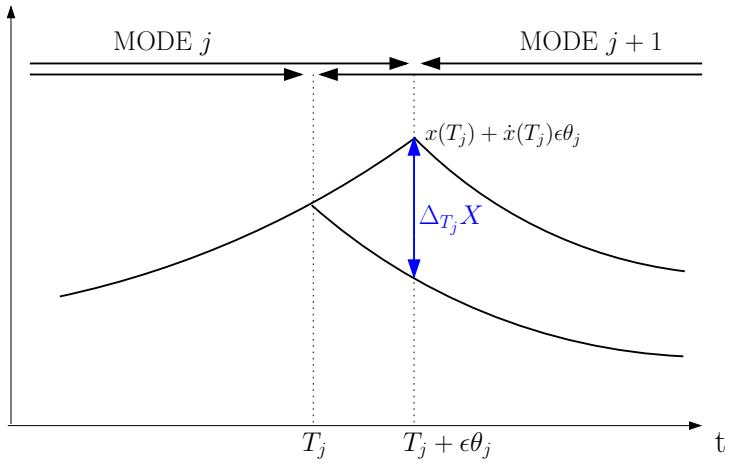


Figure 32: Optimal Control of Continuous Switched Systems: Induced Perturbation at a Switch.

$\Delta_{T_i} x$ is propagated to consecutive modes. As each subsequent switch will add a similar term, it is clear that the effects of such perturbations will accumulate in subsequent modes, and keeping track of all these effects will complicate a derivation requiring the explicit computation of perturbations. In keeping with the philosophy of calculus of variations, we shall avoid having to keep track of these by introducing independent Lagrange multipliers for each subinterval. As we showed in the derivation of necessary conditions for the non-delay case in the previous section, these can be chosen in a very convenient way in order to avoid computation of the induced variations. A similar method results in the theorem below. For a detailed derivation, please refer to [82].

Theorem 8.2.2 *The separable mode switched system*

$$\dot{x}(t) = f_i(x(t)) + g_i(x(t-\tau)) \tag{25}$$

makes the performance index

$$J = \sum_{i=1}^{N} \int_{T_{i-1}}^{T_i} L_i(x(t))dt + \Phi(x(T)) \tag{26}$$

stationary with fixed initial time T_0 and terminal time T_N if the switching times T_i, $i = 1, \ldots, N-1$ are chosen to satisfy:

Euler-Lagrange Equations: $T_{i-1} < t < T_i;\ i = 1, \ldots, N-1,$

$$\dot{\lambda}_i = -\left(\frac{\partial L_i}{\partial x}\right)^T - \left(\frac{\partial f_i}{\partial x}\right)^T \lambda_i - \chi_i^+ \left(\frac{\partial g_i}{\partial x}\right)^T \lambda_i^\tau - \chi_{i+1}^- \left(\frac{\partial g_{i+1}}{\partial x}\right)^T \lambda_{i+1}^\tau, \tag{27}$$

$$\dot{\lambda}_N = -\left(\frac{\partial L_N}{\partial x}\right)^T - \left(\frac{\partial f_N}{\partial x}\right)^T \lambda_N - \chi_N^+ \left(\frac{\partial g_N}{\partial x}\right)^T \lambda_N^\tau. \tag{28}$$

where $\lambda_i^\tau = \lambda_i(t+\tau)$ *and the* χ *are indicator functions:*

$\chi_i^+(t) = 1$ *if* $t \in [T_{i-1}, T_i - \tau]$, *and* 0 *otherwise,*

$\chi_{i+1}^-(t) = 1$ *if* $t \in [T_i - \tau, T_i]$, *and* 0 *otherwise,*

$\chi_N^+ = 0$ *is understood if* $T_{N-1} > T - \tau$.

Boundary Conditions:

$$\lambda_N(T_N) = \left(\frac{\partial \Phi}{\partial x}\right)^T \tag{29}$$

$$\lambda_i(T_i^-) = \lambda_{i+1}(T_i^+) \tag{30}$$

Optimality Conditions:

$$\frac{\partial J}{\partial T_i} = H_i(T_i^-) - H_{i+1}(T_i^+) \tag{31}$$

where $H_i = L_i(x) + \lambda_i(f(x) + g(x_\tau))$ *and* $x_\tau(t) = x(t-\tau)$ *(Hamiltonians).*

Note how, in Equation (26), we added to the generality of the original cost function in Equation (15) by 1) considering a running cost L depending on the number of past impulses (i.e. L_i instead of L) 2) adding a final cost $\Phi(x(T))$.

Example:

For the simple delay system ($\tau = 1$) with modes

$$\dot{x}(t) = 1 - x(t-1)$$

$$\dot{x}(t) = -1 + x(t-1)$$

and initial condition $x(t) = 0$ for $t < 0$, the performance index

$$J = \frac{1}{2}\int_0^5 [x(t) - 1.5]^2\, dt$$

is evaluated explicitly for two switches. The gradient algorithm converges in a few steps. The state trajectory of the solution is shown in Figure 33 for the *optimal* mode sequence $1 \rightarrow 2 \rightarrow 1$. Figure 34 shows the cost descent (in green) in the $T1 - T2$ plane, where closed curves represent iso-J and the red line delimits the area of the graph where the refractory period assumption is respected.

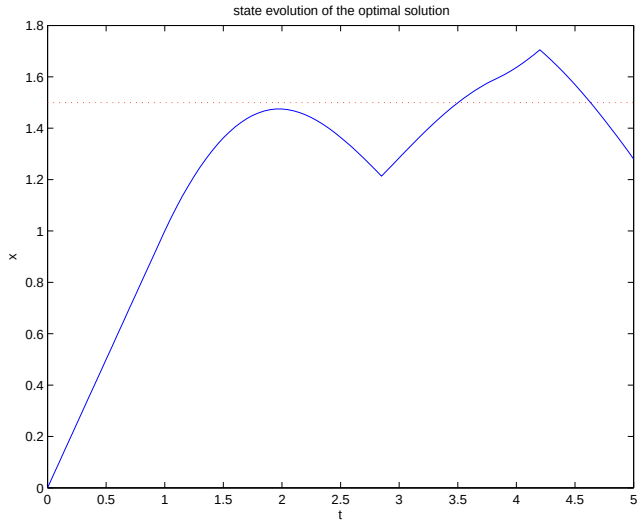


Figure 33: State Trajectory of the Optimal Solution.

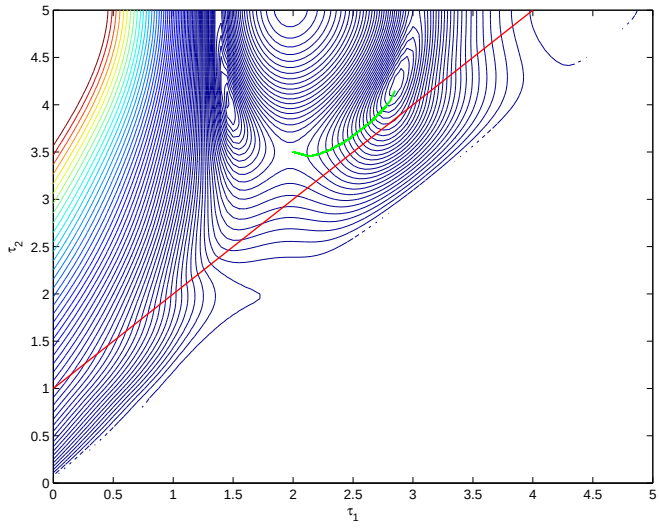


Figure 34: Gradient Descent.

8.3 Optimal Impulsive Control of Switched Autonomous Systems

In this section, we push the generalization a little further by now considering systems with discontinuities at the switching times. We assume these discontinuities are given by a set of *jump amplitude* functions $\{G_i\}_{i=1}^{N-1}$ such that

$$x(T_i^+) = x(T_i^-) + G_i(x(T_i^-), u_i, T_i), \quad i = 1, ..., N-1. \tag{32}$$

In this equation, the *jump variables* $u_1, ..., u_{N-1}$ constitute a new set of control variables, in addition to the switching times $T_1, ..., T_{N-1}$. Equation (32) allows for a diversity of discontinuities including resets ($G_i = -x(T_i^-)$) and free jumps ($G_i = u_i$). This problem is also more general in the sense that setting $G_i = 0$ should produce the same results as in the previous sections.

As before, we assume that the switched autonomous systems under consideration present delayed dynamics:

$$\dot{x}(t) = f_i(x(t)) + g_i(x(t-\tau)), \quad t \in (T_{i-1}, T_i), \quad i = 1, ..., N. \tag{33}$$

Finally, the cost function

$$J = \sum_{i=1}^{N} \int_{T_{i-1}}^{T_i} L_i(x(t))dt + \sum_{i=1}^{N-1} K_i(x(T_i^-), u_i, T_i) + \Phi(x(T_N)) \tag{34}$$

presents additional terms K_i ($i = 1, ..., N-1$) accounting for punctual costs associated with the jumps (Note that the notations can be eased by adjoining the final cost to these punctual costs, i.e., set $K_N(x(T_N^-), u_N, T_N) = \Phi(x(T_N))$).

Figure 35 shows the induced state variation at T_i and $T_i + \tau$ when a slight variation $T_i \to T_i + \epsilon\theta_i$ is applied.

Figure 35 illustrates what was mentioned in the previous section: the jump at T_i results in a 'kink' (C_0-behavior) at $T_i + \tau$. The first order of the cost variation will thus involve contributions at T_i and $T_i + \tau$. Both contributions are propagated through the next modes. Also, each variation in the control variable $u_i \to u_i + \epsilon\nu_i$ contributes to two cost variations: an immediate one through K, and an indirect one through the integration (via L) of an

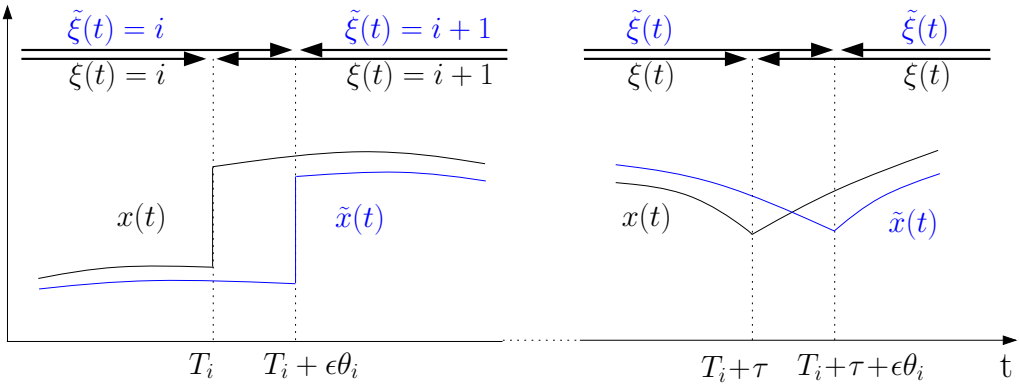


Figure 35: Optimal Impulsive Control of Switched Systems: Induced Perturbation at T_i and $T_i + \tau$.

induced state variation. Again, to avoid an explicit computation of such variations, we adjoin the dynamic equations in (33) and discontinuity equations (32) to the cost equation (34) via two sets of Lagrange multipliers λ_i and μ_i, $i = 1, ..., N$. Within the Calculus of Variations framework, a convenient choice of these Lagrange multipliers leads to an amenable necessary condition for optimality. Again, we choose to present just the final theorem. A full derivation can be found in [83].

Theorem 8.3.1 *The impulsive system in Equations (32)-(33) attains a stationary point of the performance index J in Equation (34) if the jump variables u_i, and switching times T_i are chosen as follows:*

Define:

$$H_i = L_i + \lambda_i[f(x) + g(x_\tau)] \tag{35}$$

$$M_i = K_i + \mu_i G_i \tag{36}$$

Euler-Lagrange Equations

$$\begin{aligned} \dot{\lambda}_i = &-\left(\tfrac{\partial L_i}{\partial x}\right)' - \left(\tfrac{\partial f_i}{\partial x}\right)' \lambda_i - \chi_i^+ \left(\tfrac{\partial g_i}{\partial x}\right)' \lambda_i^\tau \\ &-\chi_{i+1}^- \left(\tfrac{\partial g_{i+1}}{\partial x}\right)' \lambda_{i+1}^\tau \\ \textit{with} \quad & T_{i-1} < t < T_i, \quad i = 1, \ldots, N-1 \\ \dot{\lambda}_N = &-\left(\tfrac{\partial L}{\partial x}\right)' - \left(\tfrac{\partial f_N}{\partial x}\right)' \lambda_N - \chi_N^+(t) \left(\tfrac{\partial g_N}{\partial x}\right)' \lambda_N^\tau \end{aligned} \tag{37}$$

Boundary Conditions

$$\lambda_N(T_N) = 0, \tag{38}$$

$$\lambda_i(T_i^-) = \lambda_{i+1}(T_i^+) + \left(\frac{\partial M_i}{\partial x}\right)', \quad i = 1 \dots N \tag{39}$$

Multipliers

$$\mu_i = \lambda_{i+1}(T_i^+), \quad i = 1 \dots N \tag{40}$$

Optimality Conditions

$$\frac{\partial J}{\partial u_i} = \frac{\partial M_i}{\partial u} = 0 \tag{41}$$

$$\frac{\partial J}{\partial T_i} = \frac{\partial M_i}{\partial T_i} + H_i(T_i^-) - H_{i+1}(T_i^+) + \lambda_{i+1}(T_i + \tau)[g(X(T_i^+) - g(X(T_i^-)] \tag{42}$$

To optimally solve for both the switching times T_i and the jump parameters u_i, the partial derivatives in the theorem above are then used in a modified version of the gradient-descent algorithm 7.3.1, now accounting for the two sets of control variables.

The system in Equations (32) and (33) can fit many applications. We choose to present two of them.

Application 1 - Control of Epidemics:

In epidemiology, a common and successful approach for modeling and simulating epidemic outbreaks is to divide an entire population into a finite number of relevant classes or *compartments* [22, 46]. Perhaps the most famous example, the SIR model, used to model infectious diseases such as measles, mumps and rubella, is represented in Figure 36. The dynamic equations modeling the flux of populations from one class to another are usually nonlinear.

Some diseases require more complex models, i.e., more compartments to account for their specific transmission (genetically, sexually, vector transmitted diseases, etc). In most models, the characteristic times of a disease, e.g., incubation period, recovery period, or immunization period, are represented via transition rates. More realistic models integrate

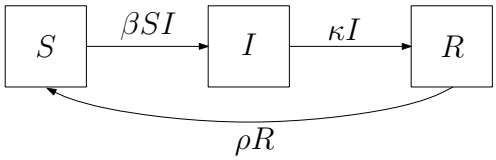


Figure 36: SIR Model.

delays, but they bring a lot more complexity to their analysis. Figure 37 shows a model for malaria found in [2], where two populations 1 and 2 (respectively humans and female mosquitoes, the "vectors") are each divided into three compartments S for susceptible, E for exposed, and I for infected. The dynamic equations of all compartments are coupled, nonlinear, and involve two delays representing the incubation period for each population.

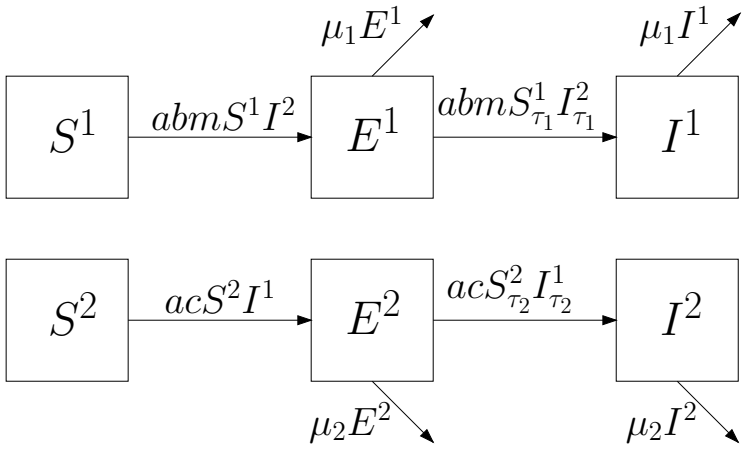


Figure 37: Two-Population Two-Delay Malaria Model.

For such models, many prevention or cure techniques (quarantine, vaccination, vector eradication, etc.) can be modeled as a forced transfer of a part of a population from one class to another. Moreover, this transfer can be assumed instantaneous, when compared to the time scale of an epidemic. Hence, our results on optimal impulsive control of delay systems can be used to determine strategies for controlling the epidemic spread of a disease. An example is shown in Figure 38, where an optimal pulse vaccination strategy is found for the malaria model in Figure 37. Here, we assume that the control consists of a preventive vaccine applied to a portion of the susceptible class of humans. The cost function to

minimize is

$$J(u,T) = \frac{cu^2}{T} + \int_{t_0}^{t_f} I^1(t)dt.$$

The integral term measures the burden of disease (total time spent sick in the population) during the epidemic, and the quadratic control cost reflects the added logistical burden for mass production of vaccines under time pressure. The plots on Figure 38 show the state and costate trajectories of the optimal solution, for one impulse. The costates become of interest when we relate them to "sensitivities" [8].

Figure 38 shows a graphical user interface where the user first selects the parameter values and a initial guess for the control vectors T and U (their length, i.e., the total number of impulses, is free). By clicking the "run" button, the performance index J, its partial derivatives, the state trajectory, and the costate trajectory for this particular control vector are computed (and plotted). Then, the user can improve the performance index by selecting a new control vector in the direction of negative gradient. To do so, the user must specify two stepsizes γ_T and γ_U, then click the "update control" button. The new performance index, partial derivatives, and trajectories are computed and plotted after the "run" button is pressed again. By repeating the process ("update control"/"run"), the user can eventually make the performance index converge to a local minimum.

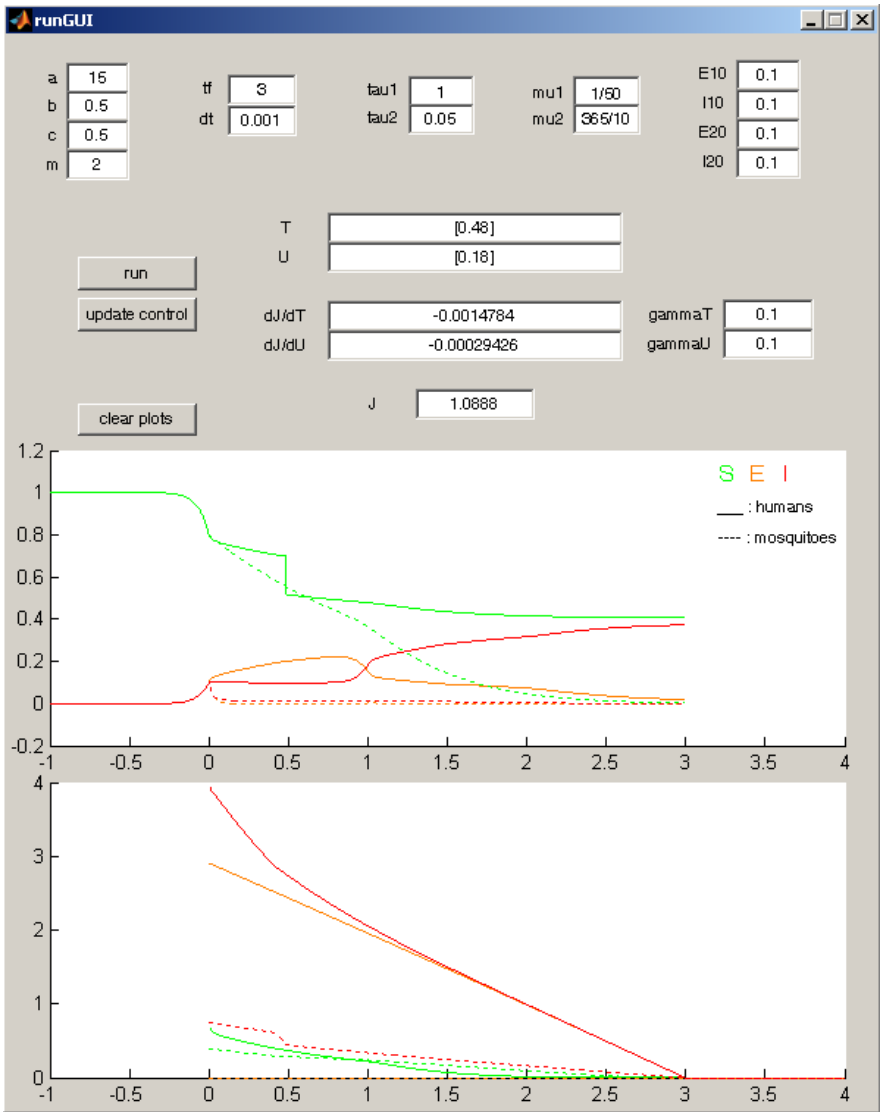


Figure 38: Optimal Impulsive Strategies for the Control of Malaria.

Application 2 - Hybrid Function Approximation:

Here, the impulsive control of switched autonomous systems is used to optimally approximate a given continuous curve. The main application for this method is data compression, where the original trajectory is represented through a mode sequence, switching times, and jump heights.

In a similar way to what was done in Section 8.1, the problem formulation of Equations (32),(33), and (34) is adjusted to accommodate a tracking application [27]. First, the integrand $L(x(t), \tilde{x}(t))$ in the performance index is now a function of both the state x of the approximating system and the state $\tilde{x}$ of the system to approximate. This change allows for a cost criterion of the form $\int_{t_0}^{t_f} ||x(t) - \tilde{x}(t)||^2 dt$. Also, the jumps are now modeled by $x(T_i^+) = x(T_i-) + G_i(x(T_i^-), \tilde{x}(T_i), u_i, T_i)$. Of particular interest, here, is the dependency on the value $\tilde{x}(T_i)$ of the original trajectory at the moment of the switch. For example, by setting $G_i = \tilde{x}(T_i) - x(T_i^-)$, we can consider problems where the tracking error is reset to zero at every mode switch.

Figure 39 illustrates an example where a continuous function is optimally approximated by piecewise constant curves using nine segments. The top picture shows the optimal solution when the value of the switched system is reset to the value of the continuous curve at each mode switch (i.e. $G_i = \tilde{x}(T_i) - x(T_i^-)$). The bottom picture shows the optimal solution when the switched system is free to jump to any value at each mode switch (i.e. $G_i = u_i$).

Figure 40 shows a more complex setting where a continuous curve is approximated by a piecewise linear system with resets. On each segment, the dynamics of the autonomous hybrid system are computed using a Taylor expansion of the target function at the switching time. This result in $\dot{x}(t) = A(T_i)x(t) + B(T_i)$, $t \in (\tau_i, \tau_{i+1})$, where $A(T_i)$ and $B(T_i)$ involve the zero, first, and second derivatives of $\tilde{x}$ at the reset time T_i. Note here that it is assumed that the original and the approximating curve both start from the same value after resets. This, however, can easily be remedied by incorporating the rest values as free parameters in the optimization problem.

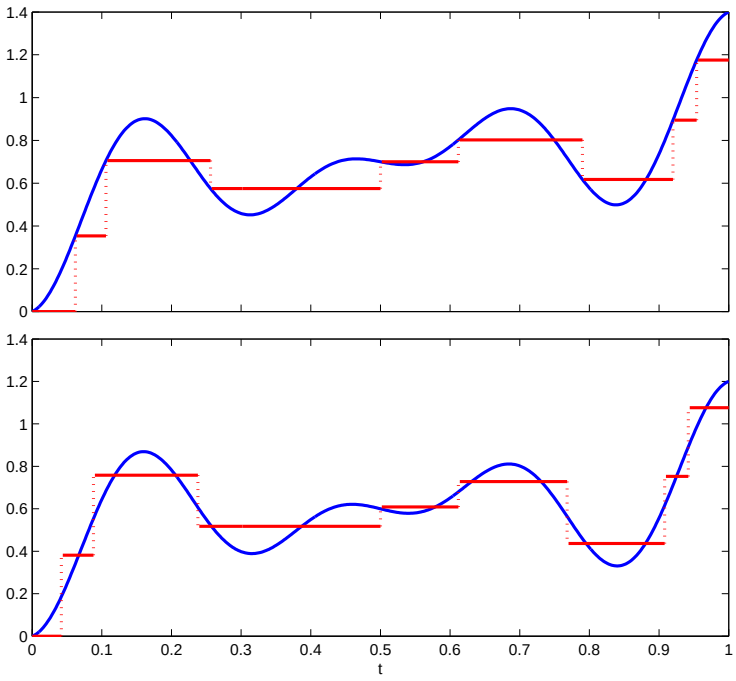


Figure 39: Optimal Piecewise Constant Approximation with Resets or Free Jumps.

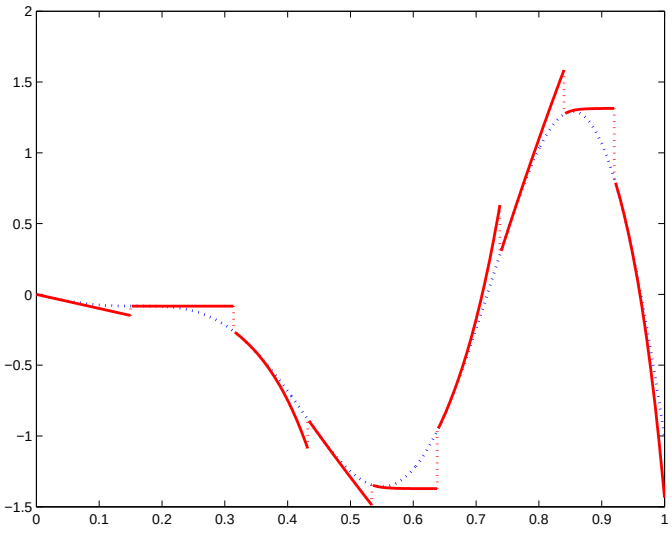


Figure 40: Optimal Approximation Using Piecewise Linear Systems.

CHAPTER IX

EXTENSIONS AND CONCLUSIONS

9.1 Optimal Number of Switches

All the optimal control problems solved in Chapters 7 and 8 were done for a fixed number of modes N. However, since the partial derivatives of J are all computed from the same state and costate trajectories, the computation of the gradient does not suffer much from the number of time switches. In fact, most of the computation time is spent computing the state and costate trajectories only. Consequently, the fast convergence of the method, regardless of N, allows us to solve each problem for an increasing number of switches. Hence, the problem of minimizing J with respect to the time switches can be extended to the number of switches as well. In this perspective, an extra term penalizing the number of switches should be added to J if one wants to avoid cases where the optimal solution would involve an infinite number of switches (the so-called *Zeno-phenomenon*, see [54]).

As an illustration, we solved a similar problem to the one in Chapter 7 for an increasing number of mode switches. To reflect the extra burden of making the controller switch attention from one system to the other, we added a linear cost term to the original cost expression $J = \int_0^1 ||x(t)||^2 dt$. Figure 41 shows the evolution of the optimal cost as N increases, with and without the penalizing term. The decrease of the original cost (blue curve) suggests that without a penalizing cost on the number of modes, the optimal number of modes is infinite. With the extra-term (red curve), the optimal cost reaches a minimum for an optimal number of four mode switches.

9.2 Optimal Sequencing

In this chapter, optimal switching times were solved for a given mode sequence. More generally, the *optimal sequencing* problem consists of finding an optimal sequence of modes, all drawn from a finite set of modes, minimizing a given performance index. Assuming a

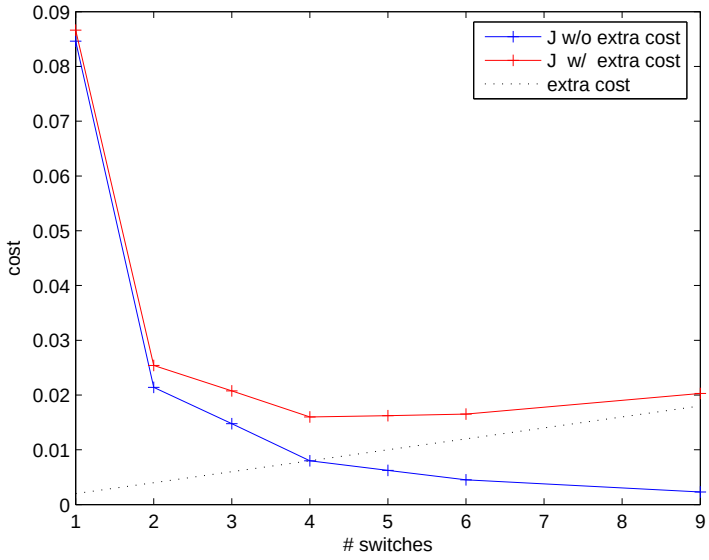


Figure 41: Optimal Cost as a Function of the Number of Switches.

mode alphabet of size M, the number of possible mode sequences with a given length N is M^N. When N is not restricted, the number of possibilities quickly becomes astronomic, and prevents the use of a naive approach computing all possibilities. Also, the discrete nature of the problem does not allow for continuous-parameter optimization techniques. However, the gradient formula derived in Theorem 7.2.1 presents a special structure that enables the application of gradient-descent algorithms to the problem of optimal sequencing. In fact, once the costate trajectory has been computed, it can be used for all of the partial derivatives of J. This observation leads us in [38] to a formula for the directional derivative of J, in the direction defined by inserting new modes into the schedule. Specifically, let $\dot{x}(t) = f_i(x(t))$ in an open interval containing a point τ, and consider inserting a modal function f_j $(j \neq i)$ in an interval of length δ centered at τ. Viewing the cost functional J as a function of δ, the directional derivative is given by the following expression

$$\frac{dJ(0)}{d\delta^+} = \lambda(\tau)^T \Big(f_j(x(\tau)) - f_i(x(\tau)) \Big).$$

Note that a gradient-descent algorithm can exploit the above formula by finding a point τ such that $\lambda(\tau)^T \Big(f_j(x(\tau)) - f_i(x(\tau)) \Big) < 0$, and then pursuing a descent in the direction of injecting the switching function f_j in the interval $[\tau - \delta/2, \tau + \delta/2]$. The same costate $\lambda(t)$ is

used for every possible mode insertion at any time $t \in [0, T]$, and hence the costate trajectory needs to be computed only once for the purpose of determining an adequate insertion schedule. By alternating between mode insertions and optimal switching algorithms, we can then converge to a local solution of the optimal sequencing problem.

Another approach to the optimal sequencing problem is suggested in [82]. Instead of slowly increasing the number of modes by insertions, the idea is to start with an over-estimated number of modes. Suppose the available modes are labelled 1 through M, a mode sequence is constructed by cycling them many (k) times: $1 \to 2 \to ... \to M \to 1 \to 2 \to ... \to M \to ... \to M$. When optimizing the switching times between these modes, Algorithm 7.3.1 will locally filter out unnecessary modes by making consecutive switching times converge. In other words, when the optimal switching time sequence has $T_i = T_{i+i}$, then the i^{th} mode of the mode sequence can be excised. Performing all such excisions should lead to a local solution of the optimal sequencing problem for modes sequences of length at most kM.

The implementation of these two methods, as well as their comparative analysis, is an open problem.

9.3 Expressiveness

Given that the underlying objective of the multi-modal control design is to generate as rich a set of trajectories as possible, one could tie the issue of *expressiveness* directly to the optimal timing control problem. Suppose that the behavior one wants the system to exhibit can be characterized by $\tilde{x}(t),\ t \in [0, T]$, where

$$\dot{\tilde{x}} = f(\tilde{x}, \tilde{u}),$$

for a given control signal $\tilde{u} \in \mathbb{R}^m$. Then, given a multi-modal control string $\sigma = (k_1, \Delta t_1), \ldots, (k_N, \Delta t_N)$ with $k_i : \mathbb{R}^p \to \mathbb{R}^m$ and $\Delta t_i \geq 0,\ \Delta t_1 + \cdots + \Delta t_N \leq T$, one can define the tracking error as

$$J_{track}(\sigma, \tilde{u}) = \int_0^T \|\tilde{x}(t) - x(t)\|^2 dt,$$

where

$$\dot{x} = f(x, k_i(h(x))) = f_i(x),\ t \in [\Delta t_1 + \ldots + \Delta t_{i-1}, \Delta t_1 + \ldots + \Delta t_i).$$

Now, the problem of minimizing J_{track} with respect to σ can be (locally) solved through a combination of the previously discussed scheduling and optimal timing control algorithms. Note that for this to be a well-posed problem, upper bounds on the length of the mode strings must be imposed in order to avoid infinitely long strings. In this setting, we could investigate the *expressiveness* ($\mathcal{E}$) of the mode set by solving the differential game, max-min problem

$$\mathcal{E} = \max_{\tilde{u} \in \mathcal{U}} \{ \min_{\sigma \in (\mathcal{K} \times \mathcal{T})^r} \{ J_{track}(\sigma, \tilde{u}) \} \},$$

where $\mathcal{U}$ is the set of admissible inputs and $(\mathcal{K} \times \mathcal{T})^r$ is the set of all mode strings of length $\leq r$.

With this definition, $\mathcal{E}$ represents the ability of the multi-modal control to track the most aggressive trajectories the system can produce. Two parameters play an important role here: the number of available modes (i.e. the motion alphabet size) and the maximum length r of the available mode strings. It is easy to show that the expressiveness $\mathcal{E}$ defined above is a decreasing function for both parameters. This again, translates to the existence of a tradeoff between complexity and expressiveness.

CHAPTER X

CONCLUSIONS

To summarize, the contributions in this thesis fall under the general banner of multi-modal control systems and they basically fall into two different categories, namely *mode recovery* and *optimal control*. And, although these two topics may seem disparate, they are in fact connected. In order to figure out the optimal sequencing and timing of different modes (optimal control) one first needs to establish a suitable mode set to which the modes belong. One way in which such a set can be constructed is by looking at examples of multi-agent control. From such examples (e.g., human operators controlling a mobile robot), an appropriate mode set can thus be constructed (mode recovery), and in the following paragraphs, the individual contributions made in this thesis in the areas of optimal control and mode recovery are briefly discussed.

For the mode recovery problem, a novel set of algorithms and tools are designed that, given a string of input and output data, reconstructs control programs (strings of feedback-interrupt pairs) that are consistent with the example data. And, since the number of such consistent control programs is potentially quite large, we focused our efforts on the problem of producing low-complexity programs. These programs were moreover transformed into a hybrid automata setting, from which executable, hybrid control laws were the outcome. In fact, these developments culminated in MODEbox: a toolbox for automatically going from example data to executable hybrid automata.

The work concerning optimal control of multi-modal systems mainly focused on the problem of optimal timing control of switched systems. A number of generalizations to the original switched-system formulation were also given. These generalizations included impulse control, control of delayed systems, and hybrid function approximation.

REFERENCES

[1] G.A. Ackerson and K.S. Fu. Estimation in switching environments. *IEEE Transactions on Automatic Control*, 15(1):10–17, Feb. 1970.

[2] Roy M. Anderson and Robert M. May. *Infectious Diseases of Humans - Dynamics and Control*. Oxford University Press, 1991.

[3] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[4] L. Armijo. Minimization of functions having lipschitz continuous first-partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[5] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/nongaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2002.

[6] K.J. Astrom and B.M Bernhardsson. Comparison of Riemann and Lebesgue sampling for first order stochastic systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 2011 – 2016, Las Vegas, Nevada, USA, Dec. 2002.

[7] A. Austin and M. Egerstedt. Mode reconstruction for source coding and multi-modal control. In Oded Maler and Amir Pnueli, editors, *Proceedings of Hybrid Systems: Computation and Control, 6th International Workshop*, volume 2623 of *Lecture Notes in Computer Science*, pages 36–49, Prague, Czech Republic, 2003. Springer.

[8] H. R. Babad and R. B. Vinter. Sensitivity interpretations of the costate function of optimal control. *IMA J Math Control Info*, 10:21–31, 1993.

[9] A. Bemporad, A. Garulli, S. Paoletti, and A.Vicino. A greedy approach to identification of piecewise affine models. In O. Maler and A. Pnueli, editors, *Hybrid Systems:*

Computation and Control, number 2623 in Lecture Notes in Computer Science, pages 97–112. Springer-Verlag, 2003.

[10] A. Bemporad, A. Garulli, S. Paoletti, and A.Vicino. Data classification and parameter estimation for the identification of piecewise affine models. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 20–25, 2004.

[11] A. Bemporad, A. Giua, and C. Seatzu. An iterative algorithm for the optimal control of continuous-time switched linear systems. In *Proceedings of the Sixth International Workshop on Discrete Event Systems*, 2002.

[12] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[13] K. Bennett and O. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3:27–39, 1993.

[14] R.G. Bennetts, J. L. Washington, and D. W. Lewin. A computer algorithm for state table reduction. *Radio and Electronic Engineer*, 42(4):513–520, 1972.

[15] M. Boccadoro, Y. Wardi, M. Egerstedt, and E. Verriest. Optimal control of switching surfaces in hybrid dynamical systems. *Journal of Discrete Event Dynamic Systems*, 15(4):433–448, Dec. 2005.

[16] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[17] R.W. Brockett. On the computer control of movement. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 534–540, New York, April 1988.

[18] R.W. Brockett. Hybrid models for motion control systems. In H. Trentelman and J. Willems, editors, *Essays in control: perspectives in the theory and its applications*, pages 29–54, Boston, 1993. Birkh.

[19] R.W. Brockett. Stabilization of motor networks. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 1484–1488, Dec. 1995.

[20] J. Chudoung and C. Beck. The minimum principle for deterministic impulsive control systems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 4, pages 3569 – 3574, Orlando, FL, Dec. 2001.

[21] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.

[22] D.J. Daley and J.M Gani. *Epidemic Modelling: an Introduction*. Cambridge University Press, 1999.

[23] F. Delmotte and M. Egerstedt. Reconstruction of low-complexity control programs from data. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, volume 2, pages 1460 – 1465, Atlantis, Bahamas, Dec. 2004.

[24] F. Delmotte and M. Egerstedt. From empirical data to multi-modal control procedures. In Anders; Zhou Yishao Dayawansa, Wijesuriya P.; Lindquist, editor, *New Directions and Applications in Control Theory*, volume 321 of *Lecture Notes in Control and Information Sciences*, page 81, Lubbock, TX, 2005. Springer-Verlag.

[25] F. Delmotte, M. Egerstedt, and A. Austin. Data-driven generation of low-complexity control programs. *International Journal of Hybrid Systems*, 4(1&2):53–72, March&June 2004.

[26] F. Delmotte, M. Egerstedt, and C. Martin. Optimal sample time selections for interpolation and smoothing. In *IFAC World Congress*, Prague, Czech Republic, July 2005.

[27] F. Delmotte, M. Egerstedt, and E.I. Verriest. Hybrid function approximation : a variational approach. In *Proceedings of the 44th IEEE Conference on Decision and Control, and 2005 European Control Conference*, pages 374 – 379, Seville, Spain, 2005.

[28] F. Delmotte, T.R. Mehta, and M. Egerstedt. MODEbox: A software tool for obtaining hybrid control and robot navigation strategies from data. *to appear in IEEE Robotics and Automation Magazine*, 2006.

[29] F. Delmotte, E.I. Verriest, and M. Egerstedt. Optimal impulsive control of multi-delay systems. Submitted to Control, Optimisation and Calculus of Variations in July 2006.

[30] M. Egerstedt. Linguistic control of mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 877 – 882, Maui, Hawaii, Oct. 2001.

[31] M. Egerstedt. Motion description languages for multi-modal control in robotics. In H. Cristensen A. Bicchi and D. Prattichizzo Eds., editors, *Control Problems in Robotics*, Springer Tracts in Advanced Robotics, pages 75–90, Las Vegas, NV, Dec. 2002. Springer-Verlag.

[32] M. Egerstedt. On the specification complexity of linguistic control procedures. *International Journal of Hybrid Systems*, 2(1 & 2):129–140, March & June 2002.

[33] M. Egerstedt. Some complexity aspects of the control of mobile robots. In *Proceedings of the 2002 American Control Conference*, volume 1, pages 710 – 715, Anchorage, Alaska, May 2002.

[34] M. Egerstedt. Some complexity aspects of the control of mobile robots. In *American Control Conference*, Anchorage, Alaska, May 2002.

[35] M. Egerstedt, T. Balch, F. Dellaert, F. Delmotte, and Z. Khan. What are the ants doing? vision-based tracking and reconstruction of control programs. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4182 – 4187, Barcelona, Spain, April 2005.

[36] M. Egerstedt and R.W. Brockett. Feedback can reduce the specification complexity of motor programs. *IEEE Transactions on Automatic Control*, 48(2):213–223, Feb. 2003.

[37] M. Egerstedt and Y. Wardi. Multi-process control using queuing theory. In *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[38] M. Egerstedt, Y. Wardi, and F. Delmotte. Optimal control of switching times in switched dynamical systems. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 2138 – 2143, Maui, Hawaii, Dec. 2003.

[39] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine and hybrid systems. *Automatica*, 39:205–217, 2003.

[40] G. Ferrari-Trecate and M. Schinkel. Conditions of optimal classification for piecewise affine regression. In *Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control*, volume 2623, pages 188–202. Springer-Verlag, 2003.

[41] A. Giua, C. Seatzu, and C. Van der Mee. Optimal control of switched autonomous linear systems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 3, pages 2472 – 2477, 2001.

[42] A. Giua, C. Seatzu, and C. Van Der Mee. Optimal control of autonomous linear systems switched with a pre-assigned finite sequence. In *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, pages 144 – 149, 2001.

[43] Y. Go, Y. Xiaolei, and A. Bowling. Navigability of multi-legged robots. *Transactions on Mechatronics, IEEE/ASME*, 11(1):1–8, 2006.

[44] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IRE Trans. Electronic Computers*, EC-14:350359, 1965.

[45] S. Hedlund and A. Rantzer. Optimal control of hybrid systems. In *Proceedings of the 38th IEEE Conference on Decision and Control*, pages 3972–3977, 1999.

[46] H.W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42:599–653, 2000.

[47] H. Higuchi and Y. Matsunaga. A fast state reduction algorithm for incompletely specified finite state machines. In *Proceedings of the 1996 Design Automation Conference*, pages 463–466, 1996.

[48] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 2nd edition, 2001.

[49] J.E. Hopcroft and G. Wilfong. Motion of objects in contact. *International Journal of Robotics Research*, 4:32–46, 1986.

[50] D. Hristu and K. Morgansen. Limited communication control. *Systems Control Letters*, 37(4):193–205, 1999.

[51] D. Hristu-Varsakelis. Feedback control systems as users of shared network : Communication sequences that guarantee stability. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 4, page 36313631, 2001.

[52] D. Hristu-Varsakelis and S. Andersson. Directed graphs and motion description languages for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2689 – 2694, Washington, DC, May 2002.

[53] D. Hristu-Varsakelis, P.S. Krishnaprasad, S. Andersson, F. Zhang, P. Sodre, and L. D'Anna. The mdle engine a software tool for hybrid motion control. Technical Report ISR Technical Report TR2000-54, University of Maryland, 2000.

[54] K.H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry. On the regularization of zeno hybrid automata. *Systems and Control Letters*, 38:141–150, 1999.

[55] A. Juloski, S. Weiland, and W. Heemels. A bayesian approach to identification of hybrid systems. In *Proceedings of the 43rd Conference on Decision and Control*, pages 13–19, Paradise Island, Bahamas, 2004.

[56] A.Lj. Juloski, W.P.M.H Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and J.H.G. Niessen. Comparison of four procedures for the identification of hybrid systems. In L. Thiele M. Morari, editor, *Proceedings of the 8th International Workshop, Hybrid*

Systems: Computation and Control, Lecture Notes in Computer Science, pages 354–369. Springer-Verlag, March 2005.

[57] H.H. Kagiwada. *System identification: methods and applications*, volume 4 of *Applied mathematics and computation*. Reading, Massachussets, 1974.

[58] H. Kano, M. Egerstedt, H. Nakata, and C.F. Martin. B-splines and control theory. *Applied Mathematics and Computation*, 145(2-3):265–288, 1993.

[59] R. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, 1972.

[60] H. Kobayashi, K. Kikuchi, K. Ochi, and Y. Onogi. Navigation strategies referring to insect homing in flying robots. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, volume 2, pages 1695 – 1700, 2001.

[61] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 2nd edition, 1997.

[62] B. Lincoln and A. Rantzer. Optimizing linear system switching. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 3, pages 2063 – 2068, 2001.

[63] L. Ljung. *System Identification : Theory for the User*. Englewood Cliffs, 1987.

[64] S.P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[65] V. Manikonda, J. Handler, and P.S. Krishnaprasad. Formalizing behavior-based planning for nonholomic robots. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, volume 1, pages 142–149, August 1995.

[66] V. Manikonda, P.S. Krishnaprasad, and J. Handler. A motion description language and a hybrid architecture for motion planning with nonholomic robots. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, volume 2, pages 2021–2028, May 1995.

[67] V. Manikonda, P.S. Krishnaprasad, and J. Handler. Languages, behaviors, hybrid architectures and motion control. In John Baillieul and Jan C. Willems, editors, *Mathematical Control Theory*, pages 199–226. Springer, 1998.

[68] M. Moonen, B. De Moor, L. Vandenberghe, and J.Vandewalle. On- and off-line identification of linear state-space models. *International Journal of Control*, 49(1):219–232, 1989.

[69] M. Paull and S. Unger. Minimizing the number of states in incompletely specified state machines. *IRE Trans. Electronic Computers*, EC-8:356367, 1959.

[70] J. M. Pena and A. L. Oliveira. A new algorithm for exact reduction of incompletely specified finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18:1619–1632, 1999.

[71] C.F. Pfleeger. State reduction in incompletely specified finite state machines. *IEEE Transactions on Computers*, C-22:1099–1102, 1973.

[72] H. Rehbinder and M. Sanfirdson. Scheduling of a limited communication channel for optimal control. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 1011 – 1016, 2000.

[73] J.-K. Rho, G.D. Hachtel, F. Somenzi, and R.M. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):167–177, 1994.

[74] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15 of *Series in Computer Science*. World Scientific, River Edge, NJ, 1989.

[75] J. Roll, A. Bemporad, and L. Ljung. Identification of piecewise affine systems via mixed-integer programming. *Automatica*, 40(1):37–50, Jan. 2004.

[76] M.S. Shaikh and P. Caines. On trajectory optimization for hybrid systems: Theory and algorithms for fixed schedules. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 1997 – 1998, Las Vegas, NV, Dec. 2002.

[77] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[78] S. Sun, M. Egerstedt, and C.F. Martin. Control theoretic smoothing splines. *IEEE Transactions on Automatic Control*, 45:2271–2279, 2000.

[79] H.J. Sussmann. A maximum principle for hybrid optimal control problems. In *Proceedings of the 38th IEEE Conference on Decision and Control*, volume 1, pages 425 – 430, Dec. 1999.

[80] D.D. Sworder and J.E. Boyd. *Estimation Problems in Hybrid Systems*. Cambridge Univ Press, 1999.

[81] C. Tomlin and M.R. Greenstreet, editors. *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of *Lecture Notes in Computer Science*. Springer, 2002.

[82] E.I. Verriest and F. Delmotte. Optimal control for switched point delay systems with refractory period. In *IFAC World Congress*, Prague, Czech Republic, July 2005.

[83] E.I. Verriest, F. Delmotte, and M. Egerstedt. Optimal impulsive control for point delay systems with refractory period. In *IFAC Workshop on Time-Delay Systems*, Leuven, Belgium, Sept. 2004.

[84] E.I. Verriest, F. Delmotte, and M. Egerstedt. Control of epidemics by vaccination. In *Proceedings of the 2005 American Control Conference*, volume 2, pages 985–990, June 2005.

[85] R. Vidal. Identification of pwarx hybrid models with unknown and possibly different orders. In *Proceedings of IEEE American Control Conference*, volume 1, pages 547 – 552, 2004.

[86] R. Vidal, S. Soatto, Y. Ma, and S. Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. In *Proceedings of IEEE Conference on Decision and Control*, 2003.

[87] G. Walsh, H. Ye, and L. Bushnell. Stability analysis of networked control systems. In *Proceedings of American Control Conference*, page 28762880, 1999.

[88] L.Y. Wang, A. Beydoun, J.Cook, J. Sun, and I. Kolmanovsky. Optimal hybrid control with applications to automotive powertrain systems. In A.S. Morse, editor, *Proceedings of the Block Island Workshop on Control using Logic Based Switching*, Lecture Notes in Control and Information Science. Springer Verlag, 1996.

[89] X. Xu and P. Antsaklis. Optimal control of switched autonomous systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 4401–4406, Las Vegas, NV, Dec. 2002.

[90] X. Xu and P.J. Antsaklis. An approach for solving general switched linear quadratic optimal control problems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 3, pages 2478–2483, Dec. 2001.

[91] X. Xu and P.J. Antsaklis. An approach to switched systems optimal control based on parameterization of the switching instants. In *Proceedings of the 15th IFAC World Congress*, 2002.

[92] L.A. Zadeh. From circuit theory to system theory. In *Procceedings IRE*, volume 50, pages 856–865, 1962.