Learning and Validation of Human Control Strategies

Michael C. Nechyba

CMU-RI-TR-98-06

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

May 1998

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This research was funded in part through a National Science Foundation Graduate Research Fellowship, as well as a Department of Energy Doctoral Research Fellowship.

Abstract

In this thesis, we apply machine learning techniques and statistical analysis towards the learning and validation of human control strategy (HCS) models. This work has potential impact in a number of applications ranging from space telerobotics and real-time training to the Intelligent Vehicle Highway System (IVHS) and human-machine interfacing. We specifically focus on the following two important questions: (1) how to efficiently and effectively model human control strategies from real-time human control data and (2) how to validate the performance of the learned HCS models in the feedback control loop. To these ends, we propose two discrete-time modeling frameworks, one for continuous and another for discontinuous human control strategies. For the continuous case, we propose and develop an efficient neural-network learning architecture that combines flexible cascade neural networks with extended Kalman filtering. This learning architecture demonstrates convergence to better local minima in many fewer epochs than alternative, competing neural network learning regimes. For the discontinuous case, we propose and develop a statistical framework that models control actions by individual statistical models. A stochastic selection criterion, based on the posterior probabilities for each model, then selects a particular control action at each time step.

Next, we propose and develop a stochastic similarity measure — based on Hidden Markov Model (HMM) analysis — that compares dynamic, stochastic control trajectories. We derive important properties for this similarity measure, and then, by quantifying the similarity between model-generated control trajectories and corresponding human control data, apply this measure towards validating the learned models of human control strategy. The degree of similarity (or dissimilarity) between a model and its training data indicates how appropriate a specific modeling approach is within a specific context. Throughout, the learning and validation methods proposed herein are tested on human control data, collected through a dynamic, graphic driving simulator that we have developed for this purpose. In addition, we analyze actual driving data collected through the Navlab project at Carnegie Mellon University.

Acknowledgments

First, I sincerely thank my advisor Yangsheng Xu for his friendship and intellectual guidance throughout the years; I cannot imagine a better role model for myself on how to conduct and coordinate the many duties of a professor. I also thank Yangsheng for giving me the wonderful and unique opportunity to study in Hong Kong. I thank the other members of my thesis committee, Dean Pomerleau and Andrew Moore for their valuable and constructive suggestions throughout. Special thanks to Shumeet Baluja, whose many insightful comments have all been offered in his recent and short time on my committee.

Thanks to the many, many people who gave of their time to "drive" through my driving simulator. Without their data, it would have been impossible to develop and test the learning methods and analysis tools developed in this thesis. Special thanks to "Larry," "Curly," "Moe," "Groucho," "Harpo," and "Zeppo," who are acknowledged anonymously below.

I thank Andrew Moore and Jeff Schneider for the use of their memory-based learning and principal component analysis software. I thank Scott Fahlman for his advice in using the cascade learning architecture, and Michael Kingsley and David C. Lambert for their cascade training software. I thank Doug Baker for sharing his data and learning results for the cascade learning architecture. I also thank Parag Batavia and the Navlab group for sharing driving data collected on Pennsylvania roads.

I thank Ofer Barkai for our friendship and our many discussions on Hidden Markov Models. Likewise I thank Kan Deng for our frequent and interesting discussions on time series analysis. I thank Jie Yang, whose work on human modeling inspired some of mine. Thanks also go to Ava Cruse, Marie Elm, Carolyn Ludwig, and Marce Zaragoza for their excellent support over the years.

In addition, I thank the following people for their friendship, both personal and professional, throughout my time at CMU: Marcel Bergerman, Fabio Cozman, Andrew Gove, John Hancock, Lalit Katragadda, Chris Lee, John Murphy, Mark Ollis, Henry Schneiderman and Harry Schum. Special thanks to Geoff and Paula Gordon for housing me during my thesis writing months in Pittsburgh. Also, thanks to Jingyan Song and Winston Sun for their help and friendship during my time in Hong Kong.

Finally, with great love and respect, I acknowledge the support, love and encouragement of my family — my parents, my grandmother (“Oma”), Andy, Tom, Christian, and more recently, Raquel, Stacy and Lyana — without whom I surely would not have reached this stage in my life. Special thanks to Tom whose decision to pursue a Ph.D. encouraged my decision to do the same.

Contents

Abstract **iii**

Acknowledgments **v**

1 Introduction **1**

1.1 Motivation 1

1.2 Related work 4

1.2.1 Skill learning through exploration 5

1.2.2 Skill modeling from human data 5

1.2.3 Neural network learning 8

1.2.4 Locally weighted learning 10

1.3 Overview 11

2 Experimental design **15**

2.1 Motivation 15

2.2 Simulation environment 17

2.2.1 Dynamic driving simulator 17

2.2.2 Road descriptions 19

2.3 Model class 21

3 Cascade Neural Networks with Kalman Filtering 25

3.1 Motivation 26

3.2 Cascade neural networks 28

3.3 Node-decoupled extended Kalman filtering 30

3.3.1 Learning architecture 30

3.3.2 Computational complexity 33

3.4 Comparison experiments 36

3.4.1 Problem descriptions 36

3.4.2 Learning results 39

3.4.3 Noisy learning results 43

3.4.4 Discussion 44

4 HCS Models: Continuous Learning 47

4.1 Cascade with quickprop learning 48

4.1.1 Experimental data 48

4.1.2 Model inputs and outputs 48

4.1.3 *Cq* training 49

4.1.4 HCS models 50

4.2 Cascade with NDEKF learning 50

4.2.1 Model inputs and outputs 50

4.2.2 *Ck* training 55

4.2.3 HCS models 56

4.3 Analysis 56

4.3.1 Model stability 56

4.3.2 Learning convergence 64

4.3.3 Discussion 65

5 HCS Models: Discontinuous Learning 69

5.1 Hybrid continuous/discontinuous control 70

5.1.1 General statistical framework 70

5.1.2 Action definitions 72

5.1.3 Statistical model choice 73

5.1.4 Prior probabilities 75

5.1.5 Task-based modifications 76

5.2 Experimental results 78

5.2.1 Model training 78

5.2.2 HCS models 79

5.3 Analysis 84

5.3.1 Sample curve control 84

5.3.2 Probability profile 87

5.3.3 Modeling extension 90

6 Model validation 93

6.1 Need for model validation 94

6.2 Stochastic similarity measure 97

6.2.1 Hidden Markov Models 98

6.2.2 Similarity measure 100

6.2.3 Properties 101

6.2.4 Distance measure 104

6.2.5 Data preprocessing 106

6.2.6 Vector quantization 113

6.2.7 Discretization compensation 114

6.2.8 HMM training 119

7 Human-to-human similarity 123

7.1 Comparing human control strategies 123

7.1.1 Experimental data 123

7.1.2 Classification experiments 124

7.1.3 Bayes classification 126

7.1.4 Spectral classification 132

7.1.5 Task-based classification 134

7.1.6 Classification with performance drift 135

7.2 Comparing Navlab driving data 137

7.2.1 Experimental data 137

7.2.2 Classification experiments 138

7.3 Analysis 139

7.3.1 One-sided similarity measure 139

7.3.2 Bayes classifier limitations 141

7.3.3 Similarity measure variations 143

8 Human-to-model validation 149

8.1 Human-to-model comparisons 149

8.2 Classification experiment 151

8.3 Model inputs revisited 152

9 Conclusion 155

9.1 Contributions 155

9.2 Future work 156

Appendix A Human control data **161**

A.1 Larry 163

A.2 Curly 165

A.3 Moe 167

A.4 Groucho 169

A.5 Harpo 171

A.6 Zeppo 173

Appendix B HMM Training **175**

B.1 Forward-backward algorithm 175

B.2 Baum-Welch algorithm (with scaling) 177

Appendix C Author's Publications **181**

Bibliography **183**

Chapter 1
Introduction

1.1 Motivation

Over the past two decades, rapid advances in computer performance have not been matched with similar advances in the development of intelligent robots and systems. Although humans are quite adept at mastering complex and dynamic skills, we are far less impressive in formalizing our behavior into algorithmic, machine-codable strategies. Therefore, it has been difficult to duplicate the types of intelligent skills and actions, we witness every day as humans, in robots and other machines. This not only limits the capabilities of individual robots, but also the extent to which humans and robots can safely interact and cooperate with one another. Nevertheless, human actions are currently our only examples of truly “intelligent” behavior. As such, there exists a profound need to abstract human skill into computational models, capable of realistic emulation of dynamic human behavior.

Models of human skill can transfer intelligent control behaviors to robots. This is especially critical for robots which have to operate in remote or inhospitable environments, which humans cannot reach. For example, sending humans to operate in space is often expensive, dangerous, or outright impossible, while sending robots instead is comparatively cheap and involves no risk to human life. Replacing astronauts with robots is only feasible, however, if the robots are equipped with sufficient autonomous skill and intelligence; we suggest that a robot can acquire those necessary capabilities through abstracted models of human skill.

In other robotic applications, we would like robots to carry out tasks which humans have traditionally performed. For example, the Intelligent Vehicle Highway System (IVHS), currently being developed through massive initiatives in the United States, Europe, and Japan [25, 26], envisions automating much of the driving on our highways. The required automated vehicles will need significant intelligence to interact safely with variable road conditions and other traffic. Modeling human intelligence offers one way of building up the necessary skills for this type of intelligent machine.

With increased intelligence and sophistication in robotic systems, analysis of human-robot coordination in tightly coupled human-machine systems will become increasingly relevant. In IVHS, for example, there will be ubiquitous interaction between autonomous vehicles and their human drivers/passengers. Moreover, the currently limited application domain for robots may broaden into other aspects of consumer life, where household and service robots will interact primarily with non-experts. To ensure safe coordination with humans in a shared workspace, we must incorporate appropriate models of human behavior into the world model of the robots. We can assess the quality of joint human-machine systems by including computational models of human behavior in the overall system analysis.

Realistic simulation of human behavior is required not only in human-machine systems, but also in the burgeoning field of virtual reality. As graphic displays become increasingly life-like, the dynamic behavior of the virtual world will need to match the increased visual realism. Computational models of human skill can impart the necessary sense of realism to the actions and behaviors of virtual humans in the virtual world. Consider, for example, a NASCAR video game. Rather than have preprogrammed behaviors, human driver models, abstracted from different race car drivers, could generate more diverse and human-like driving behaviors in the simulated competitors.

Finally, accurate models of human skill can contribute to improved expert training and human-computer interfacing (HCI). Consider, for example, the tasks of teleoperating robots in remote environments or learning to fly a high-performance jet. Training for both of these tasks is difficult, expensive, and time consuming for a novice [83, 104]. We can accelerate learning for the novice operator by providing on-line feedback from virtual teachers in the form of skill models, which capture the control strategies of expert operators. Through the use of human skill models, operator performance can be monitored during training or actual task execution as information is displayed through different sensor modalities and layouts.

Thus, models of human skill find application in far ranging fields, from autonomous robot control and teleoperation to human-robot coordination and human-robot system simulation. Since scientific understanding of human intelligence is incomplete at best, however, models of human skill cannot be derived analytically. Rather, we have to model human skill through observation, or *learning*, of experimentally provided human training data. Current learning paradigms are not sufficiently rich to model the full range of human skill, from low-level muscle control to high-level reasoning and abstract thought. As such, we restrict our focus to *human control strategy*, a particular subset of human skill discussed below.

Broadly speaking, *human skill* can be grouped into two categories: (1) *action* skills, which are open-loop, and (2) *reaction* skills, which are closed-loop, and require sensory feedback to successfully execute the skill. Tossing or kicking a ball is an example of action skill. Driving a car, on the other hand, is a classic example of reaction skill, where the human closes the feedback control loop. *Human control strategy* we study in this thesis is a subset of this type of reaction skill. In terms of complexity, human control strategy lies between low-level feedback control and high-level reasoning, and encompasses a wide range of useful physical tasks with a reasonably well-defined numeric input/output representation. On the one hand, a control strategy is not only defined by the "gains" or parameters in the controller, but also the structure or approach of the strategy. Consider the skill of driving a car, for example. Figure 1-1 illustrates applied force profiles over the same road for two different individuals in a driving simulator.

The distinction between the two driving styles is a difference in kind rather than merely a difference in degree, similar to the structural difference between a linear feedback and a variable structure controller. Each represents a unique control strategy. On the other hand, the demonstrated skill in Figure 1-1 requires no high-level reasoning or abstract thought. Modeling such mental processes, of which humans are capable, requires an as-of-yet unavailable understanding of the human traits of self-awareness and consciousness.

1.2 Related work

The field of *intelligent control* [122] has emerged from the field of classical control theory to deal with applications too complex for classical control approaches. Broadly speaking, intelligent control combines classical control theory with *adaptation*, *learning*, or *active exploration*. Methods in intelligent control include fuzzy logic control, neural network control, reinforcement learning, and locally weighted learning for control.

Neural networks are used to map unknown nonlinear functions and have been applied in control most commonly for dynamic system identification and parameter adaptive control [8, 73, 79]. Locally weighted learning presents an alternative to neural networks, and maps unknown functions through local statistical regression of the experimental data, rather than through a functional representation [9, 10]. In reinforcement learning, an appropriate control strategy is found

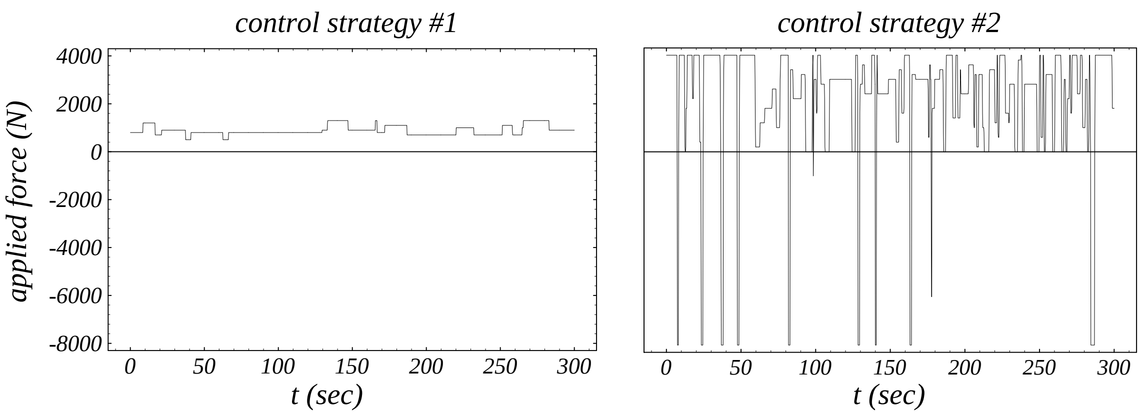


Figure 1-1: Control forces applied by two different individuals in a driving simulator for the same road and simulation parameters.The two control strategies are quite different.

through dynamic programming of a discretized state space [14]. Below, we describe previous work relating to each of these methods.

1.2.1 Skill learning through exploration

Learning skill through exploration has become a popular paradigm for acquiring robotic skills. In *reinforcement learning* [14, 76, 115, 120], data is not given as direct input/output data points; rather data is specified by an input vector and an associated (scalar) reward from the environment. This reward represents the reinforcement signal, and is akin to "learning with a critic" as opposed to "learning with a teacher." The reinforcement learning algorithm is expected to explore and learn a suitable control strategy over time. References [43] and [10] give some examples of reinforcement learning control for a robot manipulator and a simulated car in a hole, respectively. Schneider [102] learns the open-loop skill of throwing through a search of the parameter space which defines all possible throwing motions. One of the advantages of learning human control strategy directly from human control data is that we avoid the need for this type of state space search to find a suitable control strategy.

Lee and Kim [67] have proposed and verified an *inductive learning* scheme, where control rules are learned from examples of perception/action data through hypothesis generation and testing. Their learning paradigm, Expert Assisted Robot Skill Acquisition (EARSA) [68], consists of two steps: (1) skill acquisition from human expert rules, and (2) skill discovery or refinement through hypothesis generation and testing.

1.2.2 Skill modeling from human data

Interest in modeling human control goes all the way back to World War II, when engineers and psychologists attempted to improve the performance of pilots, gunners and bombardiers [48]. Early research in modeling human control is based on the *control-theory paradigm* [72], which attempts to model the human-in-the-loop as a simple feedback control system. These modeling efforts generally focussed on simple tracking tasks, where the human is most often modeled as a simple time delay in the overall human-machine system [105].

More recently, work has been done towards learning more advanced skills directly from humans. In fuzzy control schemes [63, 64], human experts are asked to specify “if-then” control rules, with fuzzy linguistic variables (e.g. “hot,” “cold,” etc.), which they believe guide their control actions. For example, simple human-in-the-loop control models based on fuzzy modeling have been demonstrated for automobile steering [60] and ships helmsmen [116]. Although fuzzy control systems are well suited for simple control tasks with few inputs and outputs, they do not scale well to the high-dimensional input spaces required in modeling human control strategy [8].

Robot learning from human experts has also been applied to a deburring robot. Asada and Yang [6] derive control rules directly from human input/output data, by associating input patterns with corresponding output actions for the deburring robot. In [125], Yang and Asada combine linguistic information and numeric input/output data for the overall control of the robot. Expert linguistic rules are acquired directly from a human expert to partition the control space. For each region, a corresponding linear control law is derived from the numeric demonstration data by the human expert. In [5, 70, 106], the same deburring robot is controlled through an associative neural network which maps process parameter features to action parameters from human control data. The proper tool feed rate is determined from the burr characteristics of the current process.

Lee and Chen [66] use feasible state transition graphs through self-organizing data clusters to abstract skill from human data. Skills are modeled as optimal sequences of one-step state transitions that transform the current state into the goal state. The approach is verified on demonstrated human Cartesian teleoperation skill. Yang, *et. al*. [126, 127] implement a different state-based approach to open-loop skill learning and telerobotics using Hidden Markov Models (HMMs). HMMs are trained to learn both telerobotic trajectories executed by a human operator and simple human handwriting gestures. Yamato, *et. al.* [123, 124] also train HMMs to recognize open-loop human actions.

Friedrich, *et. al.* [36] and Kaiser [56] review programming by demonstration and skill acquisition via human demonstration for elementary robot skills. Lee [65] investigates human-to-robot skill transfer through demonstration of task performance in a virtual environments. Voyles, *et. al.* [119] program a robot manipulator through human demonstration and abstraction of gesture primitives. Delson and West [28] and Ude [118] both learn open-loop robot trajectories from human demonstration. Skubic and Volz [109] transfer force-based assembly skills to robots from human demonstrations. Iba [53] models open-loop sensory motor skills in humans. Gingrich, *et. al.* [39] argue that learning human performance models is valuable, but offer results only for simulated, known dynamic systems.

Several approaches to skill learning in human driving have been implemented. In [34, 35], neural networks are trained to mimic human behavior for a simulated, circular racetrack. The task essentially involves avoiding other computer-generated cars; no dynamics are modeled or considered in the approach. Pomerleau [89, 90] implements real-time road-following with data collected from a human driver. A static feedforward neural network with a single hidden layer, ALVINN, learns to map coarsely digitized camera images of the road ahead to a desired steering direction, whose reliability is given through an input-reconstruction reliability estimator. The system has been demonstrated successfully at speeds up to 70 mi/h. Subsequently, a statistical algorithm called RALPH [88] has been developed for calculating the road curvature and lateral offset from the road median. Neuser, *et. al.* [82] control the steering of an autonomous vehicle through preprocessed inputs to a single-layer feed-forward neural network. These preprocessed inputs include the car's yaw angle with respect to the road, the instantaneous and time-averaged road curvature, and the instantaneous and time-averaged lateral offset. Driving data is again collected from a human operator. In [74], the authors provide a control theoretic model of human driver steering control. Finally, Pentland and Liu [86] apply HMMs towards inferring a particular driver's high-level intentions, such as turning and stopping.

Other, higher level skills have also been abstracted from human performance data. Kang [57], for example, teaches a robot assembly through human demonstration. The system observes a human performing a given task, recognizes the human grasp, and maps it onto an available manipulator. In other words, a sequence of camera images, observing the human demonstration, is automatically partitioned into meaningful temporal segments. Kosuge, *et. al.* [59] also abstract high-level assembly skill from human demonstration data. The high-level sequence of motion is decomposed into discrete state transitions, based on contact states during assembly. In each state, compliant motion control implements the corresponding low level control. Hovland, *et. al.* [50] encode human assembly skill with Hidden Markov Models. In [85], neural networks encode simple pick-and-place skill primitives from human demonstrations.

1.2.3 Neural network learning

Interest and research in neural network-based learning for control has exploded in recent years. References [3, 4, 19, 52, 73, 99] provide good overviews of neural network control over a broad range of applications. Most often, the role of the neural network in control is restricted to modeling either the nonlinear plant or some nonlinear feedback controller.

In choosing a neural network learning architecture, there are several choices to be made, including the (1) type, (2) architecture and (3) training algorithm. Broadly speaking there are two types of neural networks: (1) feedforward and (2) recurrent networks. *Feedforward neural networks* have connections in only one direction (from inputs to outputs). As such, they are static maps, which, in order to model *dynamic systems*, require time-delayed histories of the sensor and previous control variables as input. *Recurrent networks*, on the other hand, permit connections between units in all directions, including self connections, thereby allowing the neural network to implicitly model dynamic characteristics (i.e. discover the state) of the system. Compared to static feedforward networks, the learning algorithms for recurrent networks are significantly more computationally involved, requiring relaxation of sets of differential equations [47]. Yet, Qin, *et. al.* [93] show similar error convergence in mapping simple dynamic systems with feedforward and recurrent networks, respectively. As such, we will restrict the remainder of this discussion to feedforward models only.

Research into feedforward neural networks began in earnest with the publication of the backpropagation algorithm in 1986 [98]. Since then a number of different learning architectures have been proposed to adjust the structure of the feedforward neural network (i.e. the number and arrangement of hidden units) as part of learning. These approaches can be divided into (1) destructive and (2) constructive algorithms. In destructive algorithms, oversized feedforward models are trained first, and then, after learning has been completed, “unimportant” weights, based on some relevancy criteria are pruned from the network. See, for example [18, 22, 23, 44, 77, 78, 117]. In constructive algorithms, on the other hand, neural networks are initialized in some minimal configuration and additional hidden units are added as the learning requires. Ash [7], Bartlett [13] and Hiroshe [49], for example, have all experimented with adaptive architectures where hidden units are added one at a time in a single hidden layer as the error measure fails to reduce appreciably during learning. Fahlman [31, 32] proposes a cascade learning architecture, where hidden units are added in multiple cascading layers as opposed to a single hidden layer. Cascade learning has a comparative advantage over the other adaptive learning architectures in that (1) new hidden nodes are not arranged in a single hidden layer, allowing more complex mappings, and (2) not all weights are trained simultaneously, resulting in faster convergence. With respect to constructive algorithms, destructive algorithms compare unfavorably, since initially, a lot of effort is expended training (by definition) too many weights, and the pruned networks needs to be retrained multiple times, after each individual weight or unit has been pruned.

Finally we note that there is a selection of training algorithms available for feedforward neural networks. As we have already noted, the first of these was the backpropagation algorithm, which implements local gradient descent on the weights during training in order to minimize the sum-of-squared residuals. Since this training method was first proposed [98], modifications to standard backpropagation, as well as other training algorithms have been suggested. An adaptive learning rate [24], as well as an additive momentum term [87] are both somewhat effective in accelerating convergence of backpropagation in “flat” regions of the error hypersurface. Quickprop [31] incorporates local, second-order information in the weight-update algorithm to further speed up learning. Kollias and Anastassious [58] propose applying the Marquardt-Levenberg least squares optimization method, which utilizes an approximation of the Hessian matrix. Extended Kalman filtering [92, 108], where the weights in the neural network are viewed as states in a discrete-time finite dimensional system, outperform the previously mentioned algorithms in terms of learning speed and error convergence [92]. In all cases, experimental data is usually partitioned into two random sets — one for actual training, and the other for cross validation [47]; training is generally stopped once the error measure on the cross-validation set no longer decreases.

1.2.4 Locally weighted learning

Neural networks have received great attention for nonlinear learning and control applications. Another learning paradigm, *locally weighted learning*, has emerged more recently and has shown great success for a number of different control applications, ranging from devil sticking [100] to robot juggling [101]. Atkeson, *et. al.* [9] offer an excellent overview of locally weighted learning, while [10] addresses control-specific issues. *Locally weighted regression* is one instance of locally weighted learning and is similar in approach to CMAC [1] and RBF [75] neural networks in that local (linear) models are fit to nearby data. All the data is explicitly stored and organized in efficient data structures (such as *k*-d trees [17] or Bump trees [84], for example). When the model is queried for the output at a specified input vector, points in the database near that input vector are used to construct a local linear map.

Locally weighted regression offers several advantages over global learning paradigms (such as neural networks) [10]. First, locally weighted regression results in smooth predicted surfaces. Second, it automatically linearizes the system around a query point by providing the local linear map. Third, adding new data to the locally weighted regression model is cheap, as it merely requires inserting a new data point in the existing data base. Learning occurs in one-shot, since all the data is retained for the construction of the local linear models. Fourth, local minima are not a problem, as no gradient descent is required for learning the model. Finally, interference (e.g. the catastrophic forgetting problem) between old and new experiences does not occur.

Despite these appealing features, locally weighted regression also suffers from some shortcomings. First, computational and memory costs increase as the size of the data increases. Second, efficient data structures become less efficient as the dimensionality of the input space grows. Finally, locally weighted learning is very sensitive to *a priori* representational choices.[1]

1.3 Overview

The research in modeling skill from human data or human demonstration has thus far not addressed a number of important issues. Much of the previous work models only open-loop human action skills, static or quasi-static skills, or higher-level abstractions of human skill (e.g. assembly). In general, the work has not focussed on abstracting models of *dynamic* human control strategies, as defined above. The work that has been done in modeling dynamic human-in-the-loop control essentially views the human as a low-level feedback transfer function — nothing more than an annoying time delay with particular noise properties.

Therefore, this thesis applies machine learning techniques and statistical analysis towards abstracting models of human control strategy. It is our contention that such models can be learned efficiently to emulate complex human control behaviors in the feedback loop.

The thesis is organized as follows. In Chapter 2, we describe a graphic dynamic driving simulator which we have developed as an experimental platform for collecting, modeling and analyzing human control data. We argue that such a simulation environment is well suited for our purposes, in large measure because it protects the human subjects from injuring or harming themselves or others during control execution and data collection. We then define the class of models — static feedforward models — to which we restrict ourselves in this thesis.

1. We observe that locally weighted learning (including *k*-nearest neighbor modeling) suffers all these effects for the human control data in this thesis. First, the human control data sets are large, with approximately 30,000 data points per set. Second, the input space typically has dimensionality between 35 and 100. Consequently, short of global models, we have found it difficult to generate stable memory-based models, that run in anything close to real time (except for the most trivial data sets).

Next, in Chapter 3 we propose and develop an efficient continuous learning framework for modeling human control strategy that combines cascade neural networks and node-decoupled extended Kalman filtering. We illustrate that the resulting architecture converges to better local minima in many fewer epochs than alternative feedforward neural network approaches for some known function approximation and dynamic system identification problems.

In Chapter 4 we then apply cascade learning towards abstracting HCS models from experimental control strategy data. We compare two training algorithms introduced in the previous chapter — namely, quickprop and NDEKF, and show that cascade/NDEKF converges orders of magnitude faster than cascade/quickprop for the human control training data. While the cascade models form stable controllers, we observe that they qualitatively fail to match the human training data with a high degree of fidelity. We observe that the dissimilarity between the human and model-generated control trajectories is principally caused by switching discontinuities in one of the model outputs, and that, in fact, *any* continuous modeling framework would suffer equally in attempting to model that output without high-frequency noise.

Therefore, in Chapter 5 we propose and develop a stochastic, discontinuous modeling framework for modeling discontinuous human control strategies. This approach models different control actions as individual statistical models, which, together with the prior probabilities of each control action, combine to generate a posterior probability for each action, given the current model inputs. A control decision is then made stochastically, based on the posterior probabilities. We apply the discontinuous modeling framework towards modeling human control strategies and observe that the resulting model-generated trajectories appear to be more similar to the human training data.

In Chapter 6 we then set out to quantify the qualitative observations of model fidelity in Chapters 4 and 5. As a first step in validating the learned models of human control strategy, we propose and develop a stochastic similarity measure, based on Hidden Markov Model analysis that is capable of comparing multi-dimensional stochastic control trajectories. The goal of this similarity measure is to compare model-generated control trajectories with their respective human control trajectories.

Chapter 7 verifies the similarity measure by comparing human control data across different individuals. By comparison, an alternative statistical technique, the Bayes classifier, and an alternative Fourier spectral technique achieve significantly worse classification performance. Finally, in Chapter 8 we apply the similarity measure as a *validation measure* for the learned HCS models in Chapters 4 and 5. We confirm our qualitative analysis of model fidelity and observe that the discontinuous modeling framework exhibits markedly better similarity with the human training data than the continuous HCS models.

Chapter 2

Experimental design

2.1 Motivation

Human control strategy, as we have defined the term, encompasses a large set of human-controlled tasks. It is neither practical nor possible to investigate all of these tasks in a finite amount of time. In this thesis, we therefore look towards a prototypical control application — the task of *human driving* — to collect and model control strategy data from different human subjects.

Within the driving domain, we have a choice between *simulated* driving (i.e. driving through a simulator) and *real* driving. For our purposes, the ideal control task should embody several desirable qualities. First, during the execution of the control task, the human subject must not be injured or harmed in any way (short of wounded pride). Second, the human subject should have prior experiences that will help him complete the control task successfully. Third, the control task should pose a significant challenge to the human controller. Finally, the task should be complex enough that it allows for variations in strategy across different individuals.

Let us examine real driving in the context of these four criteria (safety, prior experience, control difficulty and control strategy variations). First, unless we ask individuals to drive very conservatively, it is difficult to guarantee the safety of our human subjects in real driving experiments. If we do ask them to drive conservatively, however, the control task will not be very challenging; moreover, variations between individuals will be somewhat muted. Finally, with respect to prior experience, real driving measures up to the qualities we seek in our control task.

Simulated driving, on the other hand, differs from real driving in a number of important respects. Most importantly, the human subject poses no threat to himself or others while driving in the simulator, no matter how recklessly he chooses to drive. Consequently, unlike in real driving, we can challenge individuals to drive near the edge of their abilities. This produces driving control strategies that are richer and more complex than their real counterparts. Because of this increased complexity, the demonstrated control strategies will potentially exhibit greater variations from one individual to the next. Finally, while human subjects may not be familiar with respect to a specific driving simulator prior to testing, they can, as experienced drivers, transition from real driving to simulated driving with relative ease and efficiency.

Table 2-1 summarizes the above discussion. *With respect to our goal of comparing and modeling human control strategies*, simulated driving embodies more of the qualities which we desire. Thus, we choose simulated driving as our primary control task. We emphasize that in choosing simulated driving, we *do not* suggest that simulation is *in general* better than reality for experimentation.[1] We only suggest that since the focus of this thesis is the human control strategies *themselves*, a simulated task can be appropriate if it bears substantial resemblance to

Table 2-1: Simulated *vs.* real driving

Category	*Simulated driving*	*Real driving*
Safety	**High**	Low
Prior experience	**Medium**	**High**
Control difficulty	**High**	Low
Control strategy variations	**High**	**Medium**

1. There are aspects of a real task that cannot be modeled well in a simulation environment. These include measurement and sensor noise, variable road conditions, etc.

a comparable real task. We believe that our driving simulation environment, which we describe in detail in the following section, does meet that criterion.

2.2 Simulation environment

2.2.1 Dynamic driving simulator

Figure 2-1 shows the real-time, dynamic, graphic driving simulator which we have developed for collecting and analyzing human control strategy data. In the interface, the human operator has full control over the steering of the car (mouse movement), the brake (left mouse button) and the accelerator (right mouse button); the middle mouse button corresponds to slowly easing off whichever pedal is currently being “pushed.” The vehicle dynamics are given in (2-1) through (2-19) below (modified from [46]):

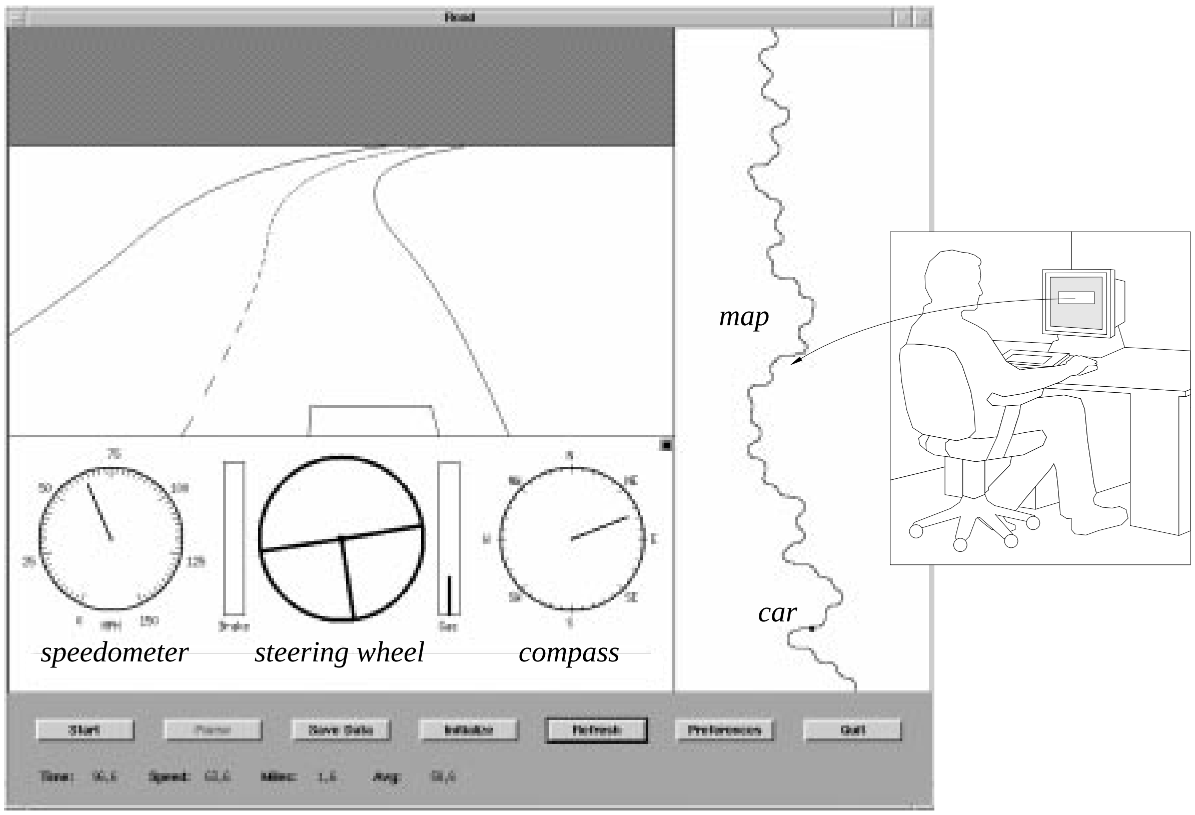


Figure 2-1: Our driving simulator generates a perspective view of the road for the user, who has independent control over steering, braking, and acceleration (gas).

$$\dot{\omega} = (l_f \phi_f \delta + l_f F_{\xi f} - l_r F_{\xi r}) / I \tag{2-1}$$

$$\dot{v}_\xi = (\phi_f \delta + F_{\xi f} + F_{\xi r}) / m - v_\eta \omega - (\operatorname{sgn} v_\xi) c_D v_\xi^2 \tag{2-2}$$

$$\dot{v}_\eta = (\phi_f + \phi_r - F_{\xi f} \delta) / m + v_\xi \omega - (\operatorname{sgn} v_\eta) c_D v_\eta^2 \tag{2-3}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_\xi \\ v_\eta \\ \omega \end{bmatrix} \tag{2-4}$$

where $\{x, y, \theta\}$ describe the Cartesian position and orientation of the car; v_ξ is the lateral velocity of the car; v_η is the longitudinal velocity of the car; and ω is the angular velocity of the car. Furthermore,

$$F_{\xi k} = \mu F_{zk} (\tilde{\alpha}_k - (\operatorname{sgn}\delta) \tilde{\alpha}_k^2 / 3 + \tilde{\alpha}_k^3 / 27) \sqrt{1 - \phi_k^2 / (\mu F_{zk})^2 + \phi_k^2 / c_k^2},\ k \in \{f, r\}, \tag{2-5}$$

$$\tilde{\alpha}_k = c_k \alpha_k / (\mu F_{zk}),\ k \in \{f, r\}, \tag{2-6}$$

$$\alpha_f = \text{front tire slip angle} = \delta - (l_f \omega + v_\xi) / v_\eta, \tag{2-7}$$

$$\alpha_r = \text{rear tire slip angle} = (l_r \omega - v_\xi) / v_\eta, \tag{2-8}$$

$$F_{zf} = (mgl_r - (\phi_f + \phi_r) h) / (l_f + l_r), \tag{2-9}$$

$$F_{zr} = (mgl_f + (\phi_f + \phi_r) h) / (l_f + l_r), \tag{2-10}$$

$$\xi = \text{body-relative lateral axis},\ \eta = \text{body-relative longitudinal axis} \tag{2-11}$$

$$c_f, c_r = \text{cornering stiffness of front, rear tires} = 50000 \text{N/rad, } 64000 \text{ N/rad} \tag{2-12}$$

$$c_D = \text{lumped coefficient of drag (air resistance)} = 0.0005 \text{m}^{-1} \tag{2-13}$$

$$\mu = \text{coefficient of friction} = 1, \tag{2-14}$$

$$F_{jk} = \text{frictional forces}, \ j \in \{\xi, z\}, \ k \in \{f, r\} \tag{2-15}$$

$$\phi_r = \text{longitudinal force on rear tires} = \begin{cases} 0 & \phi_f > 0 \\ k_b \phi_f & \phi_f < 0, \ k_b = 0.34 \end{cases} \tag{2-16}$$

$$m = 1500\text{kg}, \ I = 2500\text{kg-m}^2, \ l_f = 1.25\text{m}, \ l_r = 1.5\text{m}, \ h = 0.5\text{m}. \tag{2-17}$$

The controls are given by,

$$-0.2\text{rad} \le \delta \le 0.2\text{rad} \tag{2-18}$$

$$-8000\text{N} \le \phi = \phi_f \le 4000\text{N} \tag{2-19}$$

where δ is the user-controlled steering angle, and ϕ is the user-controlled longitudinal force on the front tires. Note that the separate brake and gas commands for the human are, in fact, the single ϕ variable, where the sign indicates whether the brake or the gas is active.

Because of input device constraints, the force (or acceleration) control ϕ is limited during each 1/50 second time step, based on its present value. If the gas pedal is currently being applied ($\phi > 0$), then the human operator can either increase or decrease the amount of applied force by a user-specified constant $\Delta\phi_g$ or switch to braking. Similarly, if the brake pedal is currently being applied ($\phi < 0$) the operator can either increase or decrease the applied force by a second constant $\Delta\phi_b$ or switch to applying positive force. Thus, the $\Delta\phi_g$ and $\Delta\phi_b$ constants define the responsiveness of each pedal. If we denote $\phi(k)$ as the current applied force and $\phi(k+1)$ as the applied force for the next time step, we can write in concise notation,

$$\phi(k+1) \in \{\phi(k), \min(\phi(k) + \Delta\phi_g, 4000), \max(\phi(k) - \Delta\phi_g, 0), -\Delta\phi_b\}, \ \phi(k) \ge 0 \tag{2-20}$$

$$\phi(k+1) \in \{\phi(k), \max(\phi(k) - \Delta\phi_b, -8000), \min(\phi(k) + \Delta\phi_b, 0), \Delta\phi_g\}, \ \phi(k) < 0 \tag{2-21}$$

2.2.2 Road descriptions

In the simulator, we define roads as a sequence of randomly generated segments of the form $\{l, 0\}$ (straight-line segments), and $\{r, \beta\}$ (curves), connected in a manner that ensures continuous first derivatives between segments. Here, l is the length of a given straight line segment, r is the radius of curvature of a curved segment, and β is its corresponding sweep angle, defined to be negative for left curves, and positive for right curves.

In order to make the driving task challenging, we place the following constraints on the individual segments:

$$100\text{m} \le l \le 200\text{m}\,,\ 100\text{m} \le r \le 200\text{m}\,, \text{ and } 20° \le |\beta| \le 180°\,. \tag{2-22}$$

No segment may be followed by a segment of the same type; a curve is followed by a straight line segment with probability 0.4, and an opposite curve segment with probability 0.6. A straight line segment is followed by a left curve or right curve with equal probability. Roads are defined to be 10m wide (the car is 2m wide), and the visible horizon is set to 100m. For notational convenience, let d_ξ denote the car's lateral offset from the road median.

Figure 2-2(a), (b) and (c) shows roads #1, #2 and #3, respectively, the three 20km roads over which we collected human driving control data. Figure 2-3(a) shows road #4, the 20km road which we used as a cross-validation road for each modeling approach. Finally, Figure 2-3(b) shows road #5, the 20km road which we reserve for testing each of the modeling approaches.[2]

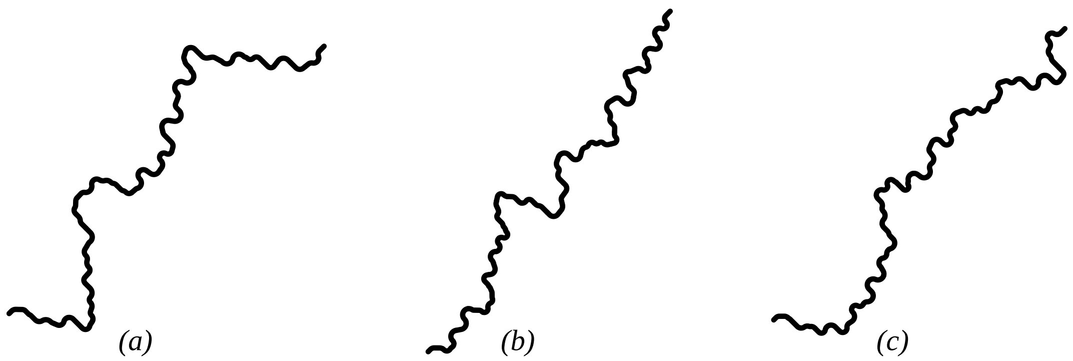


Figure 2-2: Data collection roads: (a) road #1, (b) road #2, (c) road #3.

2. The precise meaning of the terms *cross-validation road* and *test road* will be explained for each modeling approach separately.

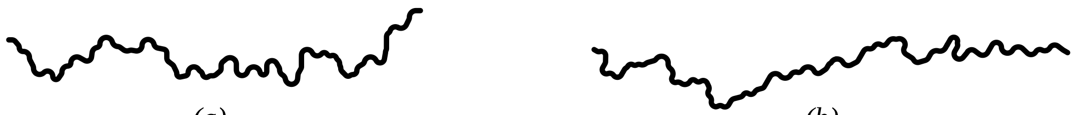


Figure 2-3: (a) Cross-validation road #4 and (b) test road #5.

2.3 Model class

We restrict the class of models we look at in this thesis to *static* (as opposed to *dynamic*) mappings between inputs and outputs. Because human control strategy is dynamic, we must map that dynamic system (i.e. the human control strategy) onto a static map.

In general, we can approximate any dynamic system through the difference equation [79],

$$\begin{aligned}\bar{u}(k+1) &= \\ &\Gamma[\bar{u}(k), \bar{u}(k-1), \ldots, \bar{u}(k-n_u+1), \bar{x}(k), \bar{x}(k-1), \ldots, \bar{x}(k-n_x+1), \bar{z}(k)]\end{aligned} \tag{2-23}$$

where $\Gamma(\ \cdot\ )$ is some (possibly nonlinear) map, $\bar{u}(k)$ is the control vector, $\bar{x}(k)$ is the system state vector, and $\bar{z}(k)$ is a vector describing the external environment at time step *k*. The order of the dynamic system is given by the constants n_u and n_x, which may be infinite. Thus, a static model can abstract a dynamic system, provided that time-delayed histories of the state and command vectors are presented to the model as input, as illustrated in Figure 2-4.

For the case of the driving simulator, the HCS model will require, in general, (1) current and previous state information $\bar{x} = \begin{bmatrix} v_\xi & v_\eta & \omega \end{bmatrix}^T$, (2) previous control information $\bar{u} = \begin{bmatrix} \delta & \phi \end{bmatrix}^T$, and a description of the road $\bar{z}$, visible from the current car position and orientation, where,

$$\bar{z} = \begin{bmatrix} r_x(1) & r_x(2) & \ldots & r_x(n_r) & r_y(1) & r_y(2) & \ldots & r_y(n_r) \end{bmatrix},^{3} \tag{2-24}$$

is a $2n_r$-length vector of equivalently spaced, body-relative (x, y) coordinates (r_x, r_y) of the visible view of the road (median) ahead.

3. This representation is reasonable, since computer vision algorithms such as RALPH [88] can abstract a very similar representation from real camera images.

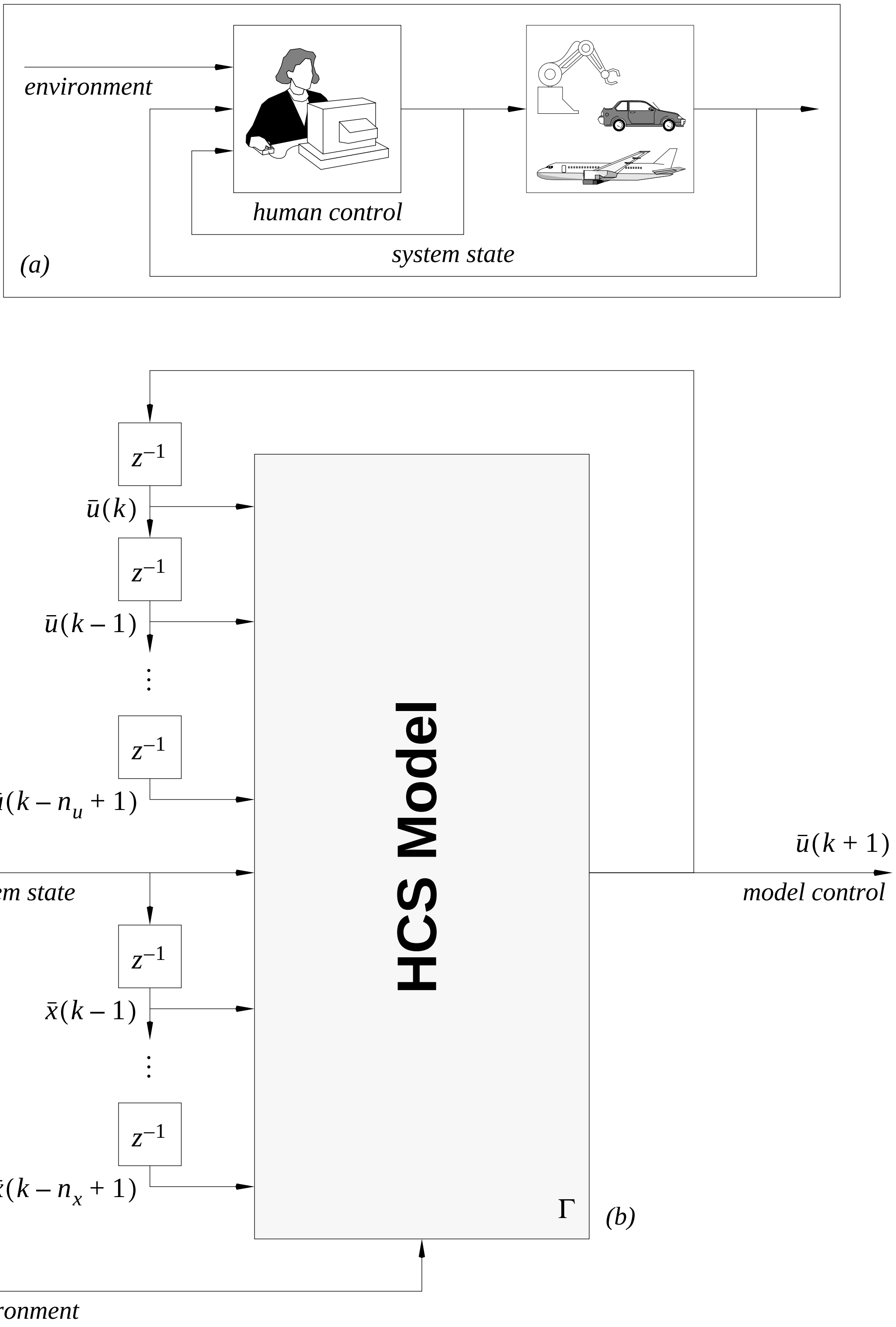


Figure 2-4: In modeling human control strategy (HCS), we want to replace the human controller (a) by a HCS model (b).

For notational convenience, we will denote the HCS model's input space for the driving simulator as,

$$\{v_\xi^{n_1}, v_\eta^{n_2}, \omega^{n_3}, \delta^{n_4}, \phi^{n_5}, r_x^{n_6}, r_y^{n_7}\}, \; n_i \geq 0, \; i \in \{1, 2, \ldots, 7\}, \tag{2-25}$$

where at time step k,

$$\chi^{n_i} = \begin{cases} \left[\chi(k-n_i+1) \;\ldots\; \chi(k-1) \;\chi(k)\right]^T & \chi \in \{v_\xi, v_\eta, \omega, \delta, \phi\} \\ \left[\chi(1) \;\ldots\; \chi(n_i-1) \;\chi(n_i)\right]^T & \chi \in \{r_x, r_y\} \end{cases}. \tag{2-26}$$

Thus, the total number of inputs n_{in} is given by,

$$n_{in} = \sum_{i=1}^{7} n_i . \tag{2-27}$$

We omit χ^{n_i} from the list in equation (2-26) if $n_i = 0$. For example, $\{\delta^3, r_x^{10}, r_y^{10}\}$ denotes a model whose input space consists of the previous three steering commands and a view of the road ahead, discretized to 10 body-relative coordinates. Unless otherwise noted, each time step k is $\tau = 1/50\text{sec}$ long. Finally, when $n_1 = n_2 = n_3$, $n_4 = n_5$, and $n_6 = n_7$, we use the short-hand notation,

$$\{\bar{x}^{n_x}, \bar{u}^{n_u}, \bar{z}^{n_r}\} = \{v_\xi^{n_x}, v_\eta^{n_x}, \omega^{n_x}, \delta^{n_u}, \phi^{n_u}, r_x^{n_r}, r_y^{n_r}\}, \; n_s, n_c, n_r \geq 0, \tag{2-28}$$

$$n_{in} = 3n_x + 2n_u + 2n_r \tag{2-29}$$

to denote the input space in equation (2-25).

Chapter 3

Cascade Neural Networks with Kalman Filtering

Here, we develop a continuous learning architecture for modeling human control strategies, based on neural networks. Unfortunately, most neural networks used today rely on rigid, fixed-architecture networks and/or slow, gradient descent-based training algorithms (e. g. backpropagation). In this chapter, we propose a new neural network learning architecture to counter these problems. Namely, we combine (1) flexible cascade neural networks, which dynamically adjust the size of the neural network as part of the learning process, and (2) node-decoupled extended Kalman filtering (NDEKF), a fast converging alternative to backpropagation. As we shall see later, for reasons of computational complexity, the resulting learning architecture is limited to applications where the number of correlated outputs are few. Generally, this is not a significant restriction in modeling human control strategies, since for such models, the number of outputs — namely, the human controls — are typically few in number.

Thus, in this chapter we first review how learning proceeds in cascade neural networks. We then show how NDEKF fits seamlessly into the cascade learning framework, and how cascade learning addresses the poor local minima problem of NDEKF reported in [92]. We analyze the computational complexity of our approach and compare it to fixed-architecture training paradigms. Finally, we report learning results for real-valued function approximation and dynamic system identification — results which show substantial improvement in learning speed and error convergence over alternative neural network training methods. In Chapter 4, we will then investigate the proposed learning architecture for abstracting HCS models.

3.1 Motivation

In recent years, artificial neural networks have shown great promise in identifying complex nonlinear mappings from observed data and have found many applications in robotics and other nonlinear control problems [8, 73, 79]. As such, they have received a great deal of attention in the learning community. Despite significant progress in the application of neural networks to many real-world problems, however, the vast majority of neural network research still relies on *fixed-architecture* networks trained through *backpropagation* or some other slightly enhanced gradient descent algorithm. There are two main problems with this prevailing approach. First, the "appropriate" network architecture varies from application to application; yet, it is difficult to guess this architecture — the number of hidden units and number of layers — *a priori* for a specific application without some trial and error. Even within the same application, functional complexity requirements can vary widely, as might be the case in modeling human control strategies from different individuals. Second, backpropagation and other gradient descent techniques tend to converge rather slowly. Since the backpropagation algorithm adjusts one weight at a time, the current weight change in the network frequently contradicts one or more of the previous weight adjustments, leading to oscillatory behavior and convergence to poor local minima [31, 47].

To address the problem of fixed architectures in neural networks, we look towards flexible cascade neural networks [32, 33]. In cascade learning, the network topology is not fixed prior to learning, but rather adjusts dynamically as a function of learning, as hidden units are added to an initially minimal network one at a time. This not only frees us from an *a priori* choice of network architecture, but also allows new hidden units to assume variable activation functions [80, 81]. That is, each hidden unit's activation function no longer need be confined to just a sigmoidal nonlinearity. *A priori* assumptions about the underlying functional form of the mapping we wish to learn are thus minimized.

To address the second problem — slow convergence with gradient-descent training algorithms — we look towards *extended Kalman filtering (EKF)* [2], previously confined to the area of optimal filtering. What makes EKF algorithms attractive is that, unlike backpropagation, they explicitly account for the pairwise interdependence of the weights in the neural network during training. By viewing the training of feedforward neural networks as an identification problem for a nonlinear dynamic system, Singhal and Wu [108] were the first to show how the EKF algorithm can be used for neural network training. While converging to better local minima in many fewer epochs than backpropagation, their *global extended Kalman filtering (GEKF)* approach, carries a heavy computational toll. GEKF's computational complexity is $O(n_w^2)$, where n_w is the number of weights in the neural network. This is prohibitive, even for moderately sized neural networks, where the weights can easily number in the thousands.

To address this problem, Puskorius and Feldkamp [92], propose *node-decoupled extended Kalman filtering (NDEKF)*, which considers only the pairwise interdependence of weights feeding into the same node, rather than the interdependence of all the weights in the network. While this approach is computationally tractable through a significant reduction in the computational complexity, the authors report that NDEKF tends to converge to poor local minima, for network architectures not carefully selected to have little redundancy (i.e. few excess free parameters).

In this chapter we show that combining cascade neural networks with NDEKF solves the problem of poor local minima reported in [92], and that the resulting learning architecture substantially outperforms other neural network training paradigms in learning speed and/or error convergence for learning tasks important in control problems. Below, we first describe how learning proceeds in cascade neural networks. We then show how NDEKF fits seamlessly into the cascade learning framework, and how cascade learning addresses the poor local minima problem of NDEKF. We analyze the computational complexity of our approach and compare it to fixed-architecture training paradigms. Finally, we report learning results for continuous function approximation and dynamic system identification. In the following chapter, we then apply the learning architecture proposed here towards modeling human control strategies.

3.2 Cascade neural networks

In learning human control strategies, we wish to approximate the functional mapping between sensory inputs and control action outputs which guide an individual's actions. Function approximation, in general, consists of two parts: (1) the selection of an appropriate functional form, and (2) the adjustment of free parameters in the functional model to optimize some criterion. For most neural networks used today, the learning process consists of (2) only, since a specific functional form is selected prior to learning; that is, the network architecture is usually fixed before learning begins.

We believe, however, that both (1) and (2) above have a place in the learning process. Thus, for modeling human control strategy, we look towards the flexible cascade learning architecture [33], which adjusts the structure of the neural network as part of learning. The cascade learning architecture combines the following two notions: (1) a cascade architecture, in which hidden units are automatically added one at a time to an initially minimal network, and (2) the learning algorithm which creates and installs new hidden units as the learning requires in order to reduce the RMS error (e_{RMS}) between the network's outputs and the training data.

As originally formulated in [33], *cascade neural network* training proceeds in several steps. Initially, there are no hidden units in the network, only direct input-output connections. These weights are trained first, thereby capturing any linear relationship between the inputs and outputs. With no further significant decrease in the RMS error between the network outputs and the training data (e_{RMS}), a first hidden unit is added to the network from a pool of *candidate* units. Using the quickprop algorithm [31] — an improved version of the standard backprop algorithm — these candidate units are trained independently and in parallel with different random initial weights.

Again, after no more appreciable error reduction occurs, the best candidate unit is selected and installed in the network. Once installed, the hidden-unit input weights are frozen, while the weights to the output units are retrained. By freezing the input weights for all previous hidden units, each training cycle is equivalent to training a three-layer feedforward neural network with a single hidden unit. This allows for much faster convergence of the weights during training than in a standard multi-layer feedforward network where many hidden-unit weights are trained simultaneously. The process is repeated until the algorithm succeeds in reducing e_{RMS} sufficiently for the training set or the number of hidden units reaches a specified maximum number. Figure 3-1 below illustrates, for example, how a two-input, single-output network grows as two hidden units are added. Note that a new hidden unit receives as input connections from the input units as well as all previous hidden units (hence the name “cascade”). A cascade network with n_{in} input units (including the bias unit), n_h hidden units, and n_o output units, has n_w connections where,

$$n_w = n_{in}n_o + n_h(n_{in} + n_o) + (n_h - 1)n_h/2 \tag{3-1}$$

Recent theorems by Cybenko [27] and Funahashi [37], which hold that standard layered neural networks are universal function approximators also hold for the cascade network topology, since any multi-layer feedforward neural network with k hidden units arranged in m layers, fully connected between consecutive layers, is a special case of a cascade network with k hidden units and some weight connections equal to zero.

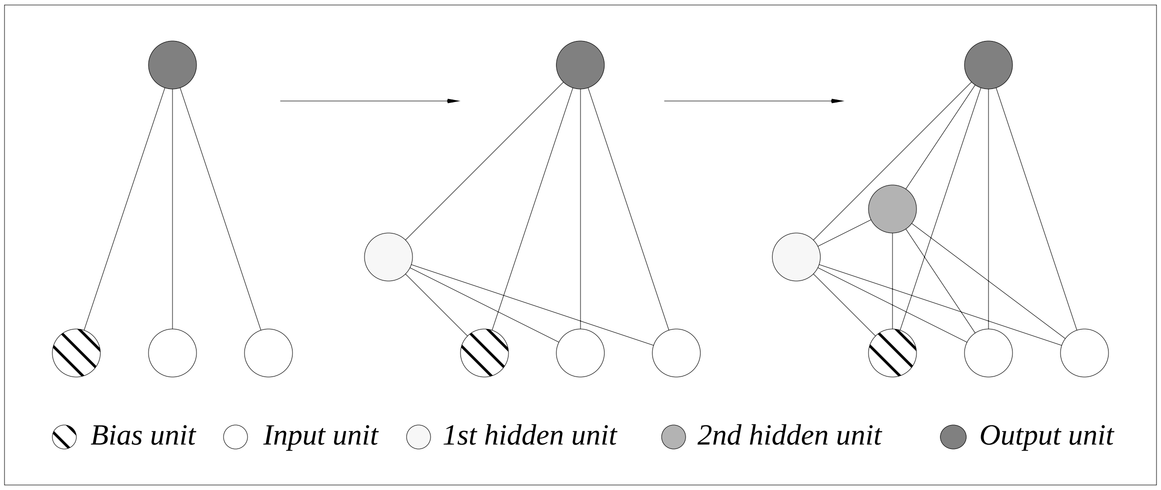


Figure 3-1: The cascade learning architecture adds hidden units one at a time to an initially minimal network. All connections in the diagram are feedforward.

Thus, the cascade architecture relaxes *a priori* assumptions about the functional form of the model to be learned by dynamically adjusting the network size. We can relax these assumptions further by allowing new hidden units to have variable activation functions [80, 81]. In fact, Cybenko [27] shows that sigmoidal functions are not the only possible activation functions which allow for universal function approximation. There are other nonlinear functions, such as *sine* and *cosine* for example, which are complete in the space of *n*-dimensional continuous functions. In the pool of candidate units, we can assign a different nonlinear activation function to each unit, rather than just the standard sigmoidal function. During candidate training, the algorithm will select for installment whichever candidate unit reduces e_{RMS} for the training data the most. Hence, the unit with the most appropriate activation function at that point during training is selected. Typical alternatives to the sigmoidal activation function are the Gaussian function, Bessel functions, and sinusoidal functions of various frequency [80].

3.3 Node-decoupled extended Kalman filtering

While quickprop is an improvement over the standard backpropagation algorithm for adjusting the weights in the cascade network, it can still require many iterations until satisfactory convergence is reached [31, 108]. Thus, we modify standard cascade learning by replacing the quickprop algorithm with *node-decoupled extended Kalman filtering* (*NDEKF*), which has been shown to have better convergence properties and faster training times than gradient-descent techniques for fixed-architecture multi-layer feedforward networks [92]. As we demonstrate later, the combination of cascade learning and NDEKF alleviates critical problems that each exhibits by itself, and better exploits the main strengths of both algorithms.

3.3.1 Learning architecture

In *general extended Kalman filtering (GEKF)* [108], an $n_w \times n_w$ conditional error covariance matrix P, which stores the interdependence of each pair of n_w weights in a given neural network, is explicitly generated. NDEKF reduces this computational and storage complexity by — as the name suggests — decoupling weights by node, so that we consider only the interdependence of weights feeding into the same unit (or node). This, of course, is a natural formulation for cascade learning, since we only train the input-side weights of one hidden unit and the output units at any one time; we can partition the weights by unit into $n_o + 1$ groups — one group for the current hidden unit, n_o groups for the output units. In fact, by iteratively training one hidden unit at a time and then freezing that unit's weights, we minimize the potentially detrimental effect of the node-decoupling.

Denote ω_k^i as the input-side weight vector of length n_w^i at iteration k, for unit $i \in \{0, 1, \ldots, n_o\}$, where $i = 0$ corresponds to the current hidden unit being trained, and $i \in \{1, \ldots, n_o\}$ corresponds to the ith output unit, and,

$$n_w^i = \begin{cases} n_{in} + n_h - 1 & i = 0 \\ n_{in} + n_h & i \in \{1, \ldots, n_o\} \end{cases} \tag{3-2}$$

The NDEKF weight-update recursion is then given by,

$$\omega_{k+1}^i = \omega_k^i + \{(\psi_k^i)^T (A_k \xi_k)\} \phi_k^i \tag{3-3}$$

where ξ_k is the n_o-dimensional error vector for the current training pattern, ψ_k^i is the n_o-dimensional vector of partial derivatives of the network's output unit signals with respect to the ith unit's net input, and,

$$\phi_k^i = P_k^i \zeta_k^i \tag{3-4}$$

$$A_k = \left[I + \sum_{i=0}^{n_o} \{(\zeta_k^i)^T \phi_k^i\} [\psi_k^i (\psi_k^i)^T] \right]^{-1} \tag{3-5}$$

$$P_{k+1}^i = P_k^i - \{(\psi_k^i)^T (A_k \psi_k^i)\} [\phi_k^i (\phi_k^i)^T] + \eta_Q I \tag{3-6}$$

$$P_0^i = (1/\eta_P) I \tag{3-7}$$

where ζ_k^i is the n_w^i-dimensional input vector for the ith unit, and P_k^i is the $n_w^i \times n_w^i$ approximate conditional error covariance matrix for the ith unit. We include the parameter η_Q in (3-

6) to alleviate singularity problems for error covariance matrices [92]. In (3-3) through (3-6), {}’s, ()’s, and []’s evaluate to scalars, vectors and matrices, respectively.

The ψ_k^i vector is easy to compute within the cascade framework. Let O_i be the value of the *i*th output node, Γ_O be its corresponding activation function, net_{Oi} be its net activation, Γ_H be the activation function for the current hidden unit being trained, and net_H be its net activation. Then,

$$\frac{\partial O_i}{\partial net_{Oj}} = 0, \forall i \neq j \tag{3-8}$$

$$\frac{\partial O_i}{\partial net_{Oi}} = \Gamma'_O(net_{Oi}), i \in \{1, \ldots, n_o\} \tag{3-9}$$

$$\frac{\partial O_i}{\partial net_H} = w_{Hi} \cdot \Gamma'_O(net_{Oi}) \cdot \Gamma'_H(net_H) \tag{3-10}$$

where w_{Hi} is the weight connecting the current hidden unit to the *ith* output unit.

Throughout the remainder of the paper, we will use the short-hand notation explained in Table 3-1 for different neural network training methodologies.

Table 3-1: Notation

Symbol	*Methodology*	*Training algorithm*
Fq	*Fixed architecture*[a]	*quickprop*
Cq	*Cascade learning*[b]	*quickprop*
Fk	*Fixed architecture*	*NDEKF*
Ck	*Cascade learning*	*NDEKF*

a. All weights are trained simultaneously.
b. Hidden units are added and trained one at a time.

3.3.2 Computational complexity

The computational complexity for cascade learning with NDEKF is,

$$O\left(n_o^3 + \sum_{i=0}^{n_o} (n_w^i)^2\right). \tag{3-11}$$

The $O(n_o^3)$ computational complexity caused by the matrix inversion in (3-5) restricts this approach to applications where the number of outputs is relatively few. Below, we compare the computational complexity of the proposed learning architecture to two other regimes: (1) layered feedforward neural networks trained with backpropagation (pattern-wise update), and (2) NDEKF alone (i.e. used on fixed-architecture networks).

First, consider the computational cost (per training pattern) in Ck of training one candidate unit for a network with n_{in} input units, $(i-1)$ hidden units, and n_o output units:

$$\text{cost}(A^{-1}) \ (n_o \times n_o \text{ symmetric matrix inversion}), \tag{3-12}$$

$$2n_t^2(n_o+1) + 4n_t n_o + n_o^3 + 4n_o^2 + 7n_o - 2 \text{ multiplications}, \tag{3-13}$$

$$(n_t^2(n_o+1) + n_t(9n_o+7) + 2n_o^2 + 4n_o - 16)/2 \text{ additions}, \tag{3-14}$$

$$3n_o + 1 \text{ function evaluations}, \tag{3-15}$$

where $n_t = n_{in} + i$.

For comparison with backpropagation, we look at the computational cost (per training pattern) for a two-layered neural network with n_{in} input units, $n_H/2$ hidden units in both hidden layers, and n_o output units:

$$(5/4)n_H^2 + n_H(2n_{in}+1) + n_o(5/2n_H + 1) \text{ multiplications}, \tag{3-16}$$

$$n_H^2 + (3/2)n_H n_{in} + n_H(2n_o - 2) \text{ additions, and} \tag{3-17}$$

$$2(n_H + n_o) \text{ function evaluations.} \tag{3-18}$$

To arrive at a composite cost for each method, we weigh multiplications and additions by a factor of 1.0, and function evaluations by a factor of 5.0. In addition, we multiply the composite cost for the cascade/NDEKF method by $n_c = 8$, a typical number for the pool of candidate units and average the cost over all $i \in \{1, 2, \ldots, n_h\}$. Let γ_{Ck} denote the average computational cost per training pattern for the Ck method, and let γ_{BP} denote the computational cost per training pattern for training the two-layered network with backpropagation. We are interested in the ratio,

$$\rho = \gamma_{Ck}/\gamma_{BP} \tag{3-19}$$

for equivalently sized neural networks. By *equivalently sized*, we mean neural networks with approximately the same final number of weights, such that,

$$n_w = n_{in}n_o + n_h(n_{in} + n_o) + (n_h - 1)n_h/2 = n_{in}(n_H/2) + n_H^2/4 + n_o(n_H/2) \tag{3-20}$$

In general, therefore, $n_h \neq n_H$. Figure 3-2, for example, plots σ for $n_{in} = \{1, 50, \ldots, 450, 500\}$, $10 \le n_h \le 100$, and $n_o = 1$. We note that for $n_h > 20$ and $n_{in} < 400$, the ratio is upper bounded by $\rho < 100$. In other words, if Ck reduces the number of epochs by a factor of 100 over standard backpropagation, our approach will be more efficient

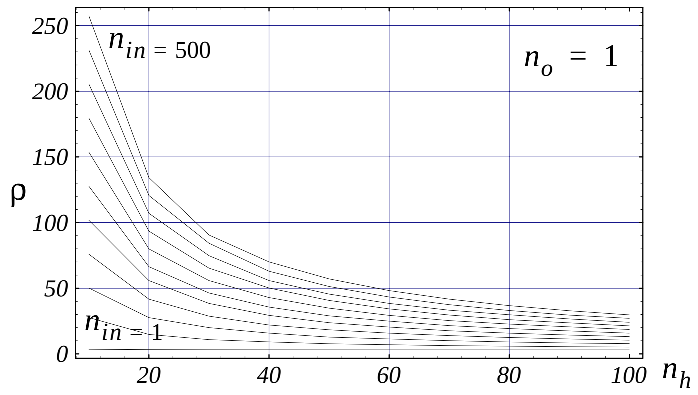


Figure 3-2: Ratio of computational costs per training pattern (between backpropagation and Ck) for various network sizes and one output unit. Higher curves reflect ratios for larger number of inputs.

even for very large input spaces. Moreover, for small input spaces, a mere factor of 5 reduction in the number of epochs will result in increased computational efficiency.

Second, we consider the difference in computational cost between the proposed approach (*Ck*) and using NDEKF alone (*Fk*). Let γ_i^{Ck} denote the cost per epoch of training the *i*th hidden unit; let γ^{Ck} denote the total cost of training the *Ck* network (n_h final hidden units and n_c candidate units per hidden unit); let ε_i^{Ck} denote the number of epochs required to train the *i*th hidden unit; and let ε^{Ck} denote the total number of epochs. Also, let γ_ε^{Fk} denote the cost per epoch of training the *Fk* network; let γ^{Fk} denote the total cost of training the *Fk* network (n_h total hidden units); and let ε^{Fk} denote the total number of epochs for training the *Fk* network. Thus,

$$\gamma^{Ck} = \sum_{i=1}^{n_h} \varepsilon_i^{Ck}\gamma_i^{Ck} = n_c \sum_{i=1}^{n_h} \frac{1}{n_c}(\varepsilon_i^{Ck}\gamma_i^{Ck}) \tag{3-21}$$

$$\gamma^{Fk} = \varepsilon^{Fk}\gamma_\varepsilon^{Fk} \tag{3-22}$$

Now, we assume that,

$$\varepsilon_i^{Ck} \approx \varepsilon_j^{Ck},\ \forall i, j \tag{3-23}$$

so that (3-21) becomes,

$$\gamma^{Ck} = n_c \frac{\varepsilon^{Ck}}{n_h} \sum_{i=1}^{n_h} \frac{1}{n_c}\gamma_i^{Ck} \tag{3-24}$$

Our experience justifies the approximation in (3-23), which states that all hidden units require approximately the same number of epochs. Furthermore, neglecting differences in derivative calculations between methods *Ck* and *Fk*, we assume that,

$$\gamma_\varepsilon^{Fk} \approx \sum_{i=1}^{n_h} \frac{1}{n_c}\gamma_i^{Ck} \tag{3-25}$$

We can now get a relationship between ε^{Ck} and ε^{Fk} corresponding to equivalent costs between methods *Ck* and *Fk*. Setting (3-21) and (3-22) equal to each other and using approximations (3-23) and (3-25), we get that,

$$\varepsilon^{Fk}\gamma_{\varepsilon}^{Fk} \approx n_c \frac{\varepsilon^{Ck}}{n_h}\gamma_{\varepsilon}^{Fk} \tag{3-26}$$

$$\varepsilon^{Ck} \approx \frac{n_h}{n_c}\varepsilon^{Fk} \tag{3-27}$$

In other words, using the cascade/NDEKF (*Ck*) algorithm, we can use approximately n_h/n_c as many epochs as for NDEKF alone (*Fk*) for the same computational cost.

3.4 Comparison experiments

3.4.1 Problem descriptions

In this section, we present learning results for five different problems in continuous function approximation and dynamic system modeling. For the first problem (A), we want to approximate the following 3-to-2 *smooth*, continuous-valued mapping,

$$f_1(x, y, z) = z\sin(\pi y) + x \tag{3-28}$$

$$f_2(x, y, z) = z^2 + \cos(\pi xy) - y^2 \tag{3-29}$$

in the interval $-1 < x, y, z < 1$. The training set consists of 1000 random points; the cross validation set consists of an additional 1000 random points; and our test set consists of another 2000 random points.

Our second problem (B) is taken from [33]. We want to approximate the following 1-to-1 *non-smooth*, continuous-valued mapping (see Figure 3-3),

$$f(x) = a \cdot \min[0.1x^2 \cdot \max(0.5\sin 2x, \sin x),\ x \cdot \max(\sin x, \cos^2 x)] - b \tag{3-30}$$

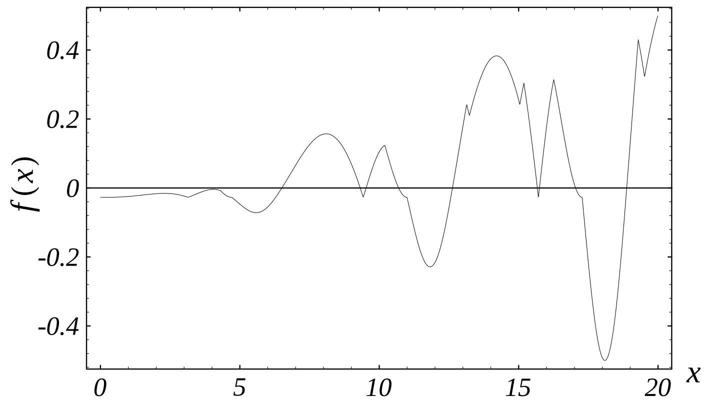


Figure 3-3: Nonsmooth, continuous function for problem (B).

for $a = 1/34.55386$, and $b = 0.027099$. Our training, cross validation, and test sets are identical to those in [33] and consist of the following: (1) 4000 evenly spaced points are generated in the interval $0 \le x < 20$; (2) 968 of those points are randomly chosen for the training set; (3) 968 are randomly chosen from the remaining 3032 points for the cross validation set; and (4) the remaining 2064 points make up the test set.

Our third problem (C) is taken from [79, 92]. We want to model the following dynamic system,

$$u(k+1) = f[u(k), u(k-1), u(k-2), x(k), x(k-1)] \tag{3-31}$$

where,

$$f[x_1, x_2, x_3, x_4, x_5] = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_2^2} \tag{3-32}$$

and the input $x(k)$ is randomly generated in the interval $-1 < x(k) < 1$. We use a 2500-length sequence for training, another 2500-length sequence for cross validation, and another 5000-length sequence for testing. The variables $u(0)$, $u(-1)$ and $u(-2)$ are initialized to zero.

Finally, our last two problems are taken once again from [33]. Here, we want to predict the chaotic Mackey-Glass time series [71], widely studied in the literature and described by,

$$\dot{x}(t) = \frac{a \cdot x(t-\tau)}{1 + x(t-\tau)^{10}} - b \cdot x(t) \tag{3-33}$$

for $a = 0.2$, $b = 0.1$, and $\tau = 17$. While the Mackey-Glass differential equation has infinite degrees of freedom (due to the time delay τ), it's stationary trajectory lies on a low-dimensional attractor, as shown in Figure 3-4. We present,

$$\{x(t-18), x(t-12), x(t-6), x(t)\} \tag{3-34}$$

as the four inputs to the neural network, while the goal of this task is to predict $x(t+k)$ for $k \in \{6, 84\}$. We will refer to $k = 6$ as problem (D) and $k = 84$ as problem (E). Our training, cross validation, and test sets are once again identical to those in [33]. The training set consists of the 500 data points from time $t = 200$ to $t = 699$; the cross validation set consists of the 500 data points from time $t = 1000$ to $t = 1499$; and the test set consists of the 500 points from time $t = 5000$ to $t = 5499$.

For problems (A) and (C) above, we train over 25 trials to 15 hidden units for each method $\{Cq, Fk, Ck\}$. By fixing the network architecture prior to training for *Fk*, it is not possible to assign variable activation functions to each hidden unit; the space of all possible permutations of variable activation functions is too large to explore. Therefore, we try two different networks for method *Fk* — one with sigmoidal activation functions, and the other with sinusoidal activation functions. In [80], we have shown that neural networks with sinusoidal activation functions

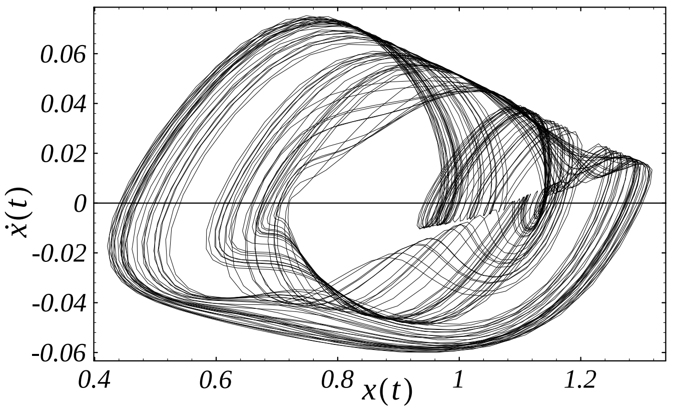


Figure 3-4: Phase plot of the Mackey-Glass time series — (D), (E).

perform approximately as well as those with variable activation functions. Both have been shown to outperform sigmoidal networks for continuous function approximation.

For problems (B), (D), and (E), taken from [33], we follow the same procedure in training with methods $\{Fk, Ck\}$, as Fahlman, *et. al.*, follow in training with methods $\{Fq, Cq\}$. In [33], *Cq* neural networks are allowed to grow to a maximum of 50 hidden units in 15 separate trials for each problem. For each trial, the best RMS error over the test set is recorded. Equivalently sized *Fq* networks are also trained, for up to 60,000 epochs per trial. The 60,000 figure is chosen to be approximately three times the maximum number of epochs required for any of the *Cq* training runs. Again, the best RMS error for the test set is recorded for each trial.

For the learning results in this chapter involving NDEKF, we use the following parameter settings throughout:

$$\eta_Q = 0.0001, \ \eta_P = 0.01 \tag{3-35}$$

In *Ck*, we upper-bound the number of epochs to 10 per hidden unit, while for *Fk*, we upper-bound the total number of epochs to 150. Finally, for the cascade methods $\{Cq, Ck\}$, we use eight candidate units, the same as in [33].

3.4.2 Learning results

Table 3-2 below reports the average RMS error ($\bar{e}_{RMS}$) over the test sets for problems (A) through (E). We note that in all cases, our cascade/NDEKF (*Ck*) approach outperforms the other three methods. Figure 3-5 reports the percentage difference in $\bar{e}_{RMS}$ between our approach and competing training regimes.

Method *Fq* (fixed-architecture/quickprop) shows by far the worst performance, yet we use 120 times fewer epochs for *Ck* (approximately 500 for problems (B), (D), and (E)). Using Figure 3-2 as an approximate guide to the computational difference between *Ck* and *Fq*, we see that, for a 50-hidden unit network with relatively few inputs, a *Ck* epoch is no more than 10 times

Table 3-2: Average RMS Error over test sets[a]

	Ck ($\times 10^{-3}$)	*Fk* ($\times 10^{-3}$)[b]	*Cq* ($\times 10^{-3}$)	*Fq* ($\times 10^{-3}$)
(A)	42.1 (4.2)	127.1 (37.3)	94.5 (6.2)	N/A
(B)	7.4 (2.0)	12.4 (3.2)	14.5 (4.0)	65.0 (18.2)[c]
(C)	15.6 (1.5)	20.7 (4.8)[d]	29.9 (2.0)	N/A
(D)	4.6 (0.6)	10.2 (4.0)	9.4 (2.7)	16.7 (2.2)
(E)	42.0 (5.9)	60.5 (3.1)	72.6 (16.3)	90.3 (8.3)

a. Standard deviations are in parentheses. Shaded cells are results taken from [33].
b. For the *Fk* results we report the better of the sinusoidal or sigmoidal networks (in all cases, the sinusoidal networks did better on average).
c. A fixed-architecture/backprop network failed to converge for this problem, despite many experiments with different learning parameters [33].
d. This is comparable to the result of 0.03 in [92] for a network with an equal number of parameters.

as computationally expensive as an *Fq* epoch. Hence, not only does our cascade/NDEKF approach generate better learning results, it is also more efficient than the fixed-architecture/quickprop approach.

Method *Fk* also performs worse than our *Ck* approach, despite allowing *Fk* to compute as much as twice as long as *Ck*. For problems (A) and (C), for example, the number of epochs required

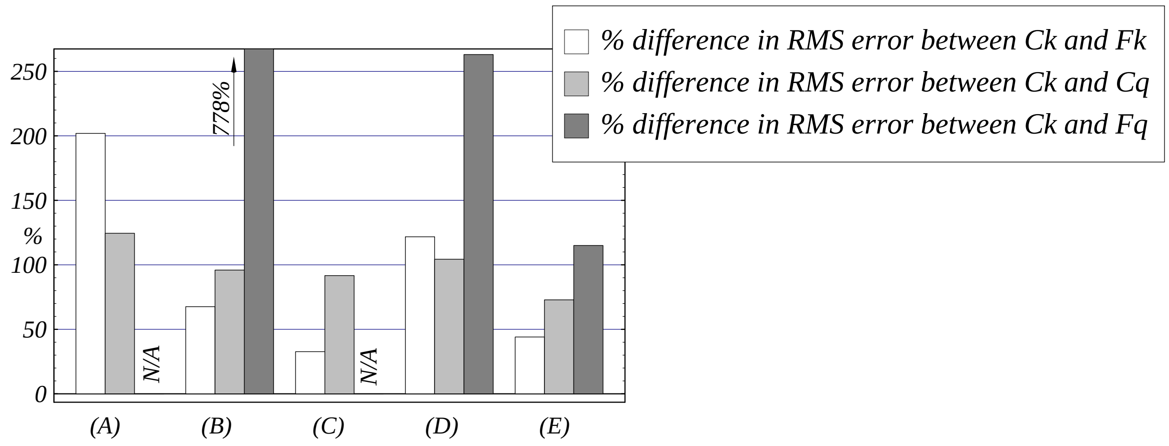


Figure 3-5: Cascade/NDEKF significantly outperforms the other learning methods for each problem.

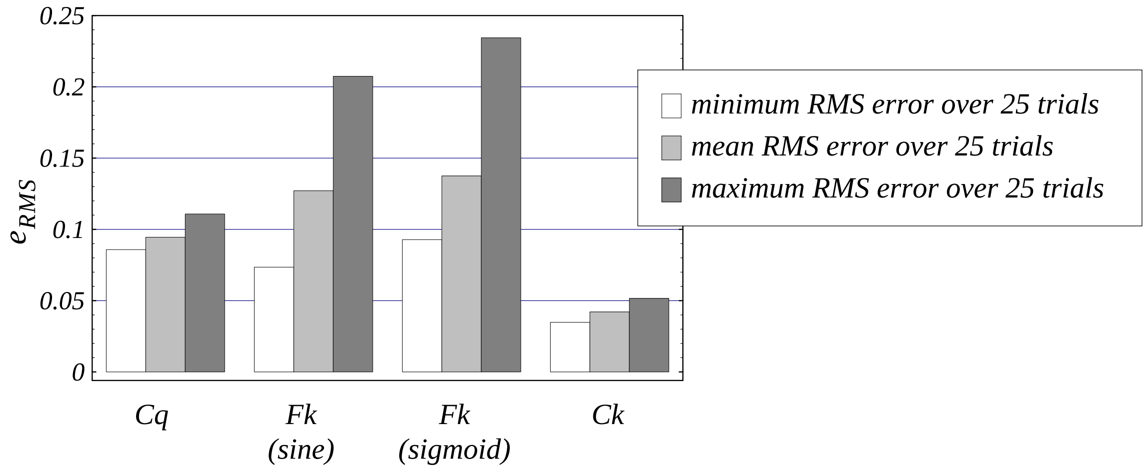


Figure 3-6: *Fk* can get stuck in bad local minima, as witnessed by the large maximum RMS errors observed for problem (A) (for 15-hidden-unit networks).

to train to 15 hidden units for *Ck* is approximately 140. Since we use eight candidate units, a roughly equivalent number of epochs in terms of computational cost for *Fk* is (from (3-27)),

$$\varepsilon^{Fk} \approx \frac{n_c}{n_h}\varepsilon^{Ck} = \frac{8}{15}(140) \approx 75 \tag{3-36}$$

Yet, we allow *Fk* to compute as much as twice that amount — 150 epochs.

One reason, *Fk* shows worse performance is its susceptibility to getting stuck in bad local minima. As the authors of the NDEKF algorithm note, “NDEKF at times requires a small amount of redundancy in the network in terms of the total number of nodes in order to avoid poor local minima for certain problems, which [they attribute] to high effective learning rates at the onset of training [92].” Consider, for example, Figure 3-6. While the minimum e_{RMS} for the *Fk* network is below that of the *Cq* network, its maximum e_{RMS} is much worse than either *Ck* or *Cq*. On the other hand, *Ck* avoids the bad local minima problem by iteratively training only a small number of weights in the network at once.

Finally, we look at the difference between the *Ck* and *Cq* methods. First, we note that *Cq* requires about 15 to 25 times as many epochs as does *Ck*. Since each *Cq* epoch is much less computationally expensive, however, *Cq* consumes only about 2/3 the time as *Ck* for the problems studied here. On the other hand, *Ck* is able to achieve local minima comparable to *Cq*'s with fewer hidden units, and therefore requires significantly less time than *Cq* to reach the same average RMS error. Consider, for example, Figure 3-7. At the onset of training, $\bar{e}_{RMS}$ for *Ck* and *Cq* training is approximately equal (4% difference). As hidden units are added, however, we see that $\bar{e}_{RMS}$ diverges for the two training algorithms. Since each hidden unit receives input from all previous hidden units, the input-side weights of the hidden units become increasingly correlated. Figure 3-8, for example, plots,

$$\rho = \sum_{i \neq j} |P_{ij}| / \sum_i |P_{ii}| \tag{3-37}$$

(i.e. the ratio of off-diagonal terms to diagonal terms in the error covariance matrix *P*) for one trial in problem (C). By explicitly storing the interdependence of these weights in the conditional error covariance matrix, cascade/NDEKF copes better with this increasing correlation than does the cascade/quickprop algorithm.

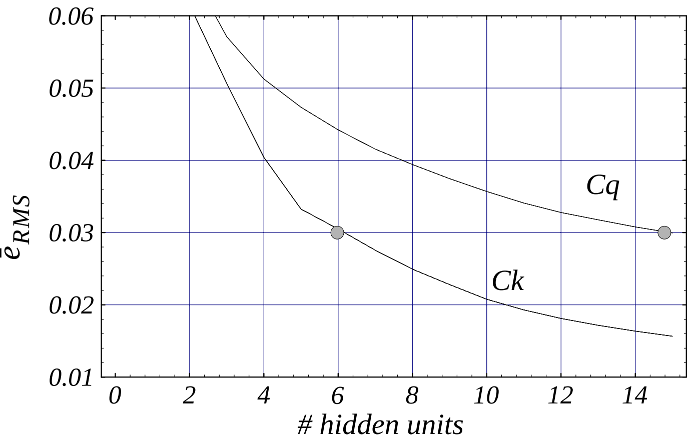


Figure 3-7: *Ck* converges to approximately the same avg. RMS error with 6 hidden units (63 weights) as *Cq* does with 15 hidden units (216 weights) for problem (C).

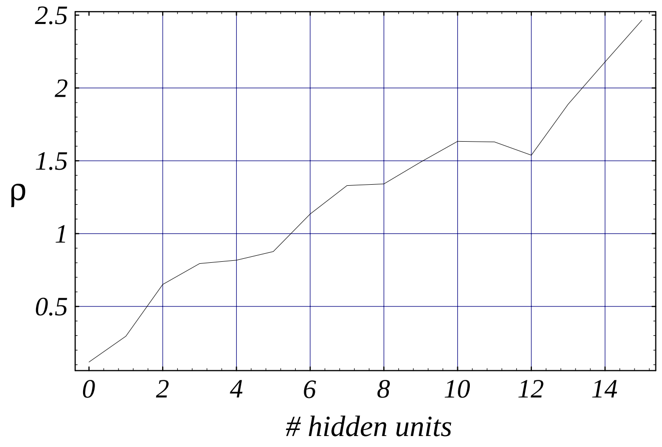


Figure 3-8: Ratio of off-diagonal terms to diagonal terms in the covariance matrix as hidden units are added to the cascade network for one trial in problem (C).

3.4.3 Noisy learning results

The results of Section 3.4.2 demonstrate that *Ck* outperforms competing neural network learning architectures for *noise-free* mappings. In this section, we explore the *Ck* algorithm's performance when we introduce noise to the learning task.

We once again look at problems (A) through (E) defined in Section 3.4.1, only now, we add Gaussian noise with standard deviation σ to both the inputs and outputs of the training, cross-validation and test data sets. Since these results no longer compare with those in [33], we follow the training regime for problems (A) and (C) in the previous section (rather than for problems (B), (D) and (E)). In other words, we train over 25 trials to 15 hidden units for each method $\{Cq, Fk, Ck\}$.

Figure 3-9 plots the percent difference in the average RMS error of the test set over 25 trials between (1) the *Ck* and *Fk* and (2) the *Ck* and *Cq* learning algorithms as a function of the input/output Gaussian noise. For all noise levels, the *Ck* algorithm performs significantly better than the *Fk* algorithm. To a lesser degree and with the exception of problem (B), cascade/NDEKF also outperforms the cascade/quickprop algorithm as the level of noise is increased. For small σ, the difference remains relatively large, while for large $\sigma = 0.05$ (approximately 10% error)

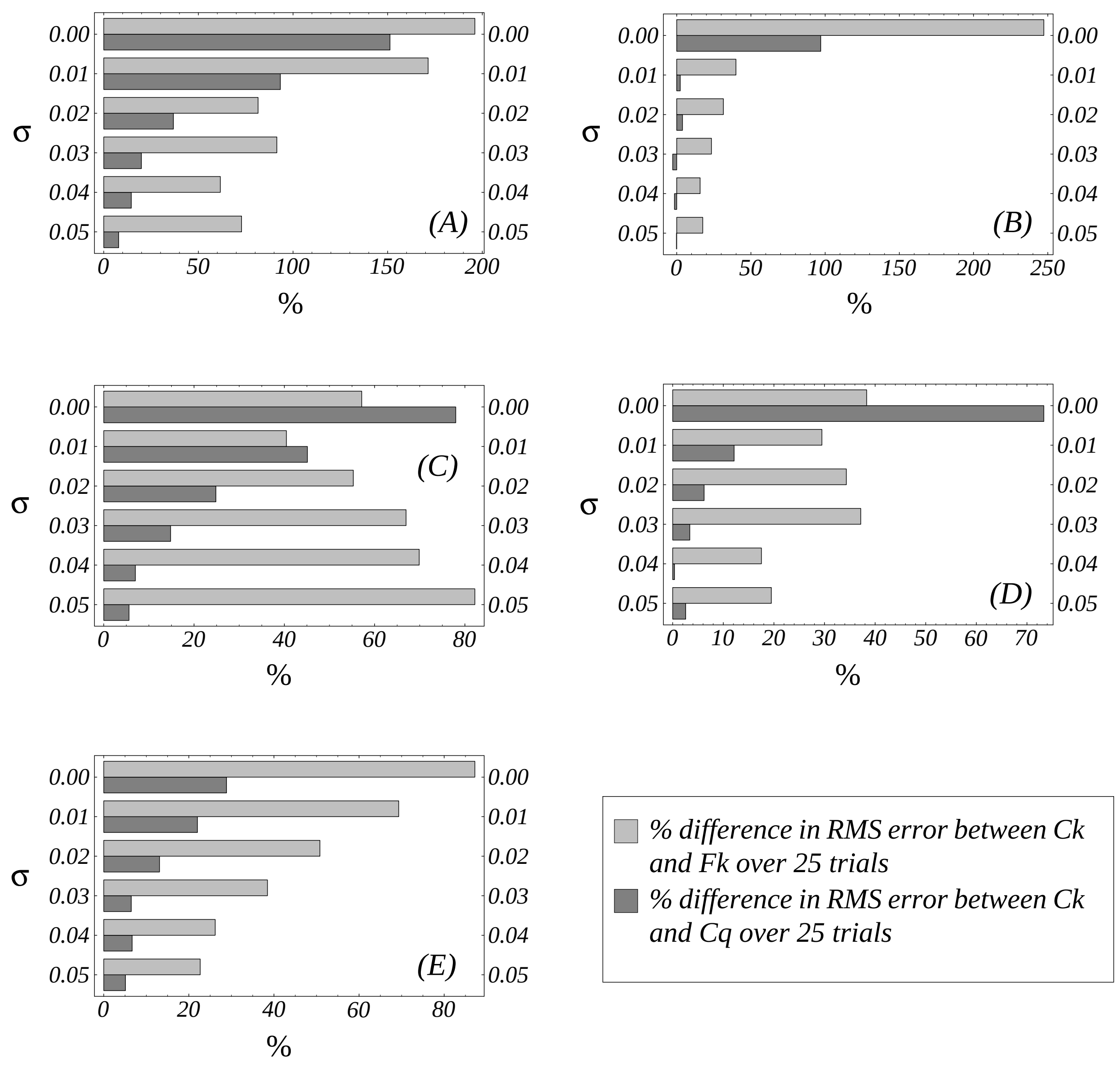


Figure 3-9: When noise is added to the learning task, *Ck*, to a lesser degree, outperforms *Fk* and *Cq* for problems (A), and (C) through (E).

the difference shrinks substantially. Nevertheless, the *Ck* algorithm still achieves marginally better results.

3.4.4 Discussion

From the above results, we make several observations. First, for the test problems studied here, we see a significant improvement in learning times and/or error convergence with cascade/ NDEKF over the other methods. Moreover, we see that incremental cascade learning and node-decoupled extended Kalman filtering complement each other well by compensating for each other's weakness. On the one hand, the idea of training one hidden unit at a time and adding hidden units in a cascading fashion offers a good alternative to the *ad hoc* selection of a network architecture. Quickprop and other gradient-descent techniques, however, become less efficient in optimizing increasingly correlated weights as the number of hidden units rises. This is where NDEKF can perform much better through the conditional error covariance matrix. On the other hand, NDEKF can easily become trapped in bad local minima if a network architecture is too redundant. Cascade learning accommodates this well by training only a small subset of all the weights at one time.

Second, we note that throughout, we used the identical parameter settings for training cascade networks with NDEKF (3-35). This stands in sharp contrast to more *ad hoc* methods such as *Cq* and *Fq*, for which the various learning parameters were tuned for each particular problem in order to achieve good results [33]. The weight-update recursion,

$$\omega_{k+1}^{i} = \omega_{k}^{i} + \{(\psi_{k}^{i})^{T}(A_{k}\xi_{k})\}\phi_{k}^{i} \tag{3-38}$$

can be thought of as an adaptive learning rate, which lessens the need for parameter tuning in NDEKF. Thus, we need spend little time tuning either learning parameters or network architectures in this approach.

Finally, while our approach performs well for the problems studied, it is clearly impractical for applications which have either a large number of inputs or a large number of correlated outputs. This, for example, tends to exclude vision-based tasks, where the input space and/or output space are typically greater than 1000. Applications with inputs numbering in the low hundreds, however, are not excluded.

Thus, in this chapter we have developed a new neural network learning methodology for real-valued function approximation and dynamic system identification. We have shown that incremental cascade learning and NDEKF complement each other well by compensating the other's weakness, and that the combination forms a flexible, powerful learning architecture, which records quicker convergence to better local minima than related neural-network training paradigms. In the next chapter, we reinforce these results as we investigate how cascade learning — specifically *Cq* and *Ck* — perform in modeling human driving control strategies.

Chapter 4

HCS Models: Continuous Learning

In the previous chapter, we developed a general learning architecture that combines cascade neural networks with node-decoupled extended Kalman filtering (NDEKF). Here we apply cascade learning to the problem of modeling human control strategies. We first report results for cascade/quickprop (*Cq*) neural networks. We then compare those results to the cascade/NDEKF (*Ck*) learning architecture. This comparison exposes the weakness of gradient-descent techniques in modeling input-output mappings with correlated inputs, such as HCS models with time-delayed state and control inputs (Figure 2-4). Although the test-set error decreases in *Cq* as multiple hidden units are added to the neural networks over *thousands* of training epochs — indicating a strong linear as well as nonlinear component in the overall human control strategy being modeled — these modeling results are quite deceptive. We show that, in fact, *Ck* converges to equivalent or lower errors over the same training data in *less than one training epoch* of a linear model. Moreover, the linear networks show markedly greater stability over a wider range of initial and environmental conditions.

Since the acceleration control of the simulated car involves (discontinuous) switching between positive and negative applied force (i.e. the gas and the brake, respectively), however, these linear models, while abstracting convergent strategies, qualitatively bear little resemblance to the original human control strategy. We argue that this is not only a shortcoming of our neural-network function approximators, but that, in fact, *any* continuous function approximator is doomed to fail in a similar manner when attempting to model control strategies that involve discontinuous switching. Chapter 5 follows up this discussion by developing a novel, statistical, discontinuous framework for successfully modeling discontinuous human control strategies.

4.1 Cascade with quickprop learning

Here, we present modeling results for the cascade/quickprop (*Cq*) learning architecture. In subsequent sections, we will compare these results to cascade/NDEKF learning (*Ck*).

4.1.1 Experimental data

Appendix A describes driving control data from six different individuals — (1) Larry, (2) Curly, (3) Moe, (4) Groucho, (5) Harpo and (6) Zeppo across three different roads, roads #1, #2 and #3 in Figures 2-2(a), (b) and (c), respectively. For notational convenience, let $X^{(i,j)}$, $i \in \{1, 2, 3, 4, 5, 6\}$, $j \in \{1, 2, 3\}$, denote the run from person (*i*) on road #*j*, sampled at 50Hz.

4.1.2 Model inputs and outputs

We defer to Section 4.2.1 a detailed discussion of the input space choices made for the HCS models described in this section. This is necessary in part because our input selection procedure is dependent on the cascade/NDEKF learning results. For now, we simply note what the choices are for each model. In Table 4-1, n_x, n_u and n_r, as defined in equation (2-28), completely characterize the input space for the results presented below. As we shall see later (Section 4.2.1), model performance remains similar over a wide range of input spaces, especially once a sufficient number of inputs are given. Here, "sufficient number" means that there are enough time-delayed values of each state and control variable such that the model is able to build necessary derivative dependencies between the inputs and outputs. The outputs of each model are, of course, the next steering angle and acceleration command $\{\delta(k+1), \phi(k+1)\}$.

4.1.3 *Cq* training

For each model $\Gamma^{(i,j)}$, we process the training data as follows. First, we excise from the complete run $X^{(i,j)}$ those segments where the human operator temporarily runs off the road ($d_\xi > 5\text{m}$). Let $[t, t + t_l]$ denote an interval of time, in seconds, that a human operator veers off the road. Then, we cut the data corresponding to time interval $[t - 1, t + t_l]$ from the training data. In other words, we not only remove the actual off-road data from the training set, but also the second of data leading up to the off-road interval. This ensures that the HCS model does not learn control behaviors that are potentially destabilizing.

Next, we normalize each input dimension of the HCS model, such that no input in the training data falls outside the interval $[-1, 1]$. Finally we randomize the input-output training vectors and select half for training, while reserving the other half for testing. All the runs $X^{(i,j)}$ are approximately equal to or longer than 10 minutes in length. Thus, at 50Hz, typical training and testing data sets will consist of approximately 15,000 data points each.

Training proceeds until the RMS error in the test data set no longer decreases. We use eight candidate units, and allow up to 500 epochs for candidate as well as for output training. Table 4-1 reports the final number of hidden units n_h for the models presented below.

Table 4-1: Input space for each *Cq* model

Run[a]		*Input space*		n_h[b]	*Figure*
		$n_x = n_u$	n_r		
Larry's second	$X^{(1,2)}$	3	10	12	Figure 4-1
Moe's first	$X^{(3,1)}$	3	10	14	Figure 4-2
Groucho's first	$X^{(4,1)}$	6	10	19	Figure 4-3
Harpo's second	$X^{(5,2)}$	5	10	9	Figure 4-4

a. See Appendix A for a detailed description of each human control data set.
b. Number of hidden units in final *Cq* model.

4.1.4 HCS models

Figures 4-1, 4-2, 4-3 and 4-4 illustrate some representative *Cq* learning results for four different runs: $X^{(1,2)}$ (Larry's second run), $X^{(3,1)}$ (Moe's first run), $X^{(4,1)}$ (Groucho's first run), and $X^{(5,2)}$ (Harpo's second run). Part (a) of each Figure plots the original human control data, while part (b) of each Figure plots the corresponding model control over the test road (#5 in Figure 2-3(b)).

Before comparing these results to *Ck*, we make the following two observations. First, the discontinuous switching in the acceleration control ϕ induces substantial high frequency noise in the *Cq* model control. This noise is especially evident in Larry's and Groucho's models (Figures 4-1 and 4-3, respectively). Second, Harpo's *Cq* model does not converge to a stable control strategy, as it loses complete track of the road after less than 5 seconds.

4.2 Cascade with NDEKF learning

4.2.1 Model inputs and outputs

For the neural networks trained in this chapter, we follow a simple experimental procedure for selecting the input space of each HCS model. Let model $\Gamma_k^{(i,j)}$ correspond to a HCS model trained with *Ck* on training data from run $X^{(i,j)}$ (i.e. person (i) on road #j) and with input space,

$$\{\bar{x}^k, \bar{u}^k, \bar{z}^{10}\},\ k \in \{1, 2, \ldots, 20\}, \tag{4-1}$$

as defined in equation (2-28). Also let,

$$\max(d_{\xi,k}^{(i,j)}),\ k \in \{1, 2, \ldots, 20\} \tag{4-2}$$

denote the maximum lateral offset for model $\Gamma_k^{(i,j)}$ over the 20km validation road (#4) shown in Figure 2-3(a). Then, we select model $\Gamma_l^{(i,j)}$ for testing over the 20km testing road (#5) shown in Figure 2-3(b)[1] such that,

1. We note that roads #4 and #5 (Figure 2-3) are different from the data collection roads #1, #2 and #3 (Figure 2-2), but share similar statistical attributes.

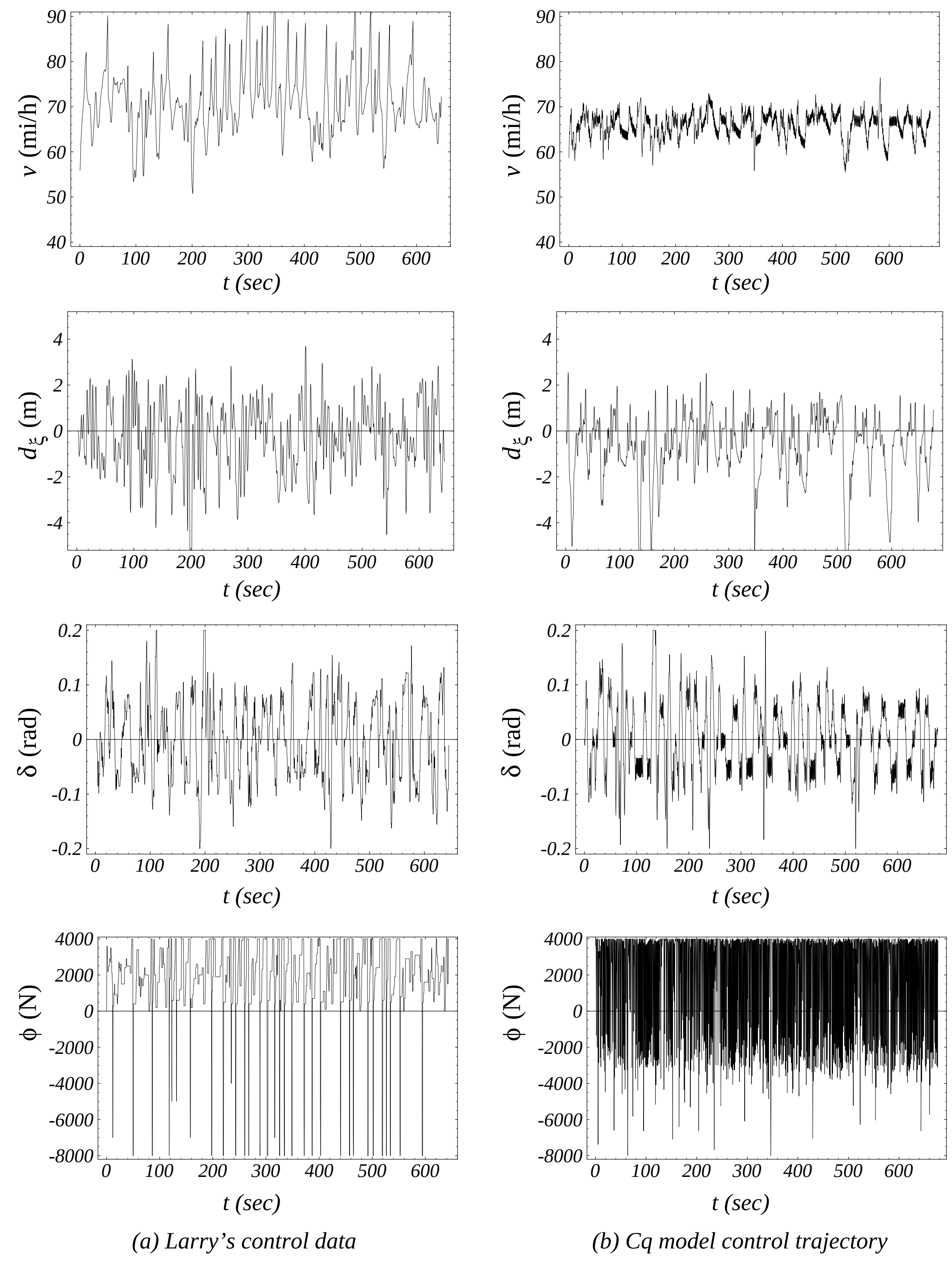
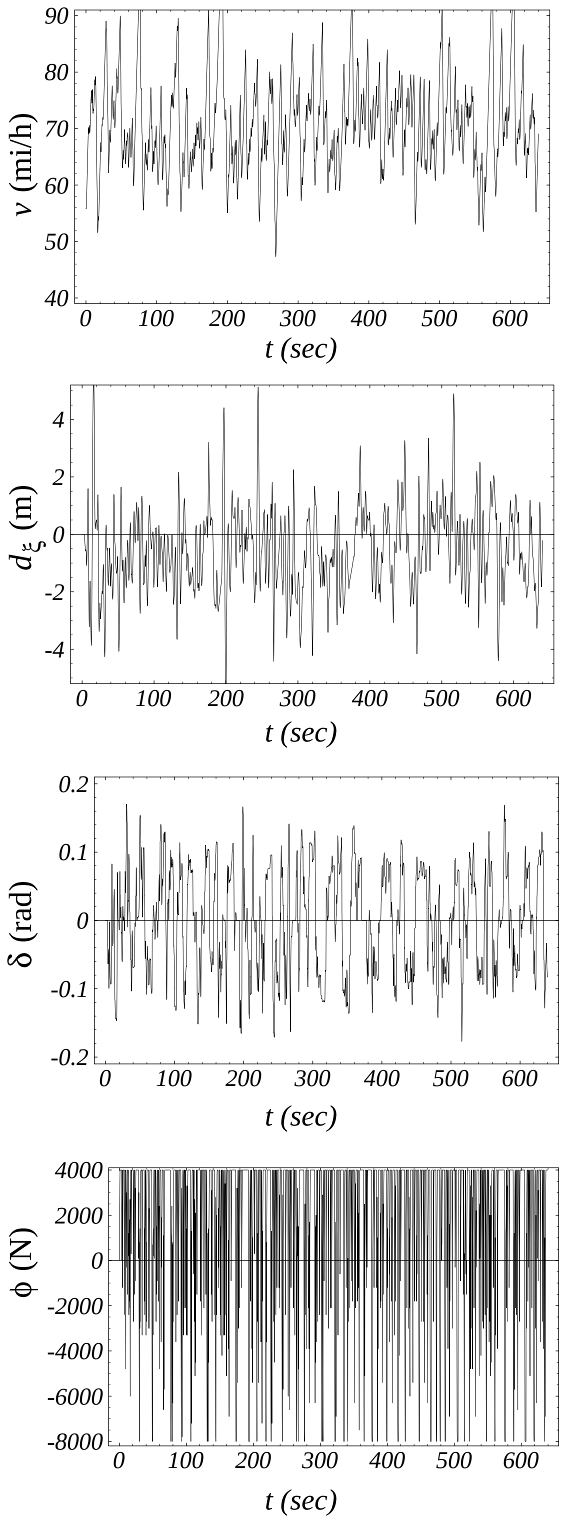
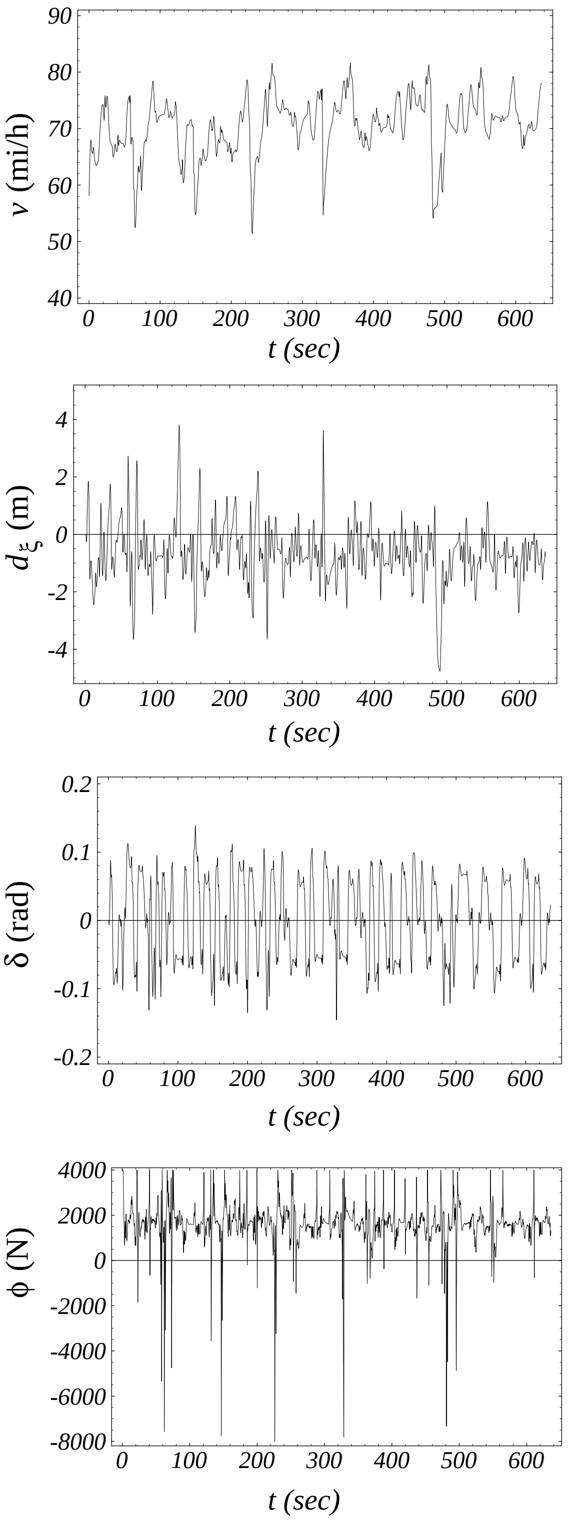


Figure 4-1: Larry's (a) training data and (b) corresponding *Cq* model data.

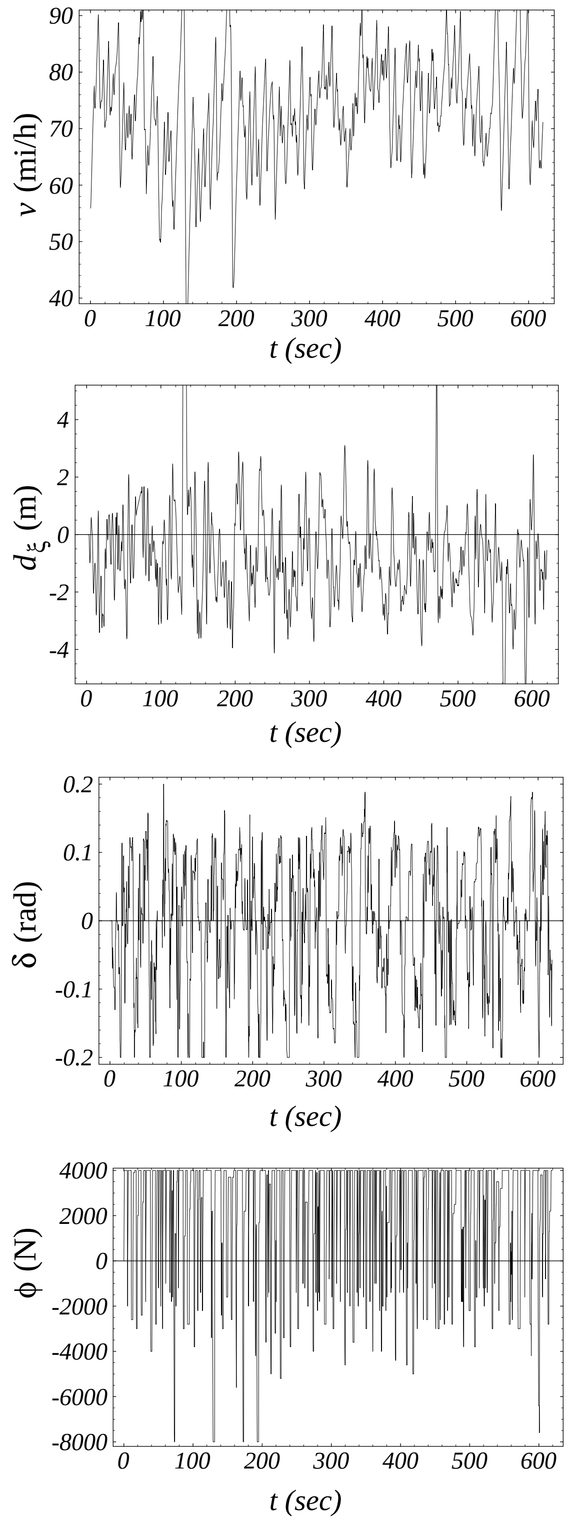


(a) Moe's control data

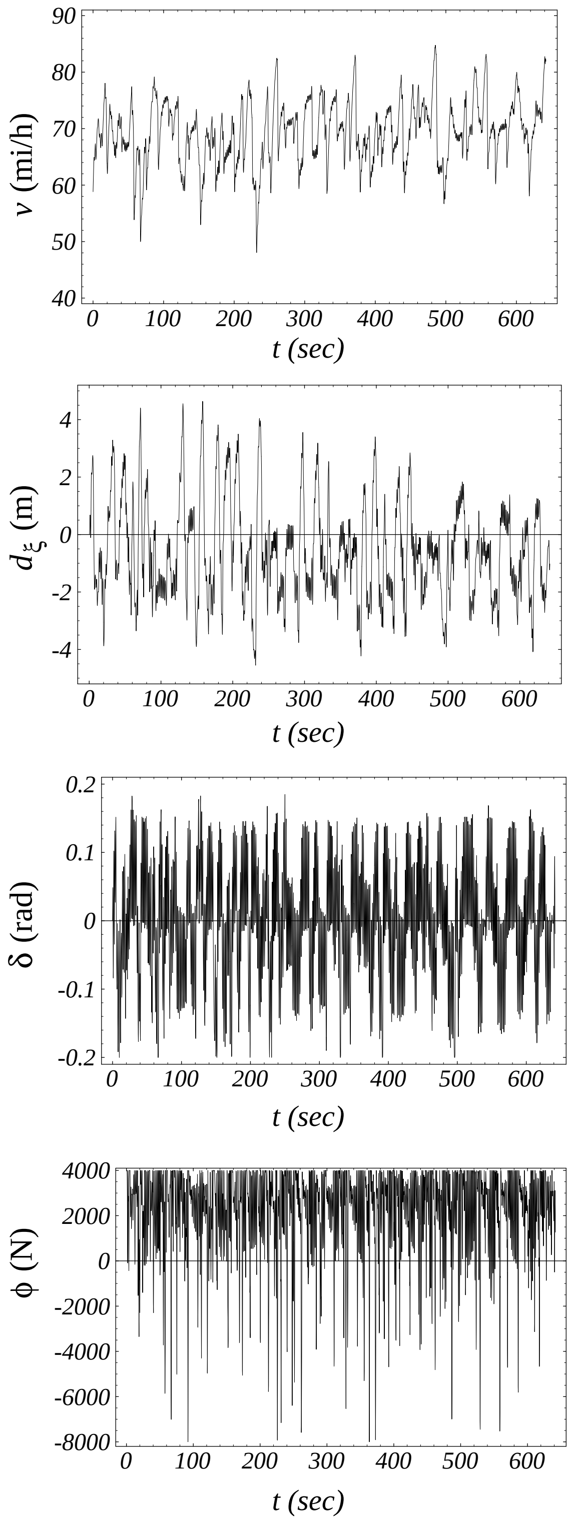


(b) Cq model control trajectory

Figure 4-2: Moe's (a) training data and (b) corresponding *Cq* model data.

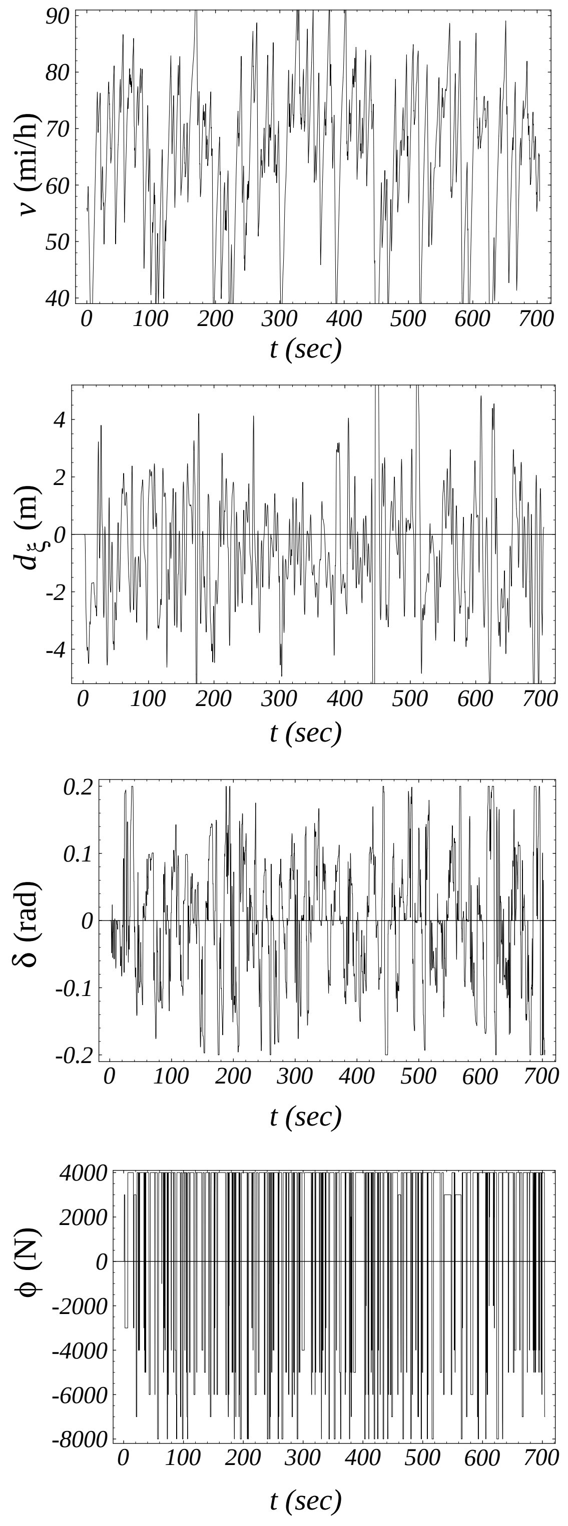


(a) Groucho's control data

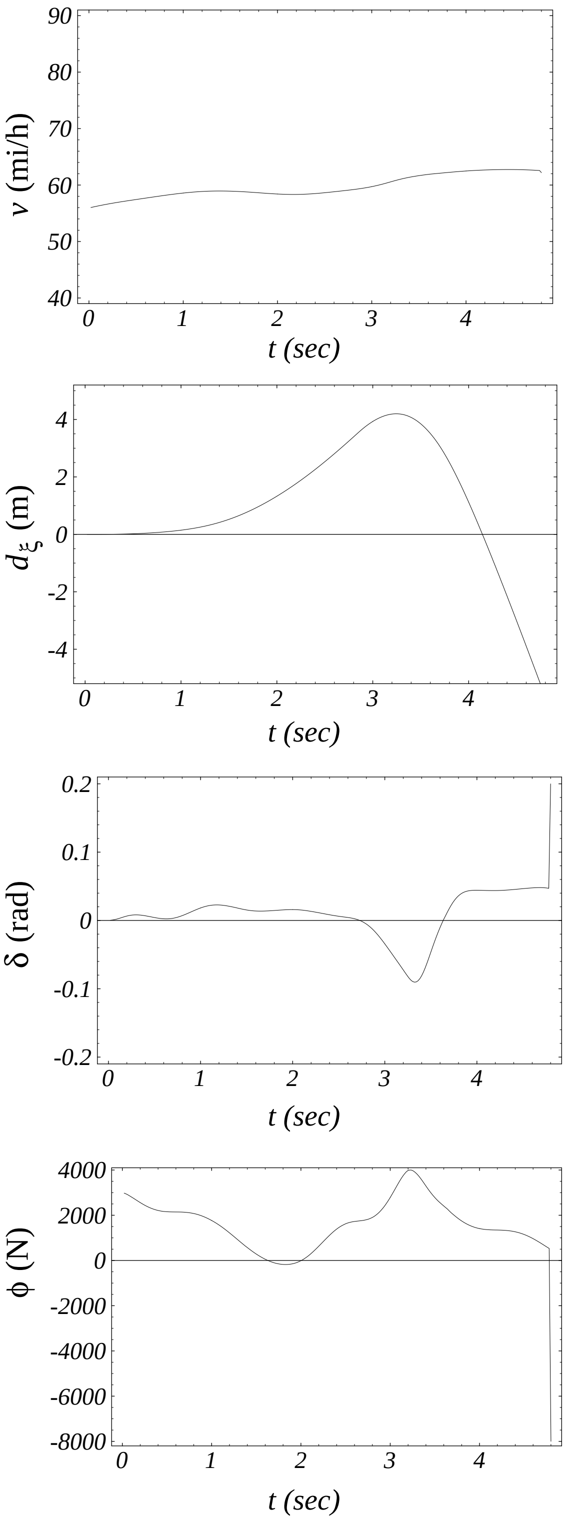


(b) Cq model control trajectory

Figure 4-3: Groucho's (a) training data and (b) corresponding *Cq* model data.



(a) Harpo's control data



(b) Cq model control trajectory

Figure 4-4: Harpo's (a) training data and (b) corresponding *Cq* model data (unstable).

$$\max(d_{\xi, l}^{(i,j)}) < \max(d_{\xi, k}^{(i,j)}), \ \forall k \neq l. \tag{4-3}$$

In other words, we choose the model with the largest stability margin over the validation road. Figure 4-5 plots $\max(d_{\xi, k}^{(i,j)})$ as a function of k for the runs listed in Table 4-1. We observe that for $k \geq 3$, model performance, as measured by the maximum lateral offset, does not change significantly. Thus, when the model is presented with enough time-delayed values of each state and control variable, the model is able to build what appear to be necessary derivative dependencies between model inputs and outputs $\{\delta(k+1), \phi(k+1)\}$.

4.2.2 *Ck* training

For each model $\Gamma_k^{(i,j)}$, we process the training data as described in Section 4.1.3. Training for a particular *Ck* neural network proceeds until the RMS error in the test data set no longer decreases. We use one candidate unit, and allow up to 5 epochs for candidate as well as for output training. After some experimentation, we settled on the following training parameter choices when training on human control data:

$$\eta_Q = 0.0, \ \eta_P = 0.000001 \tag{4-4}$$

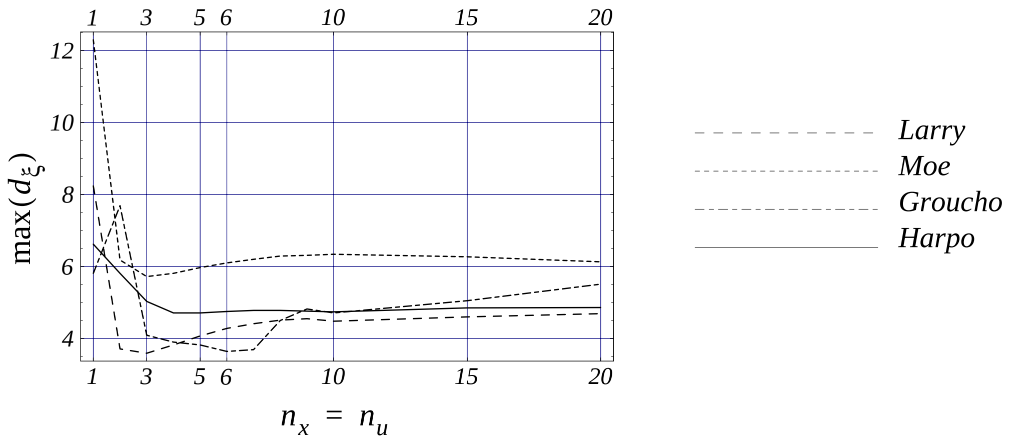


Figure 4-5: Maximum lateral offset d_ξ over validation road #4 as we vary the size of the input space.

4.2.3 HCS models

Figures 4-6, 4-7, 4-8 and 4-9 illustrate representative *Ck* learning results for the same four runs for which we report *Cq* results — namely, $X^{(1,\,2)}$ (Larry's second run), $X^{(3,\,1)}$ (Moe's first run), $X^{(4,\,1)}$ (Groucho's first run), and $X^{(5,\,2)}$ (Harpo's second run). Once again part (a) of each Figure plots the original human control data, while part (b) of each Figure plots the corresponding model control over the test road (#5).

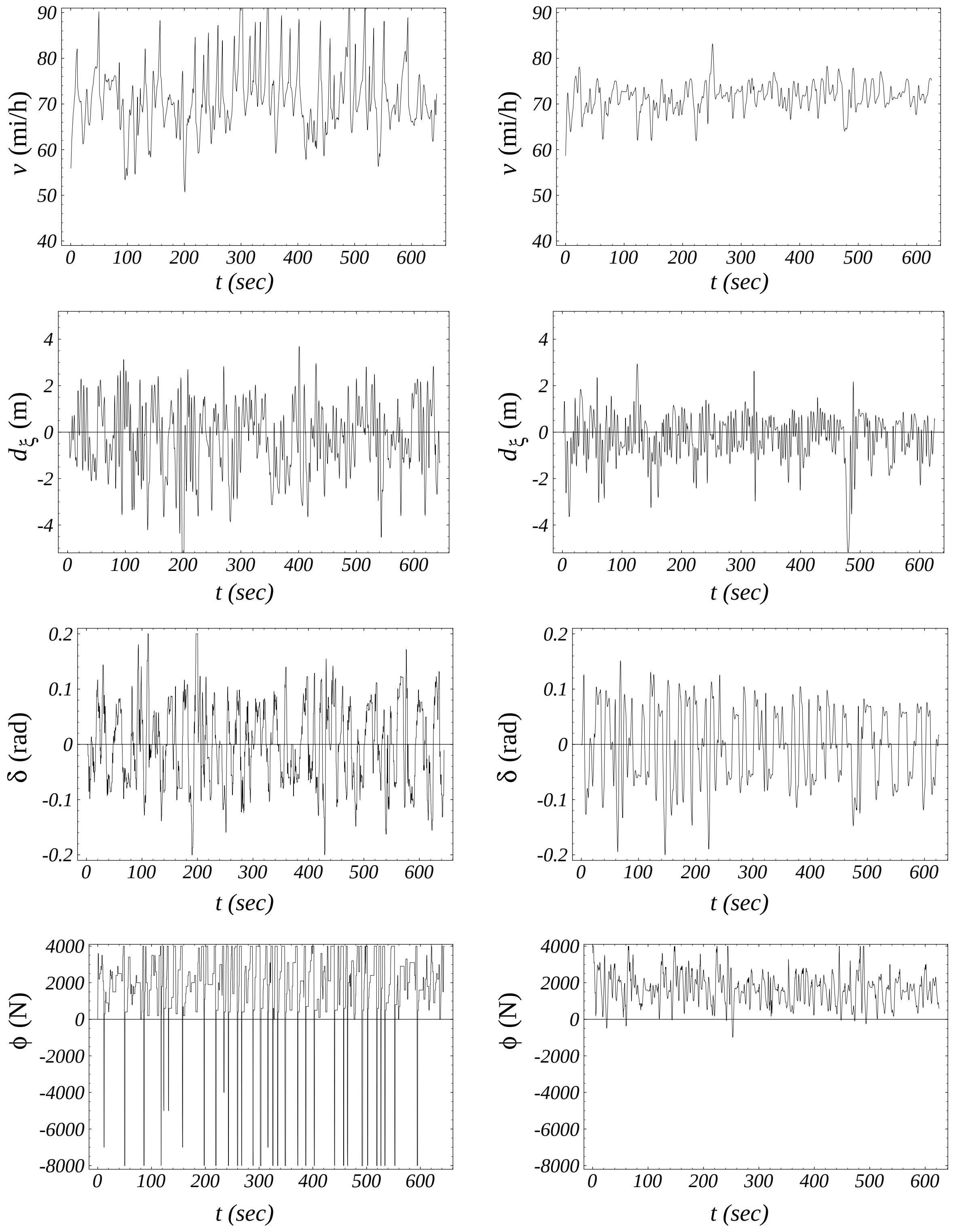
We note that the model control trajectories in Figures 4-6 through 4-9 are for *linear Ck* models; that is, models with no hidden units. Despite the discontinuous acceleration command ϕ, these linear models are able to abstract convergent control strategies — even for Harpo's data — that keep the simulated car on the test road at approximately the same average speed and lateral distance from the road median as the corresponding human controllers. At the same time, we should point out that the *linear Cq* networks do *not* form stable controllers.

Because the *Ck* models are linear, they do not exhibit the type of high-frequency noise that we observed in the nonlinear *Cq* models. Only when we allow the *Ck* models to add nonlinear hidden units, will high-frequency noise manifest itself in the *Ck* models. Figure 4-10, for example, illustrates what happens to Larry's *Ck* model control when one hidden unit (sigmoidal) is added to the linear model. Thus, in general, the linear *Ck* models do not benefit from the addition of nonlinear hidden units. In the next section, we discuss in much greater detail the implications of this result on the stability of the HCS models, the convergence properties of the *Cq* algorithm, the models' fidelity to the source training data, and the capacity of continuous function approximators to model switching control behaviors.

4.3 Analysis

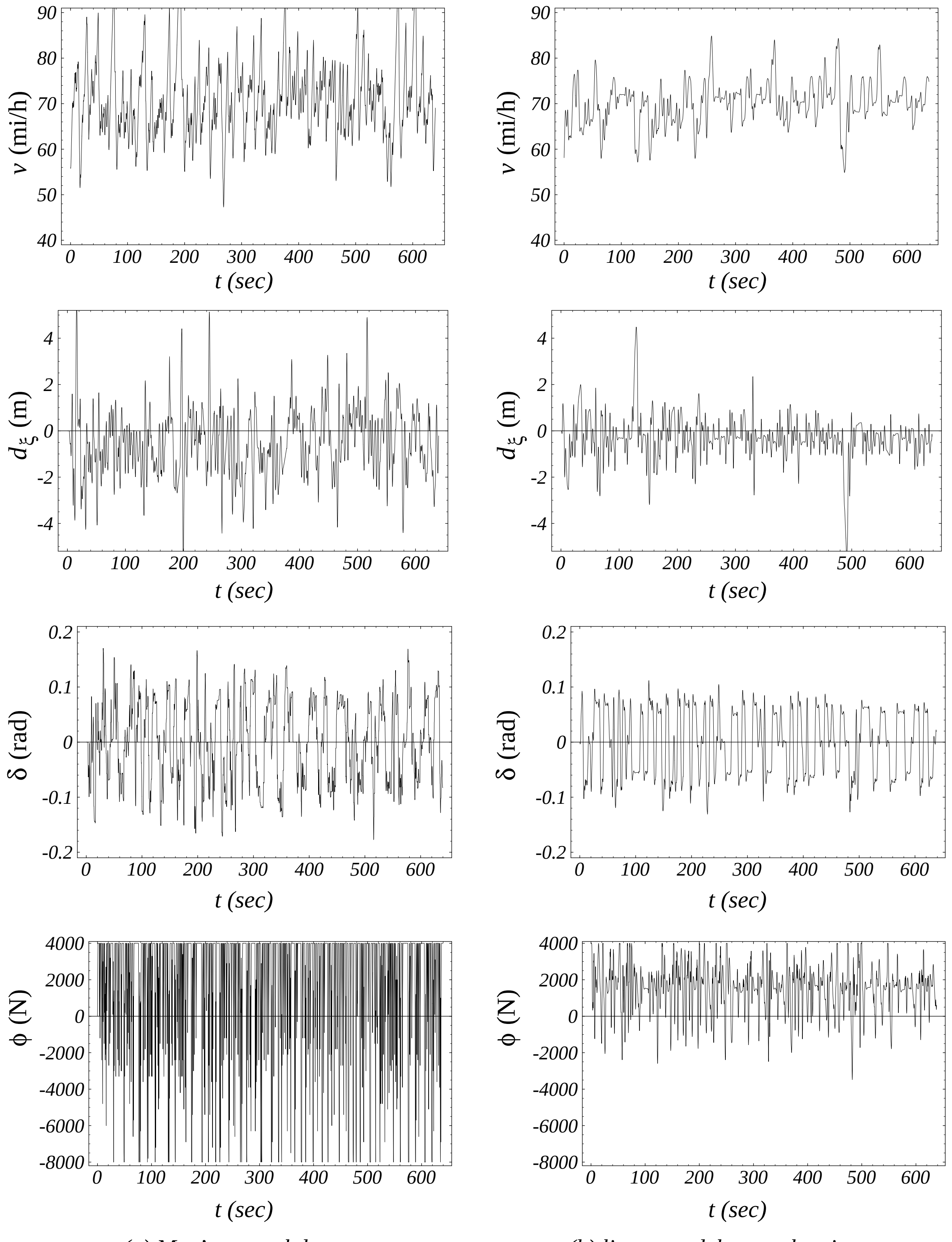
4.3.1 Model stability

In this section, we experimentally assess the stability of our *Ck* and *Cq* HCS models. While we have already seen one example of instability (Harpo's *Cq* model), we would like to determine



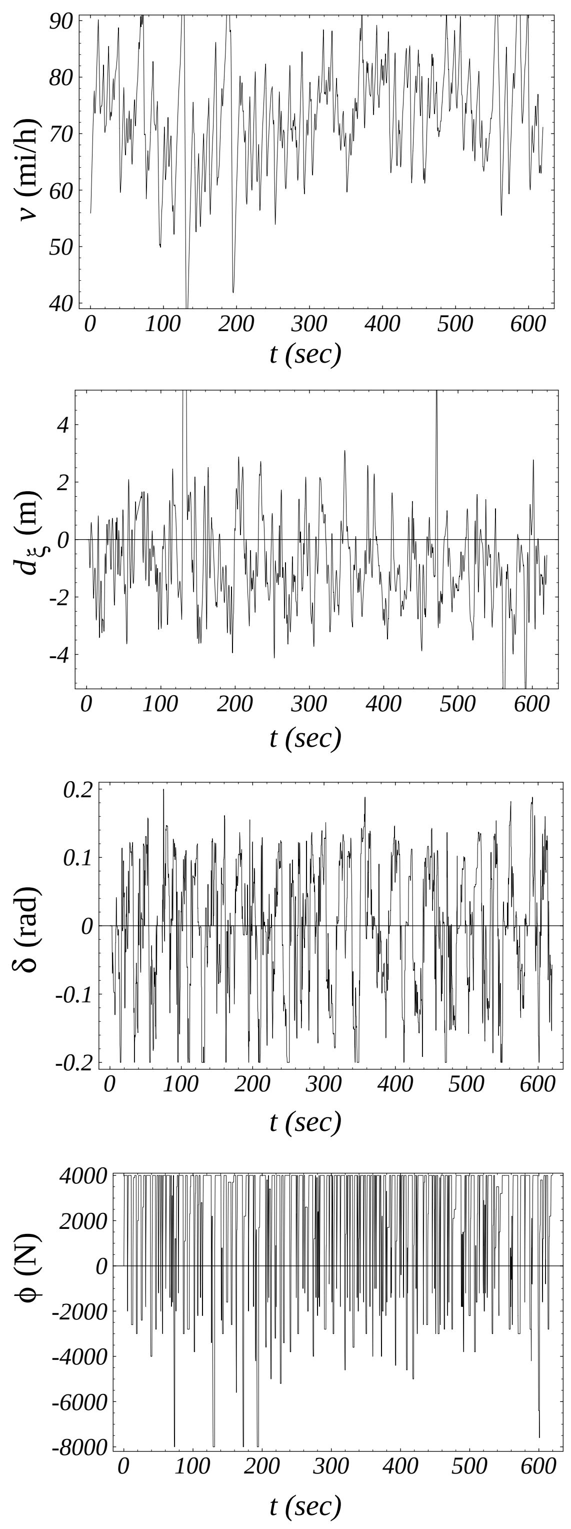
(a) Larry's control data *(b) linear model control trajectory*

Figure 4-6: Larry's (a) training data and (b) corresponding linear model data.

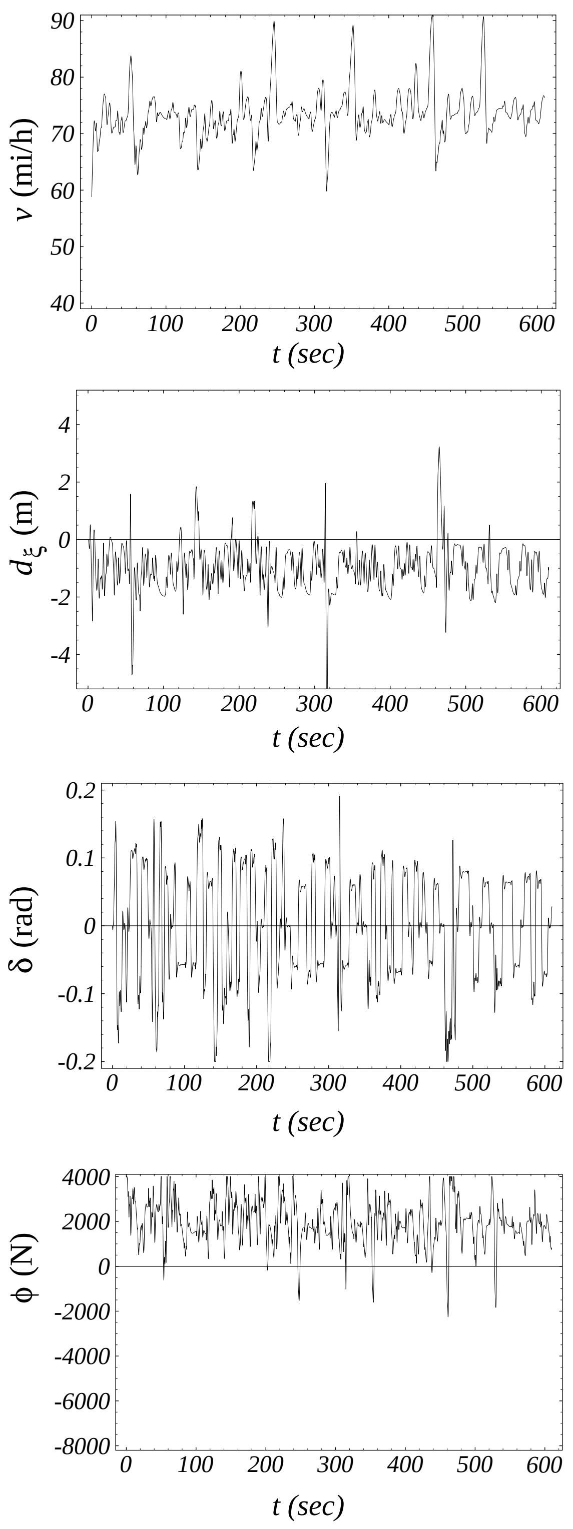


(a) Moe's control data *(b) linear model control trajectory*

Figure 4-7: Moe's (a) training data and (b) corresponding linear model data.

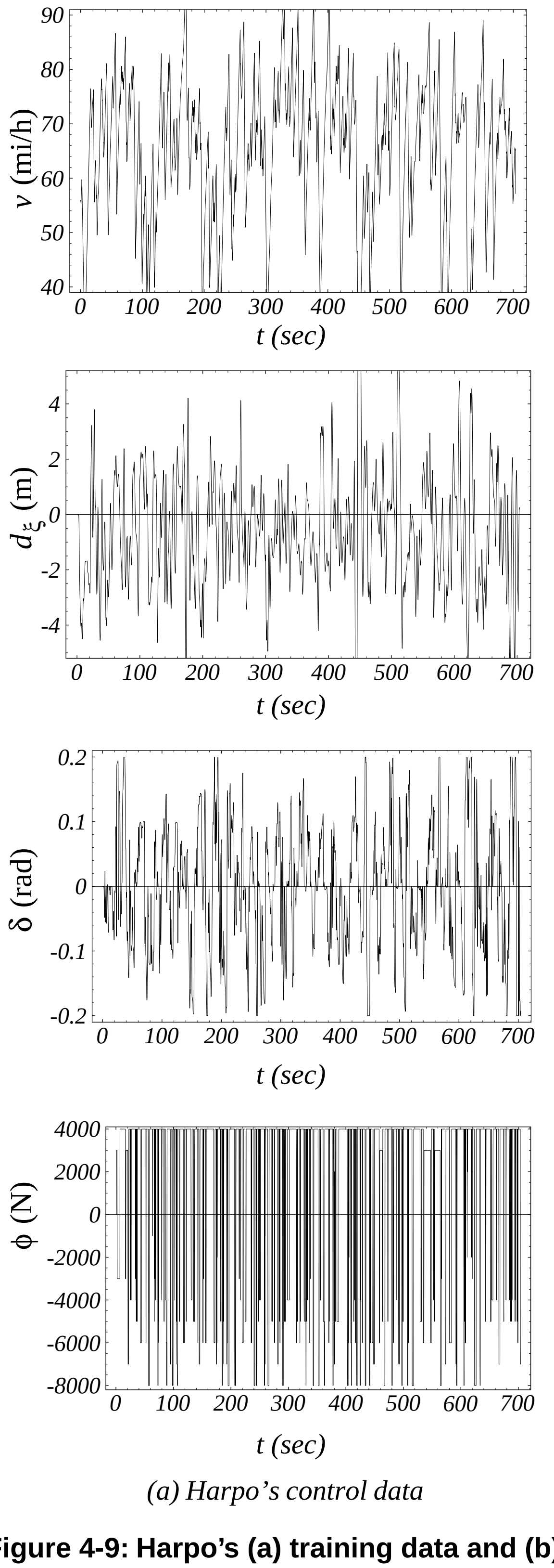


(a) Groucho's control data

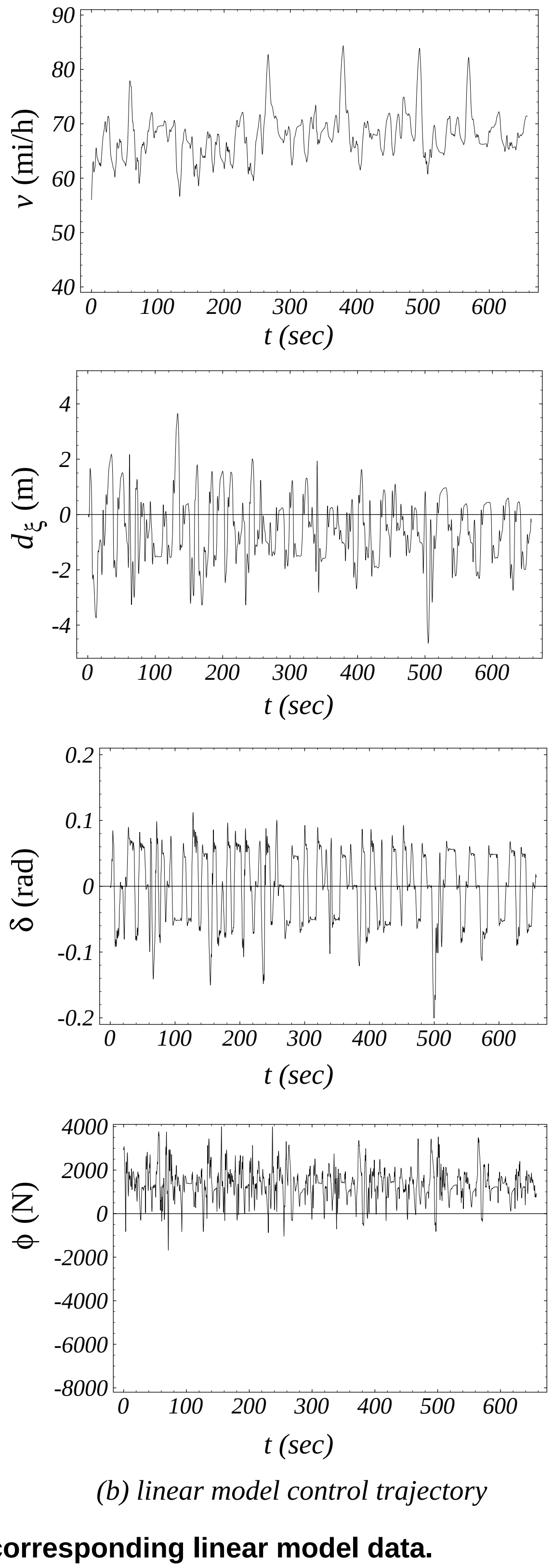


(b) linear model control trajectory

Figure 4-8: Groucho's (a) training data and (b) corresponding linear model data.



(a) Harpo's control data



(b) linear model control trajectory

Figure 4-9: Harpo's (a) training data and (b) corresponding linear model data.

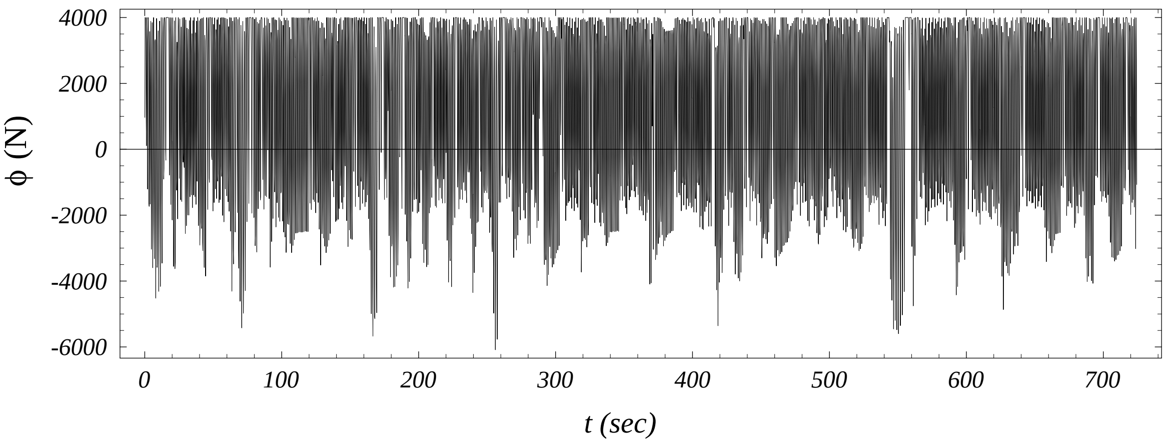


Figure 4-10: Larry's *Ck* model control with one hidden unit added. Note that the additional hidden unit causes significant (not really beneficial) high-frequency noise.

for what range of initial conditions and road curvatures each model is stable (i.e. stays on the road). To do this for a given HCS model, we record the maximum lateral offset for that model ($\max(d_\xi)$) as it attempts to negotiate the s-curve shown in Figure 4-11 for different radii r and initial velocities v_{init}. Figures 4-12, 4-13, 4-14 and 4-15 plot these stability profiles for Larry, Moe, Groucho and Harpo[2], respectively, where $90\text{m} \le r \le 250\text{m}$ and $20\text{mi/h} \le v_{init} \le 100\text{mi/h}$.

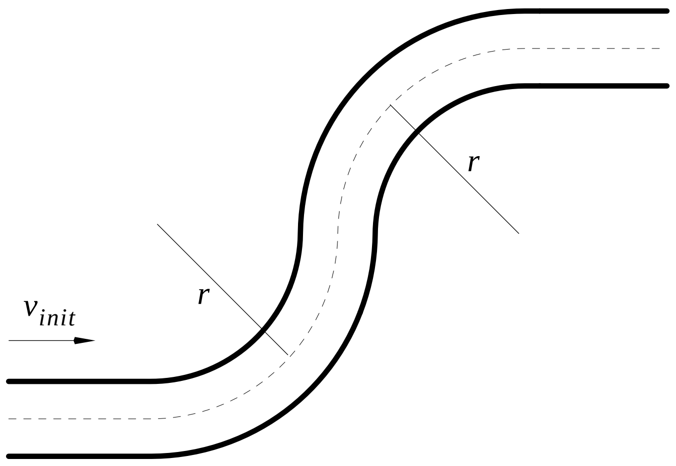


Figure 4-11: This s-curve test road is used to generate stability profiles for the human control strategy models, as v_{init} and r are varied.

2. From Harpo's stability profile, it becomes apparent why his *Cq* model fails on road #5. Road #5 begins with an s-curve whose radii are 117m and 123m, respectively.

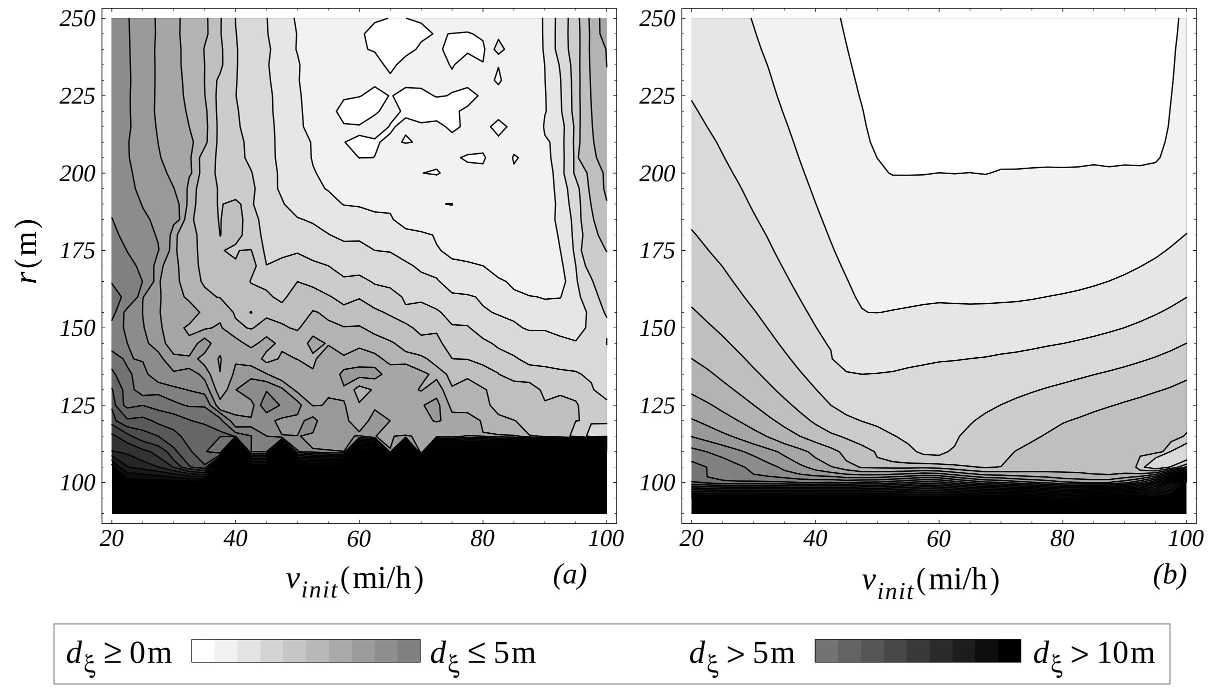


Figure 4-12: Larry's stability profiles through the s-curve for (a) the *Cq* model and (b) the *Ck* model (lighter colors are better).

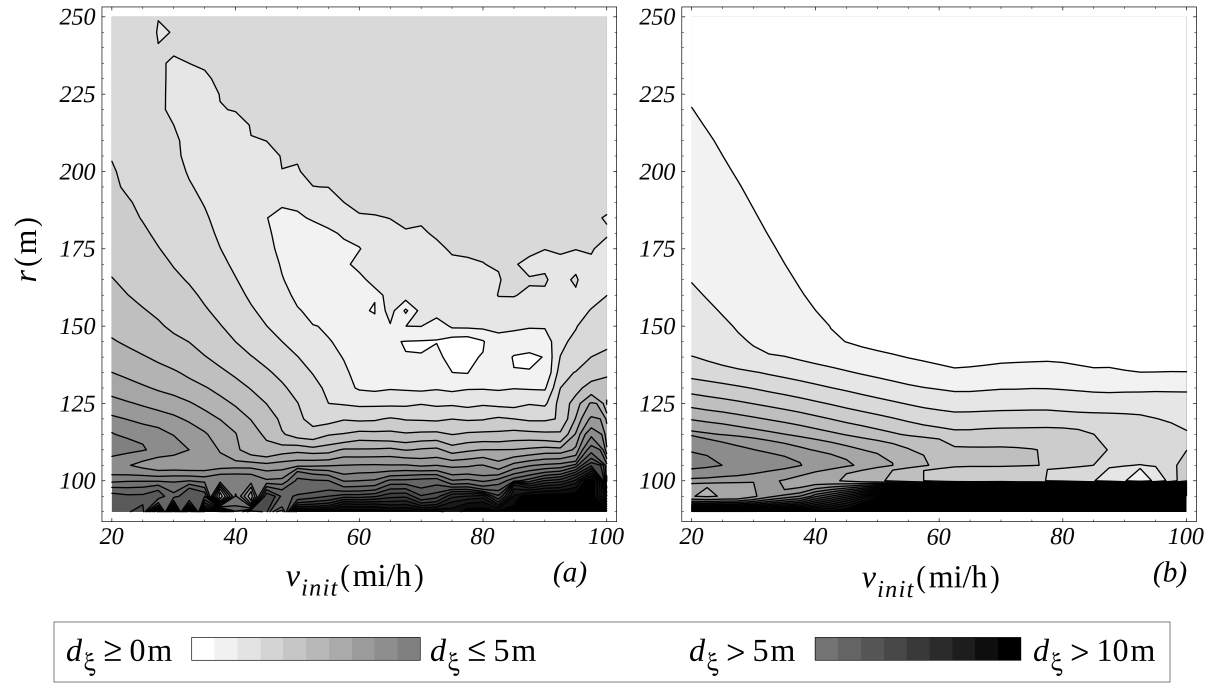


Figure 4-13: Moe's stability profiles through the s-curve for (a) the *Cq* model and (b) the *Ck* model (lighter colors are better).

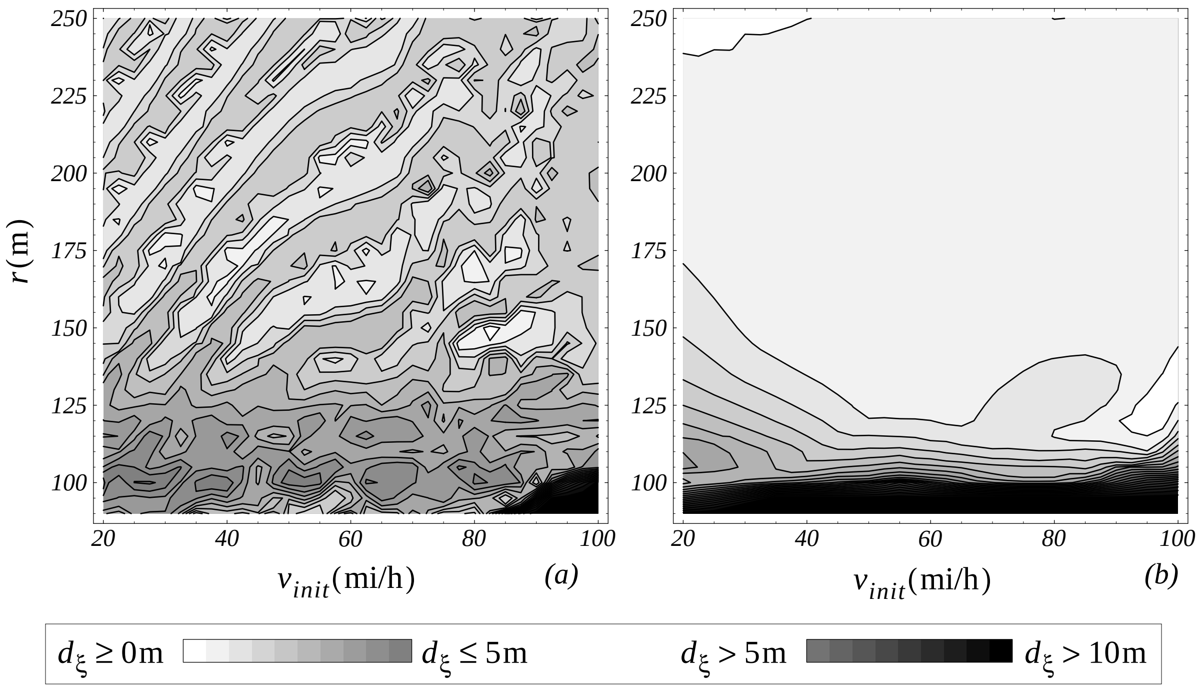


Figure 4-14: Groucho's stability profiles through the s-curve for (a) the *Cq* model and (b) the *Ck* model (lighter colors are better).

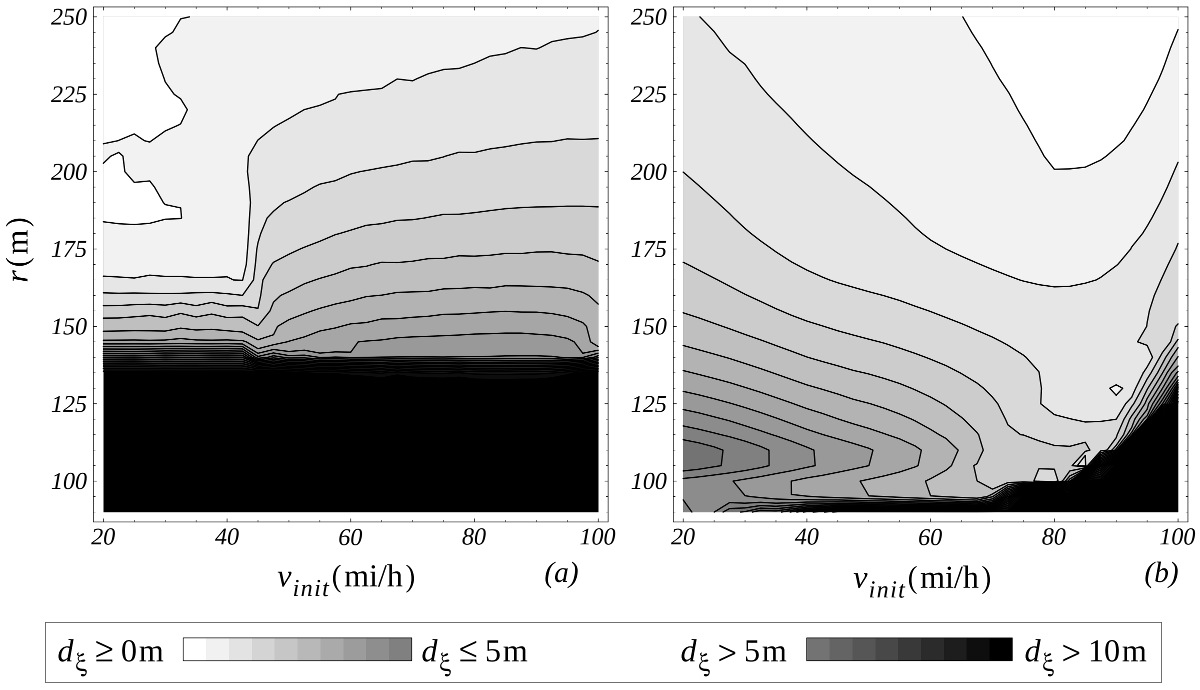


Figure 4-15: Harpo's stability profiles through the s-curve for (a) the *Cq* model and (b) the *Ck* model (lighter colors are better).

From these Figures, we observe that, in general, the *Ck* models behave in a stable manner for a greater range of initial and environmental conditions than do the *Cq* models. Moreover, the control behaviors of the *Ck* models vary more smoothly with changes in r and v_{init}. Thus, the *Cq* models, with their many additional hidden units, do not appear to learn anything beneficial with the increased nonlinearity of the larger models.

4.3.2 Learning convergence

Now, we examine the difference in learning convergence between the *Cq* and *Ck* learning architectures. As we have noted previously, the linear *Ck* networks converge in less than one epoch to approximately the same RMS error as the *Cq* networks after thousands of epochs and multiple hidden units. Consider Figures 4-16 and 4-17, for example. In Figure 4-16, we show how the RMS error over Groucho's entire training and test data sets decreases in the first epoch of the *Ck* algorithm. Even though the entire training set consists of approximately 15,000 input-output patterns, the *Ck* algorithm converges very close to the final RMS error after only 1/3 of the training data set is presented *once*. By contrast, Figure 4-17 illustrates the convergence of the *Cq* algorithm for Groucho's data. Note that its linear convergence as measured by the RMS error is substantially worse than *Ck*'s linear convergence, and that *Cq* requires about 12 hidden units and 11,000 epochs before converging to an equivalent test RMS error. This is true despite repeated attempts to optimize the learning parameters for the *Cq* algorithm.

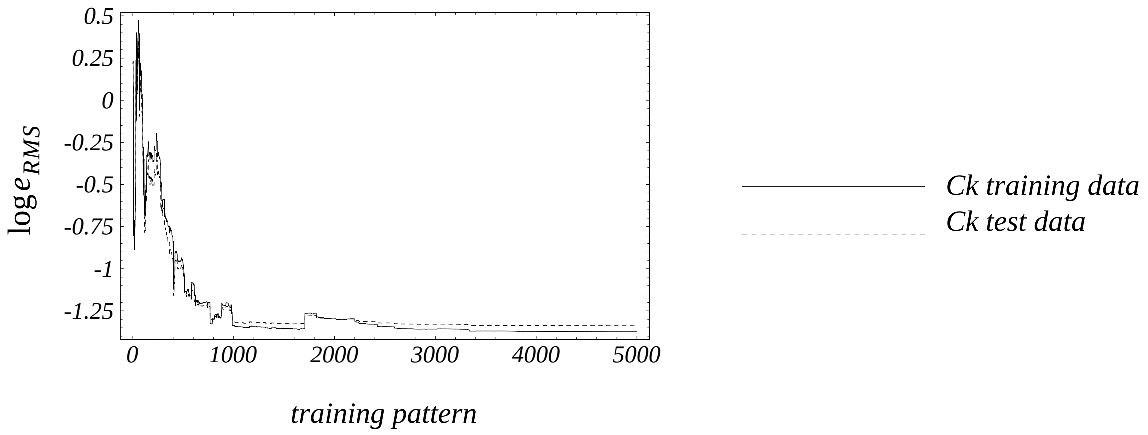


Figure 4-16: Error convergence per *training pattern* for Groucho's linear *Ck* model.

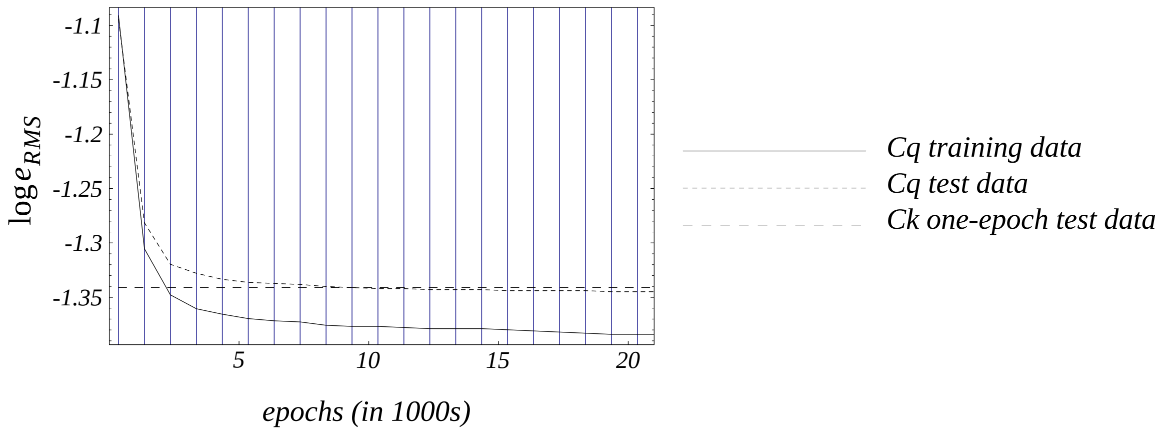


Figure 4-17: Error convergence for Groucho's *Cq* model. Vertical lines indicate the addition of a new hidden unit.

It is apparent that the *Cq* algorithm encounters significant convergence problems when presented with the correlated, time-delayed inputs of our HCS models. This becomes even more apparent if we increase n_x and n_u beyond the values in Table 4-1. As an example, consider Harpo's model. Figure 4-18 plots the stability profile for the *Ck* model, where $n_x = n_u = 20$. Although the dimensionality of the input space is increased from 45 inputs ($n_x = n_u = 5$, $n_r = 10$) to 120 inputs, the *Ck* algorithm preserves the stable control strategy of the original model. The *Cq* algorithm on the other hand, does not converge to a stable control strategy for any values $r \subset [90\text{m}, 250\text{m}]$, $v_{init} \subset [20\text{mi/h}, 100\text{mi/h}]$.

Thus, although the reduction of e_{RMS} as hidden units are added to the *Cq* models suggests that substantial *nonlinear* modeling is occurring, this is not the case, since virtually *all* the reduction in e_{RMS} can be captured by a *linear Ck* model. In fact, any training algorithm that does not explicitly factor in the interdependence of weights in the neural network model is doomed to a similar fate, due to the correlated nature of the time-delayed state and control inputs, as well as the correlation between visible road coordinates.

4.3.3 Discussion

While thus far we have argued that the *Ck* algorithm shows better convergence to stabler HCS models, we have not yet addressed how closely each of the learned models approximates its

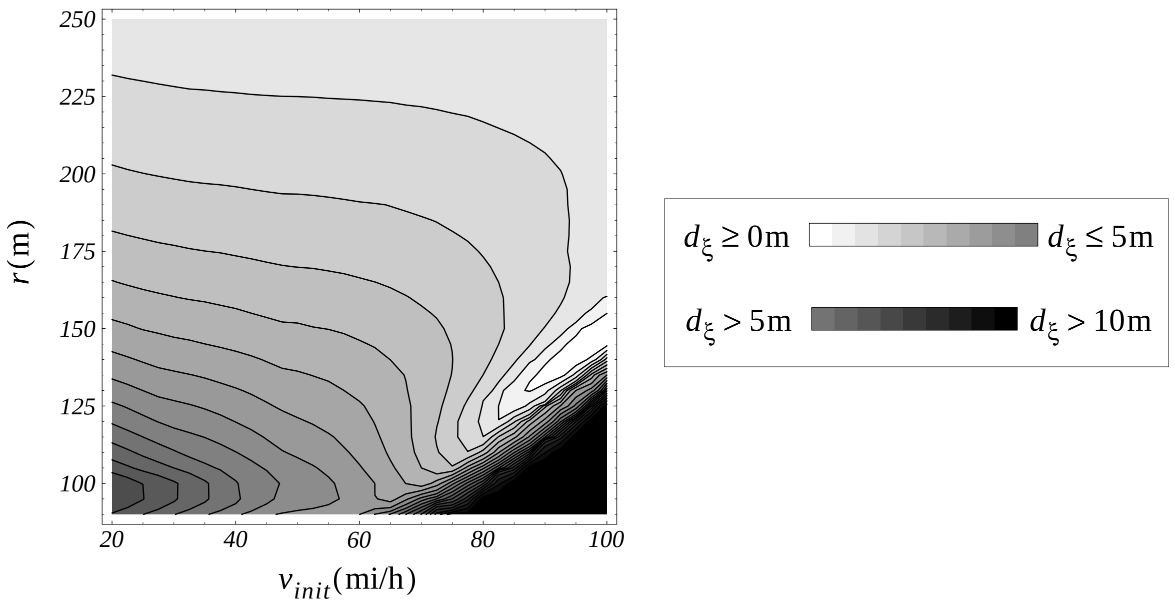


Figure 4-18: Harpo's stability profiles through the s-curve for the *Ck* model with $n_x = n_u = 20$. The *Cq* model is unstable for all r, v_{init}.

corresponding training data. Examining Figures 4-1 through 4-4 and Figures 4-6 through 4-9 we make the *qualitative* observation that none of the models' control strategies closely mirror those of the corresponding human data.[3] Neither the *Cq* nor the *Ck* learning algorithm appears to be able to model the driving control strategies with a high degree of fidelity to the source training data.

The principal source of this inability appears to be the discontinuous switching of the acceleration control ϕ. To better appreciate why this is the case, we would like to visualize how different input vectors in the training data map to different acceleration outputs $\phi(k+1)$. As an example, consider Groucho's control strategy data and let $n_x = n_u = 6$, $n_r = 10$ as before. For these input space parameters, the input training vectors $\zeta(k)$ are of length 50. Since it is impossible to visualize a 50-dimensional input space, we decompose each of the input vectors $\zeta(k)$ in the training set into the principal components (PCs) [91] over Groucho's entire data set, such that,

3. We will formalize this qualitative observation in Chapter 8.

$$\zeta(k) = c_1^k\gamma_1 + c_2^k\gamma_2 + \dots c_{50}^k\gamma_{50}, \tag{4-5}$$

where γ_i is the principal component corresponding to the *i*th largest eigenvalue σ_i. Now, for Groucho's control data we have that,

$$|\sigma_2/\sigma_1| = 0.44, |\sigma_i/\sigma_1| \leq 0.05, i \in \{3, 4, \dots, 50\}, \tag{4-6}$$

so that we coarsely approximate the input vectors $\zeta(k)$ as,

$$\zeta(k) \approx c_1^k\gamma_1 + c_2^k\gamma_2. \tag{4-7}$$

By plotting the PC coefficients (c_1^k, c_2^k) in 2D space, we can now visualize the approximate relative location of the input vectors $\zeta(k)$. Figure 4-19(a) and (b) show the results for $\phi(k) < 0$ (brake), and $\phi(k) \geq 0$ (gas), respectively. In each plot, we distinguish points by whether or not $\phi(k+1)$ indicates a discontinuity (i.e. a switch between braking and accelerating) such that,

$$\phi(k) < 0 \text{ and } \phi(k+1) > 0 \text{ [Figure 4-19(a)]} \tag{4-8}$$

$$\phi(k) > 0 \text{ and } \phi(k+1) < 0 \text{ [Figure 4-19(b)]} \tag{4-9}$$

Those points that involve a switch are plotted in black, while a representative sample (20%) of the remaining points are plotted in grey.

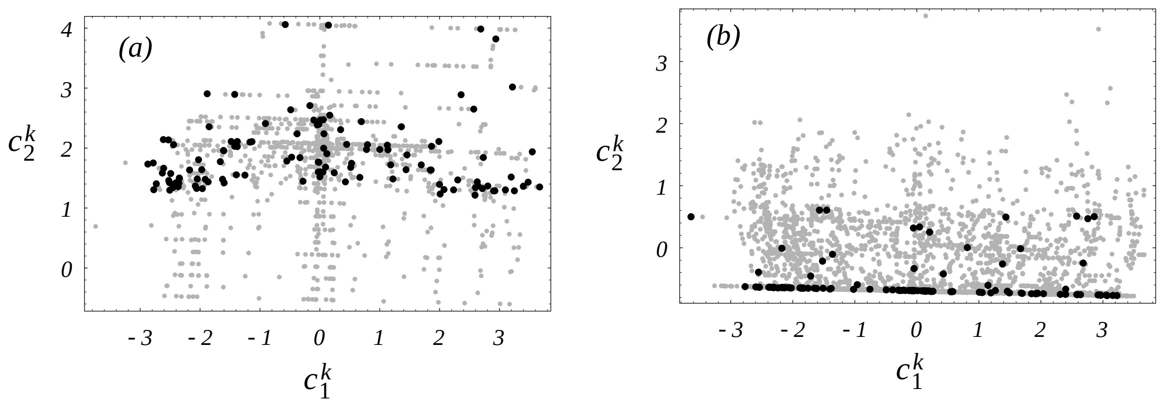


Figure 4-19: Switching actions (black) significantly overlap other actions (grey) when the current applied force is (a) negative (brake), and (b) positive (gas).

We immediately observe from Figure 4-19 that — at least in the low-dimensional projection of the input vectors — the few training vectors that involve a switch overlap the many other vectors that do not. In other words, very similar inputs $\zeta(k)$ can lead to radically different outputs $\phi(k+1)$. Consequently, Groucho's acceleration control strategy may not be easily expressible in a functional form, let alone a smooth functional form. This poses an impossible learning challenge not just for cascade neural networks, but *any* continuous function approximator. In theory, no continuous function approximator will be capable of modeling the switching of the acceleration control ϕ (Figure 4-20).

In summary, we note that while we limit the volume of results presented to representative examples, the conclusions we draw with regards to the *Cq* and *Ck* algorithms have been confirmed for other human control data sets and over countless and repeated learning trials. Thus, this chapter has shown that (1) the *Ck* algorithm converges faster and more reliably than *Cq* in modeling human control strategies; (2) the *Ck* models exhibit stability over a greater range of initial and environmental conditions; (3) as long as sufficient data is provided as input, the precise input representation affects performance only marginally; and (4) the continuous *Ck* and *Cq* algorithms abstract control strategies that are qualitatively dissimilar to the original human control strategies. In Chapter 5, we derive an alternative, discontinuous modeling framework which attempts to overcome this limitation.

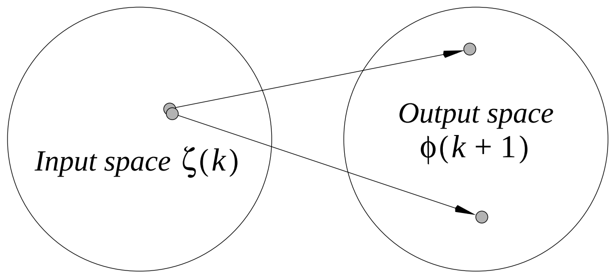


Figure 4-20: Switching causes very similar inputs to be mapped to radically different outputs.

Chapter 5

HCS Models: Discontinuous Learning

In the previous chapter, we investigated the capacity of cascade neural networks to abstract models of human control strategy, by mapping environmental inputs and time-delayed histories of state and previous control variables to control action outputs. As we observed, the resulting models, while abstracting convergent, stable control strategies, do not appear to exhibit a high degree of fidelity to the source human training data. This is in no small measure due to the acceleration command ϕ. While the steering control δ tends to vary more continuously with model inputs, the acceleration control ϕ involves discrete switching decisions between the gas and brake pedals that introduce discontinuities in the input-output mapping. Consequently very similar input spaces are mapped to radically different outputs.

To adequately model such control behavior, in this chapter we propose a stochastic, discontinuous learning algorithm. The proposed algorithm models possible control actions as individual statistical models. During run-time execution of the algorithm, a control action is then selected stochastically, as a function of both prior probabilities and posterior evaluation probabilities. We show that the resulting controller overcomes the shortcomings of continuous modeling approaches in modeling discontinuous control strategies, and that the resulting model strategies appear to exhibit a higher degree of fidelity to the human training data.

5.1 Hybrid continuous/discontinuous control

Figure 5-1 provides an overview of the proposed modeling approach. As before, we use a *Ck* model for the steering control δ. Now, however, we model the acceleration control ϕ in a discontinuous, statistical framework. The following sections describe this framework in much greater detail.

5.1.1 General statistical framework

First, let us derive a general statistical framework for faithfully modeling discontinuous control strategies. For now, we make the following assumptions. First, assume a control task where at each time step k, there is a choice of one of N different control actions A_i, $i \in \{1, \dots, N\}$. Second, assume that we have sets of input vector training examples $\{\bar{\zeta}_i^j\}$, $j \in \{1, 2, \dots, n_i\}$, where each $\bar{\zeta}_i^j$ leads to control action A_i at the next time step. Finally, assume that we can train statistical models λ_i, which maximize,

$$\prod_{j=1}^{n_i} P(\lambda_i | \bar{\zeta}_i^j), \; i \in \{1, \dots, N\}. \tag{5-1}$$

Given an unknown input vector ζ^*, we would like to choose an appropriate, corresponding control action A^*. Since model λ_i corresponds to action A_i, we define,

$$P(\bar{\zeta}^* | A_i) \equiv P(\bar{\zeta}^* | \lambda_i). \tag{5-2}$$

By Bayes Rule,

$$P(A_i | \bar{\zeta}^*) = \frac{P(\bar{\zeta}^* | A_i) P(A_i)}{P(\bar{\zeta}^*)}, \tag{5-3}$$

where,

$$P(\bar{\zeta}^*) \equiv \sum_{i=1}^{N} P(\bar{\zeta}^* | A_i) P(A_i), \tag{5-4}$$

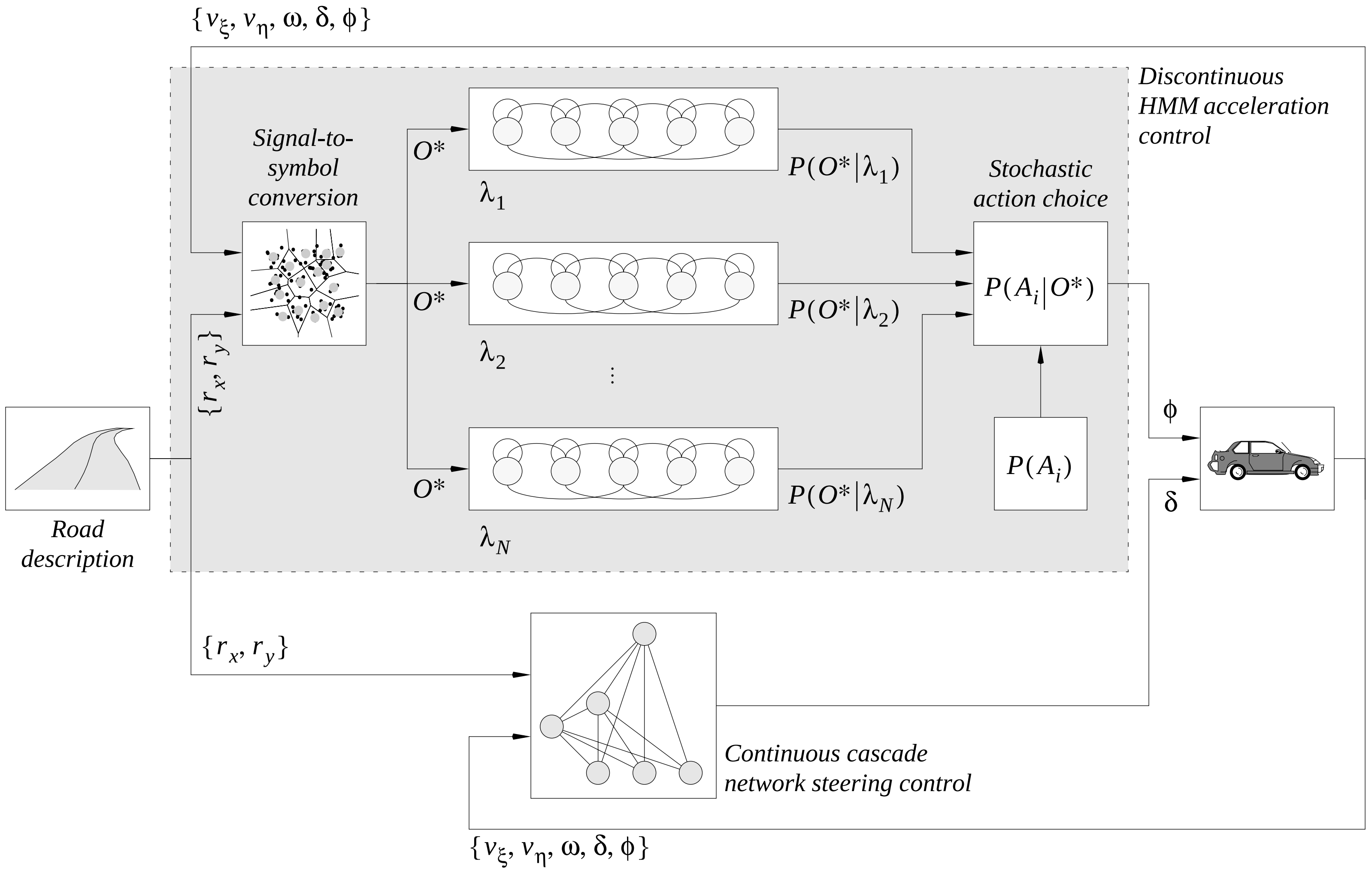


Figure 5-1: Overall hybrid discontinuous/continuous controller. Steering is controlled as before by a cascade model, while the discontinuous acceleration command is controlled by the HMM-based, stochastic framework (shaded box).

serves as a normalization factor, $P(A_i)$ represents the *prior* probability of selecting action A_i and $P(A_i|\zeta^*)$ represents the *posterior* probability of selecting action A_i given in the input vector ζ^*.

We now propose the following stochastic control strategy for A^*. Let,

$$A^* = A_i \text{ with probability } P(A_i|\bar{\zeta}^*), \tag{5-5}$$

so that, at each time step k, the control action A^* is generated stochastically as a function of the current model inputs ($\bar{\zeta}^*$) and the prior likelihood of each action.

5.1.2 Action definitions

As we point out in equations (2-20) and (2-21), the acceleration command ϕ is limited at each time step k to the following actions. When $\phi(k) \geq 0$ (the gas is currently active),

$$A_1: \phi(k+1) = \phi(k) \tag{5-6}$$

$$A_2: \phi(k+1) = \min(\phi(k) + \Delta\phi_g, 4000), \tag{5-7}$$

$$A_3: \phi(k+1) = \max(\phi(k) - \Delta\phi_g, 0), \tag{5-8}$$

$$A_4: \phi(k+1) = -\Delta\phi_b, \tag{5-9}$$

and when $\phi(k) < 0$ (the brake is currently active),

$$A_5: \phi(k+1) = \phi(k) \tag{5-10}$$

$$A_6: \phi(k+1) = \max(\phi(k) - \Delta\phi_b, -8000), \tag{5-11}$$

$$A_7: \phi(k+1) = \min(\phi(k) + \Delta\phi_b, 0), \tag{5-12}$$

$$A_8: \phi(k+1) = \Delta\phi_g, \tag{5-13}$$

Actions A_1 and A_5 correspond to no action for the next time step; actions A_2 and A_6 correspond to pressing harder on the currently active pedal; actions A_3 and A_7 correspond to easing off the currently active pedal; and actions A_4 and A_8 correspond to switching between the gas and brake pedals. The constants $\Delta\phi_g$ and $\Delta\phi_b$ are set by each human operator to the pedal responsiveness level he or she desires. Table 5-1 below lists those choices for our four test individuals.

Table 5-1: Pedal responsiveness choices

Individual	$\Delta\phi_g$ *(N)*	$\Delta\phi_b$ *(N)*
Larry	100	1000
Moe	100	300
Groucho	100	200
Harpo	1000	1000

5.1.3 Statistical model choice

In part because of our familiarity with Hidden Markov Models (see Section 6.2.1), and because of the capacity of HMMs to model arbitrary statistical distributions, we choose discrete-output HMMs to be the trainable statistical models λ_i of Section 5.1.1. Consequently, we must convert the multi-dimensional, real-valued model input space to discrete symbols.

For a particular data set X let,

$$\bar{\zeta}(k) = \left[v_{\xi}^{n_x}\ v_{\eta}^{n_x}\ \omega^{n_x}\ \delta^{n_u}\ \phi^{n_u}\ r_x^{n_r}\ r_y^{n_r}\right]^T, \tag{5-14}$$

denote the normalized model input vector at time step k corresponding to control action $\phi(k+1)$, where $\{v_{\xi}^{n_x}, v_{\eta}^{n_x}, \omega^{n_x}, \delta^{n_u}, \phi^{n_u}, r_x^{n_r}, r_y^{n_r}\}$ are defined in equation (2-26). Also, let V be a matrix, whose rows are the $\bar{\zeta}(k)$ vectors. Using the LBG VQ algorithm [69], we generate a codebook Q_L of size L that minimizes the quantization distortion $D(V, Q_L)$ defined in equation (6-67). To complete the signal-to-symbol conversion, the discrete symbols $o(k)$ corresponding to the minimum distortion of $\bar{\zeta}(k)$ are given by,

$$o(k) = T^v_{VQ}(\bar{\zeta}(k), Q_L), \tag{5-15}$$

where $T^v_{VQ}(\cdot)$ is defined in equation (6-69). Finally, let us define the observation sequence $O(k)$ of length n_O to be,

$$O(k) = \{o(k - n_O + 1), o(k - n_O + 2), \ldots, o(k)\}. \tag{5-16}$$

Now, suppose that we want to provide the Hidden Markov Models λ_i with m time-delayed values of the state and control variables as input. There are at least two ways to achieve this. We can either (1) let $n_x = n_u = m$, $n_O = 1$, or (2) let $n_x = n_u = 1$, $n_O = m$. In the first case, we vector quantize the entire input vector into a single observable, and base our action choice on that single observable. This necessarily forces the HMMs λ_i to single-state models, such that each model is completely described by its corresponding output probability vector B_i. Alternatively, we can vector quantize shorter input vectors but provide a longer sequence of observables $n_O > 1$ for HMM training and evaluation.

While in theory both choices start from identical input spaces, the single-observable, single-state case works better in practice. There are two primary reasons for this. Because the amount of data we have available for training comes from finite-length data sets, and is therefore necessarily limited in length, we must be careful that we do not overfit the models λ_i. Assuming fully forward-connected, left-to-right models λ_i, increasing the number of states from n_s to $(n_s + 1)$ increases the number of free (trainable) parameters by $n_s + L$, where L is the number of observables. Thus, having too many states in the HMMs substantially increases the chance of overfitting, since there may be too many degrees of freedom in the model. Conversely, by minimizing the number of states, the likelihood of overfitting is minimized.

A second reason that the single-observable, single-state case performs better relates to the vector quantization process. To understand how, consider that each input vector $\bar{\zeta}(k)$ minimally includes $2n_r$ road inputs $\{r_x^{n_r}, r_y^{n_r}\}$. If we let $n_r = 10$, then for $n_x = n_u = 1$, 80% of the input dimensions will be road-related, while only 20% will be state related. Thus, the vector quantization will most heavily minimize the distortion of the road inputs, while in comparison neglecting the potentially crucial state and previous command inputs. With larger values of n_x, and n_u, the vector quantization process relies more equally on the state, previous control and road inputs, and therefore forms more pertinent feature (prototype) vectors for control.

Thus, for a VQ codebook Q_L, input vector $\bar{\zeta}^*$, $n_x = n_u = m$, $n_r = 10$, and $n_O = 1$,

$$O^* = l = T_{VQ}^{v}(\bar{\zeta}^*, Q_L) \text{, and} \tag{5-17}$$

$$P(A_i|\bar{\zeta}^*) = P(A_i|O^*) \propto P(O^*|\lambda_i)P(A_i) = b(l)_i P(A_i), \tag{5-18}$$

where $b(l)_i$ denotes the lth element in the λ_i model's output probability vector B_i. If we interpret the codebook vectors $\bar{q}_l$ as states S_l, then the discontinuous controller can be viewed as a learned stochastic policy that maps states S_l to actions A_i, where,

$$P(A_i|S_l) = b(l)_i P(A_i) / \sum_{i=1}^{N} b(l)_i P(A_i). \tag{5-19}$$

5.1.4 Prior probabilities

In order to calculate $P(A_i|S_l)$, we need to assign values to the prior probabilities $P(A_i)$. One approach is to estimate $P(A_i)$ by the frequency of occurrence of action A_i in a particular control data set X. For $\phi(k) \geq 0$,

$$P(A_i) = \begin{cases} n_i / \sum_{k=1}^{4} n_k & i \in \{1, 2, 3, 4\} \\ 0 & i \in \{5, 6, 7, 8\} \end{cases}, \tag{5-20}$$

where n_i denotes the number of times action A_i was executed in the data set X; similarly, for $\phi(k) < 0$,

$$P(A_i) = \begin{cases} 0 & i \in \{1, 2, 3, 4\} \\ n_i / \sum_{k=5}^{8} n_k & i \in \{5, 6, 7, 8\} \end{cases} \tag{5-21}$$

5.1.5 Task-based modifications

While the assignment of priors in equations (5-20) and (5-21) are the best estimates for $P(A_i)$ from the data X, they are sometimes problematic when dealing with marginally stable training data. Consider, for example, Figure 5-2, where we plot a small part of Groucho's first run. We observe that Groucho's trajectory takes him close to the edge of the road; what keeps him from driving off the road is the switch from the gas to the brake at time t_s. Now, because the action selection criterion in equation (5-5) is stochastic, it is possible that the stochastic controller will only brake at time $t_s + \tau$, even if it correctly models that time t_s is the most likely time for a control switch. Braking at time $t_s + \tau$, however, may be too late for the car to maintain contact with the road.

Consequently, we would like to improve the stability margins of the stochastic control model. The stability of the system (i.e. the simulated car) is directly related to the kinetic energy T being pumped into the system,

$$T \propto \sum_k \phi(k), \tag{5-22}$$

so that the expected value of T, $E[T]$, is proportional to,

$$E[T] \propto \sum_k E[\phi(k)]. \tag{5-23}$$

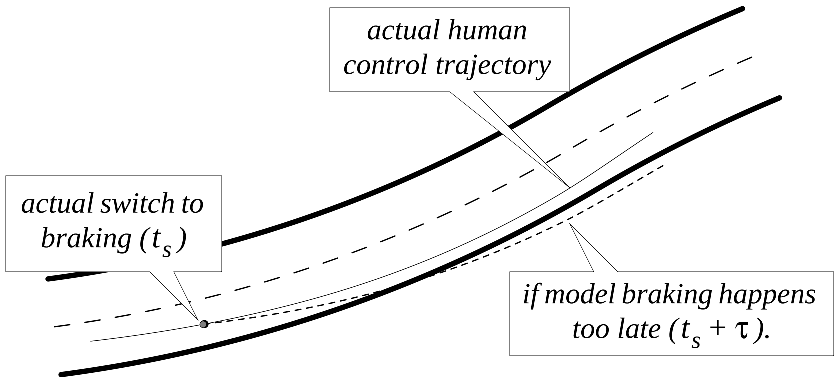

actual human control trajectory
actual switch to braking (t_s)
if model braking happens too late ($t_s + \tau$).

Figure 5-2: Instability can result if the stochastic controller switches to braking too late.

Hence, for increased stability margins, we want to adjust the stochastic model to generate $\phi'(k)$, where,

$$E[\phi'(k)] < E[\phi(k)] \tag{5-24}$$

We can realize condition (5-24) by slight increases in the priors for those actions that decrease $E[\phi(k)]$ — namely, A_3 or A_4. To stay within probabilistic constraints, we offset these changes by slight decreases in the priors A_2 or A_1, respectively, so that the modified priors are given as either,

$$P'(A_3) = P(A_3) + \varepsilon_s \text{ and } P'(A_2) = P(A_2) - \varepsilon_s\text{, or} \tag{5-25}$$

$$P'(A_4) = P(A_4) + \varepsilon_s \text{ and } P'(A_1) = P(A_1) - \varepsilon_s, \tag{5-26}$$

where $\varepsilon_s > 0$ determines the degree to which we decrease $E[\phi'(k)]$. As we shall observe later, for some human control data $P(A_3) = 0$. In that case we choose modification (5-26), so as not to introduce a control action that was never observed in the human control strategy. When $P(A_3)$ is substantial, then we choose modification (5-25).

While instability is a common failure mode of the unmodified stochastic controller, another very rare failure mode leads to exactly the opposite: the brake is engaged too long by the stochastic controller and the simulated car comes to a screeching halt. This problem is very similar to the instability problem, in that a switch — in this case from braking to accelerating — occurs too late. Once the car is stopped, the distortion for the vector-quantized input vector $\bar{\zeta}(k)$ is large for all VQ codebook vectors $\bar{q}_l$. It will be smallest, however, for those codebook vectors where the previous acceleration commands ϕ are less than zero. Hence, once the car is stopped, the brake remains engaged for a long time. Although the stochastic selection criterion in (5-5) ensures that eventually the simulated car will once again switch from braking, we would like to prevent the car from stopping altogether. Unlike the instability problem, this is significantly easier to monitor, since the velocity v directly predicts when a stopping event is about to occur. Consequently, we modify the statistical controller so that,

$$P(A_8 | v < v_{min}) = 1 \, , \, P(A_i | v < v_{min}) = 0 \, , \, i \in \{1, 2, \ldots, 7\} \, , \qquad (5\text{-}27)$$

where v_{min} is chosen to reflect the range of velocities in the human control data. Over repeated trials, condition (5-27) is invoked on average approximately one time per 20km test run.

5.2 Experimental results

5.2.1 Model training

In order to make a fair comparison of the HCS models in this chapter with those of Chapter 4, we select the same input space parameters n_x, n_u and n_r as those listed in Table 4-1. Furthermore, we let $n_O = n_s = 1$, so that the HMMs λ_i are single-state models. As we have already argued, we get significantly better performance from the more constrained models than we do if we let $n_O > 1$ and $n_s > 1$. We vector quantize the training data for each run to $L = 512$ levels. Also, for each run we choose the stabilization parameter ε_s to ensure stability over the validation road #4 (Figure 2-3(a)), and then test the resulting modified controller over test road #5 (Figure 2-3(b)). Table 5-2 summarizes the stabilization parameter and minimum velocity choices for each model. Finally, for the steering control δ, we select the same linear *Ck* as in Chapter 4.

Table 5-2: Hybrid controller design choices

Individual	v_{min} *(mi/h)*	ε_s	$P(A_i)$ *modified*
Larry	50	0.010	$\{P(A_2), P(A_3)\}$
Moe	45	0.005	$\{P(A_1), P(A_4)\}$
Groucho	40	0.005	$\{P(A_1), P(A_4)\}$
Harpo	40	0.005	$\{P(A_1), P(A_4)\}$

5.2.2 HCS models

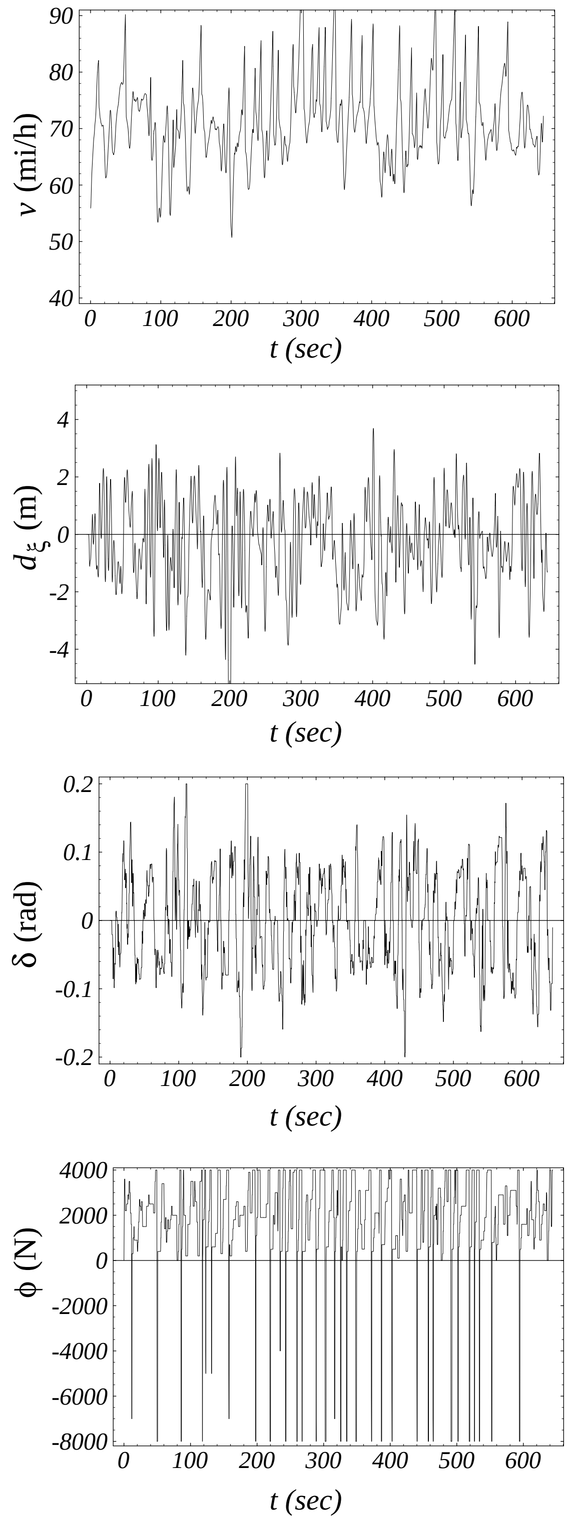
Figures 5-3, 5-4, 5-5 and 5-6 illustrate representative hybrid controller results for the same four runs for which we report cascade network results — namely, $X^{(1,2)}$ (Larry’s second run), $X^{(3,1)}$ (Moe’s first run), $X^{(4,1)}$ (Groucho’s first run), and $X^{(5,2)}$ (Harpo’s second run). Once again part (a) of each Figure plots the original human control data, while part (b) of each Figure plots the corresponding model control over the test road (#5).[1]

Comparing these modeling results to the *Cq* and *Ck* results in Figures 4-1 through 4-4 and 4-5 through 4-9, respectively, we ask ourselves, which controller, the continuous cascade network controllers, or the discontinuous stochastic controllers, perform better? The answer to that question depends on what precisely is meant by “better.”

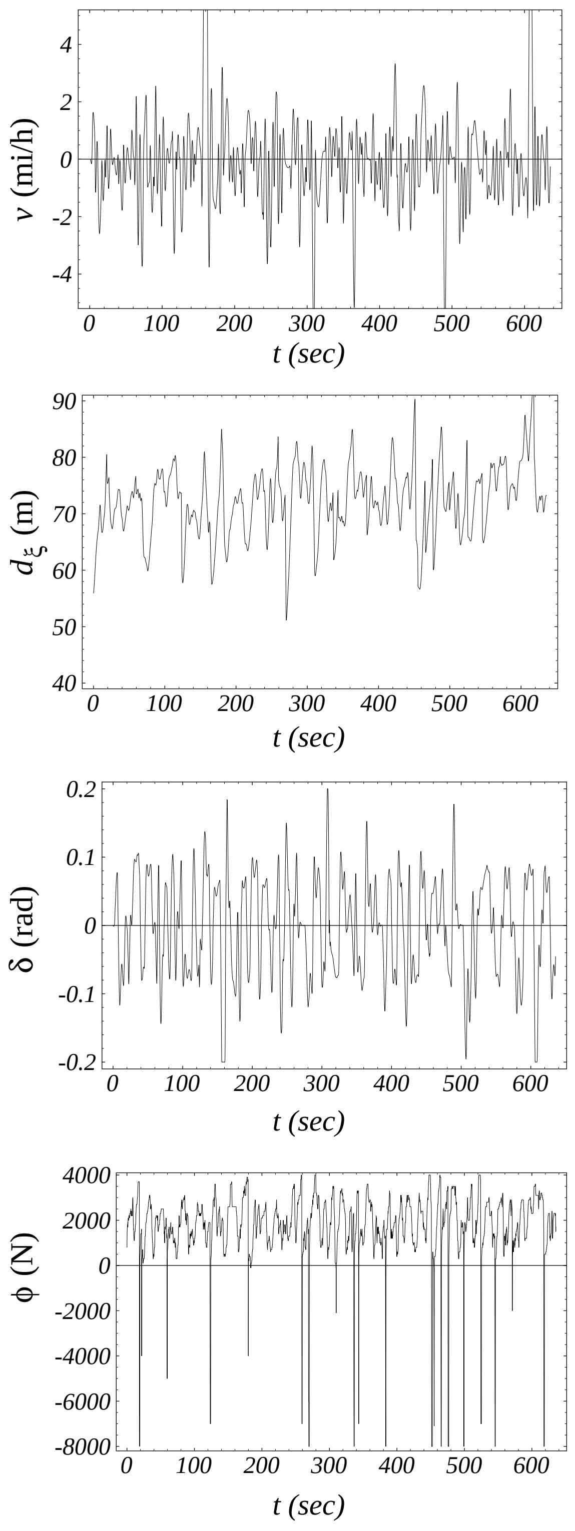
If we evaluate the two modeling strategies based on absolute performance criteria, such as range of stability, the cascade network controllers probably perform better. Whereas the linear model controllers rarely, if ever, run off the road, the hybrid controllers temporarily run off the test road more often (for $w = 10\text{m}$). Simply put, the linear controllers appear more stable than their hybrid counterparts.

If, on the other hand, we evaluate the two modeling approaches on how closely they approximate the corresponding operator’s control strategy, then the verdict likely changes. As we have already noted, the *Ck* models’ control trajectories do not look anything like their human counterparts’ trajectories, due to the continuous models’ inability to faithfully model the discontinuous acceleration command ϕ. Qualitatively, the hybrid continuous/discontinuous controllers appear to approximate more closely their respective training data.[2]

1. Note that for these results, the trajectories that are shown are but one example of the discontinuous controller’s strategy over road #5, since the control action selection criterion in (5-5) is stochastic.
2. Once again, we will formalize this qualitative observation in Chapter 8.

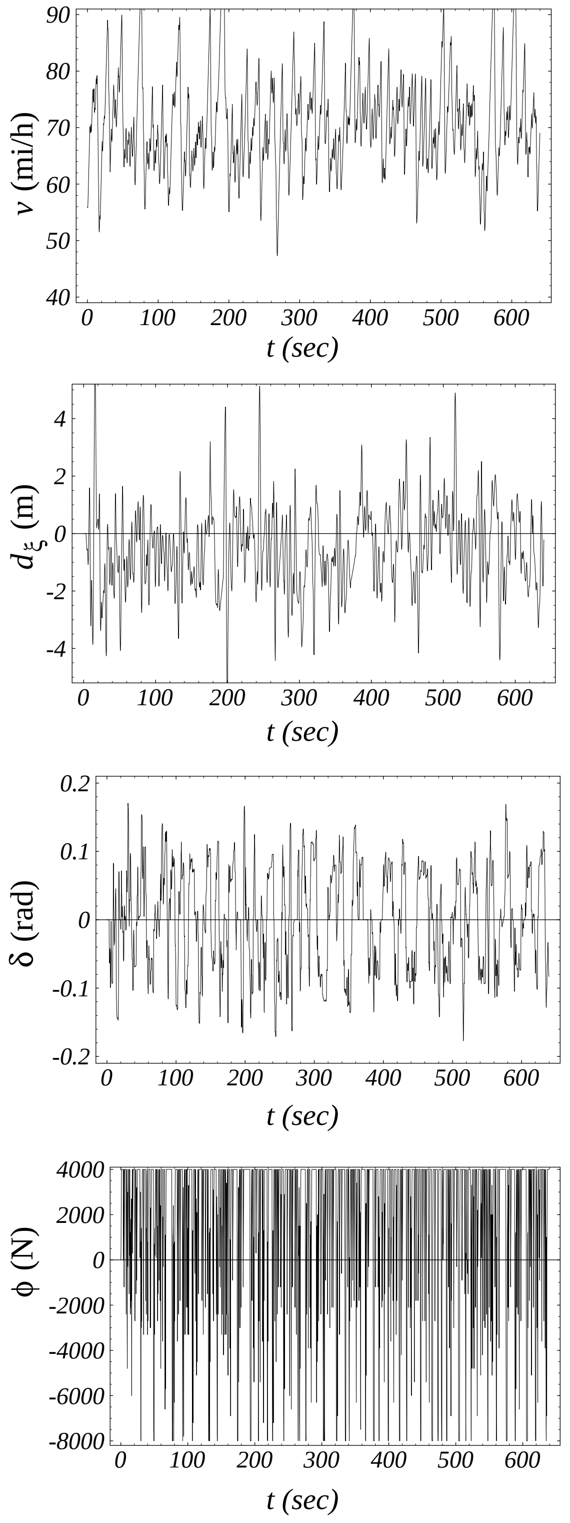


(a) Larry's control data

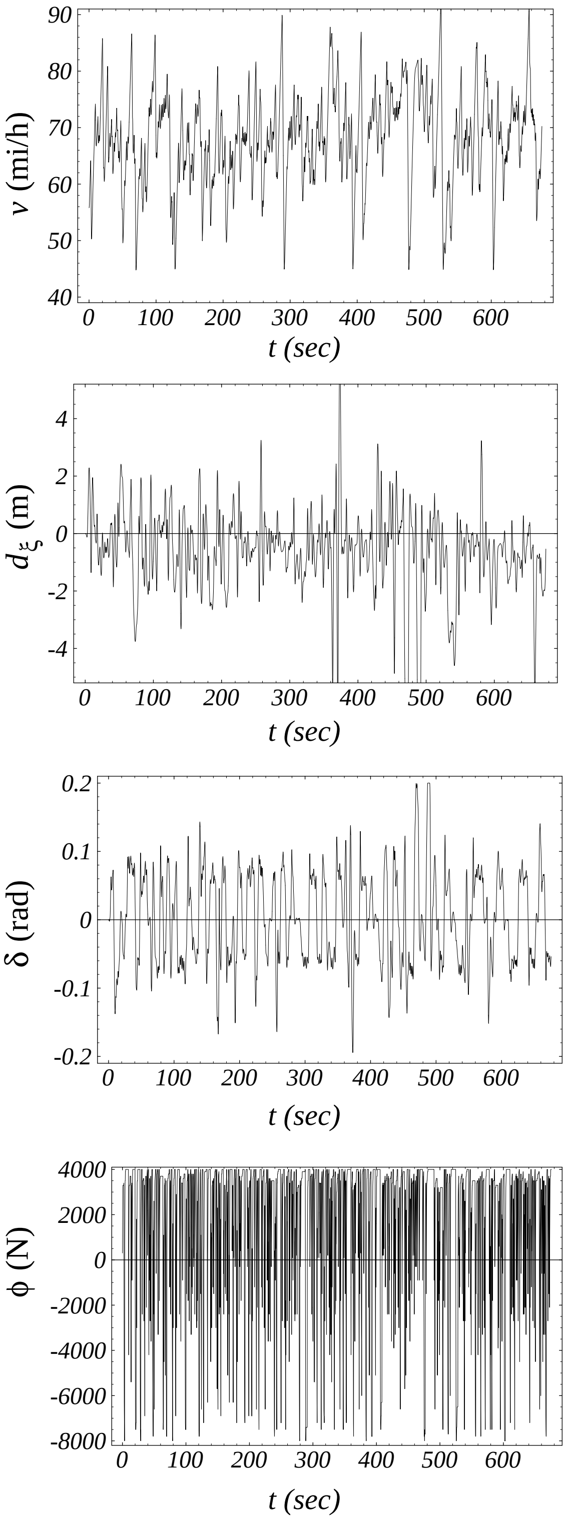


(b) hybrid model control trajectory

Figure 5-3: Larry's (a) training data and (b) corresponding hybrid controller data.

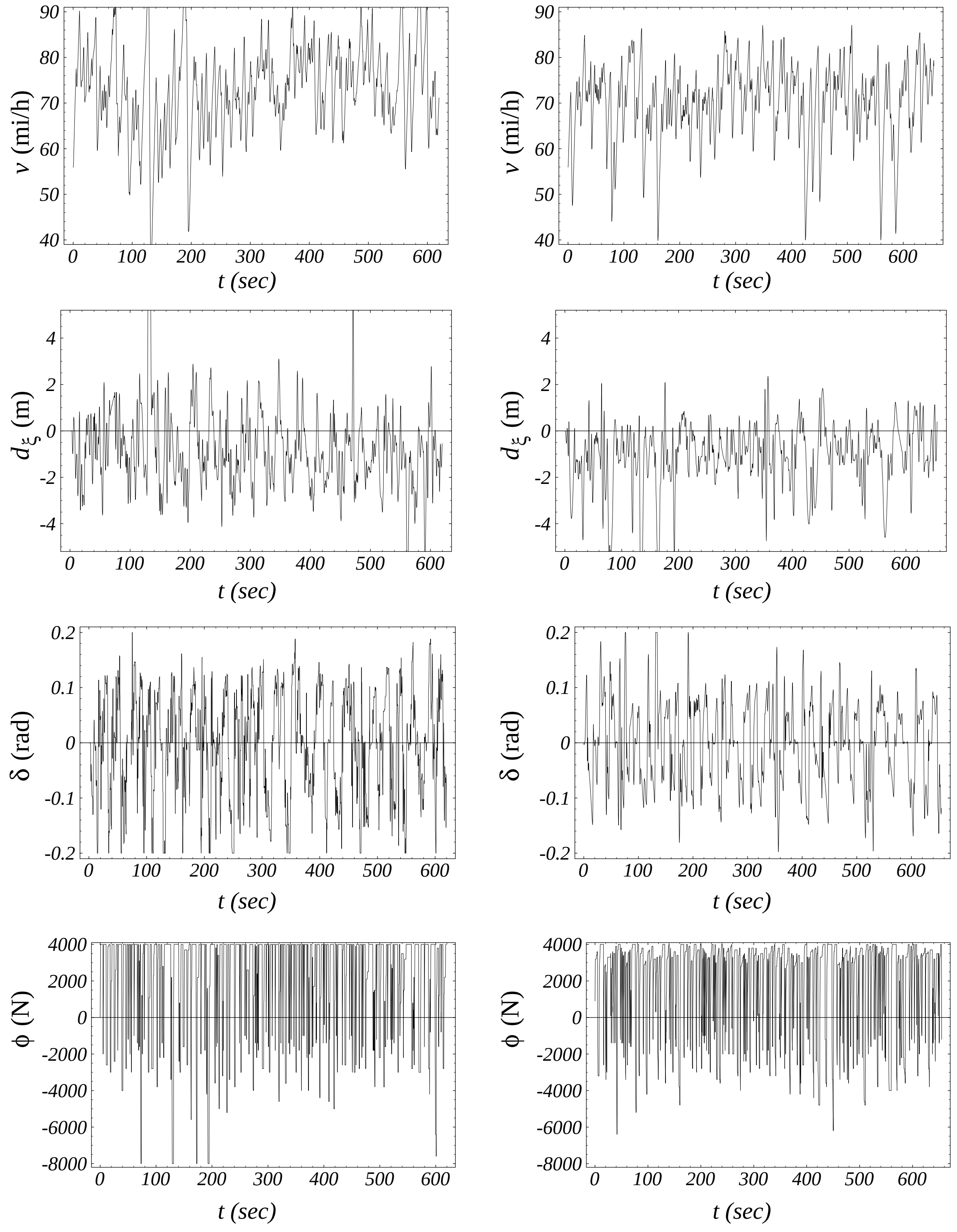


(a) Moe's control data



(b) hybrid model control trajectory

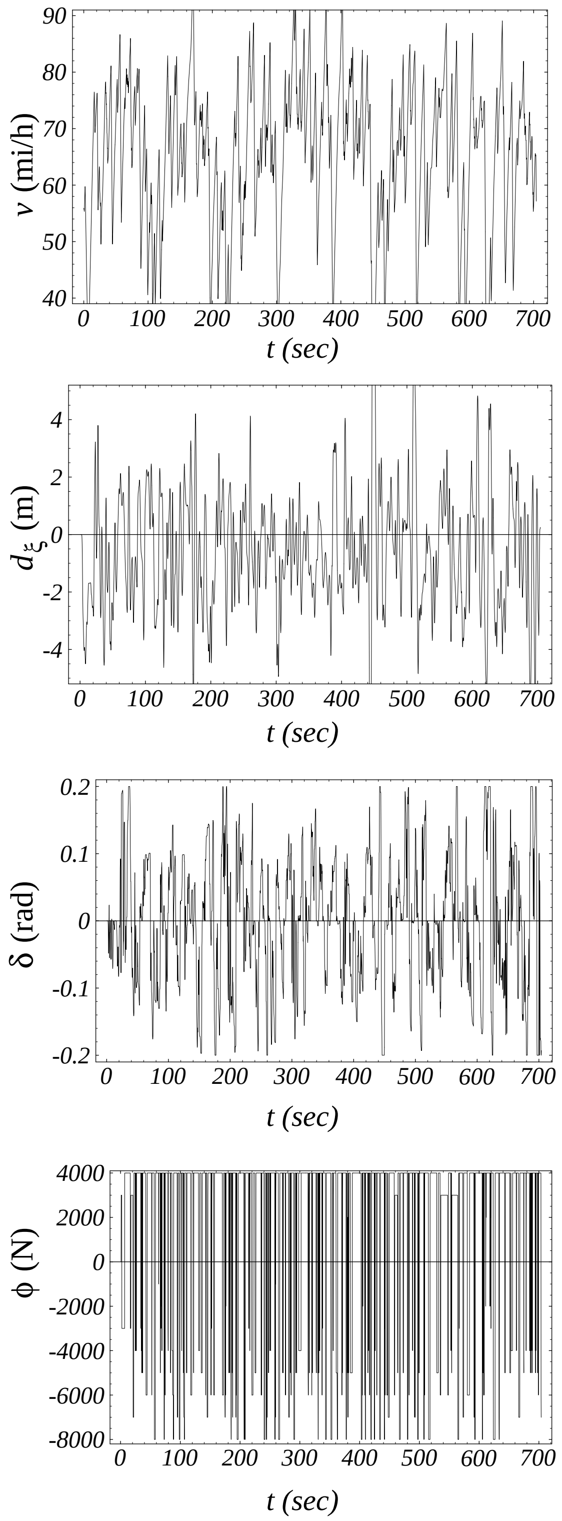
Figure 5-4: Moe's (a) training data and (b) corresponding hybrid controller data.



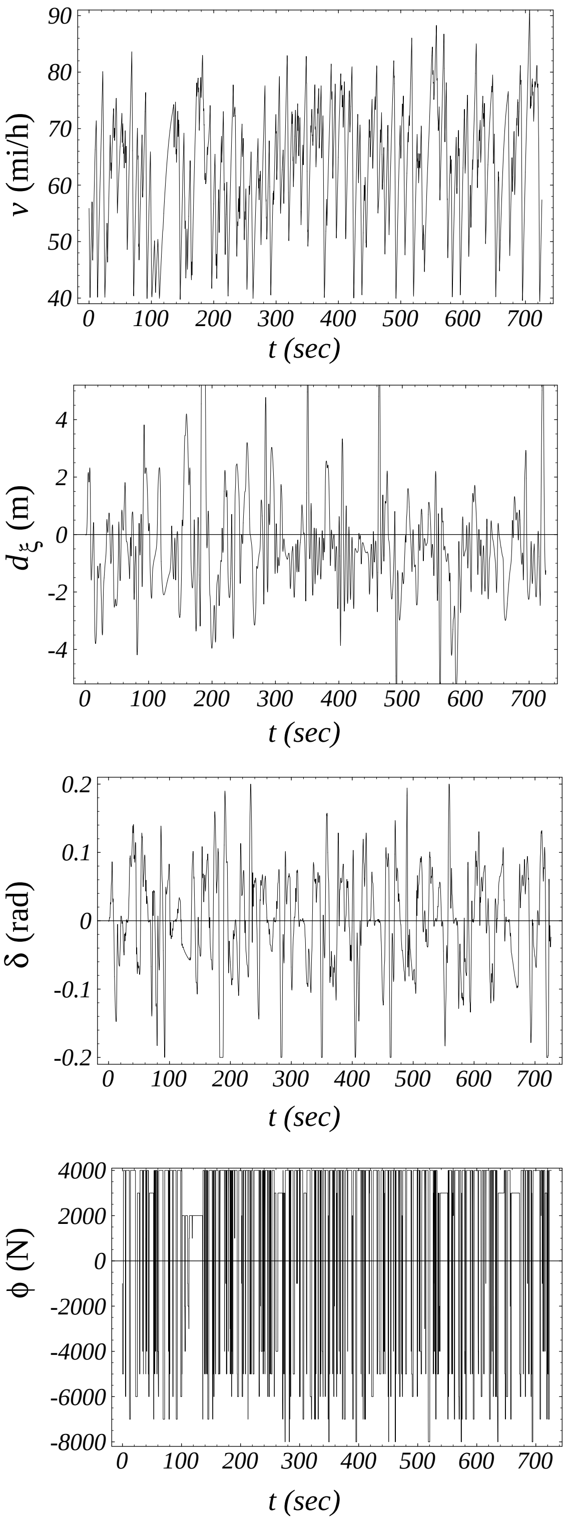
(a) Groucho's control data

(b) hybrid model control trajectory

Figure 5-5: Groucho's (a) training data and (b) corresponding hybrid controller data.



(a) Harpo's control data



(b) hybrid model control trajectory

Figure 5-6: Harpo's (a) training data and (b) corresponding hybrid controller data.

5.3 Analysis

5.3.1 Sample curve control

Here, we examine the control behavior of the hybrid models in somewhat greater detail — through a road sequence composed of (1) a 75m straight-line segment, and (2) a subsequent 150m-radius, 120° curve. Since, this particular road sequence appears in road #1 (Figure 2-2(c)), we can directly compare the actual human control strategy with the corresponding hybrid-model control.

Figures 5-7, 5-8, 5-9 and 5-10 plot the driving control for Larry, Moe, Groucho and Harpo and their respective hybrid models. In each Figure, the vertical lines indicate the start of the turn for the human (dashed) and the corresponding hybrid-model control data (solid).

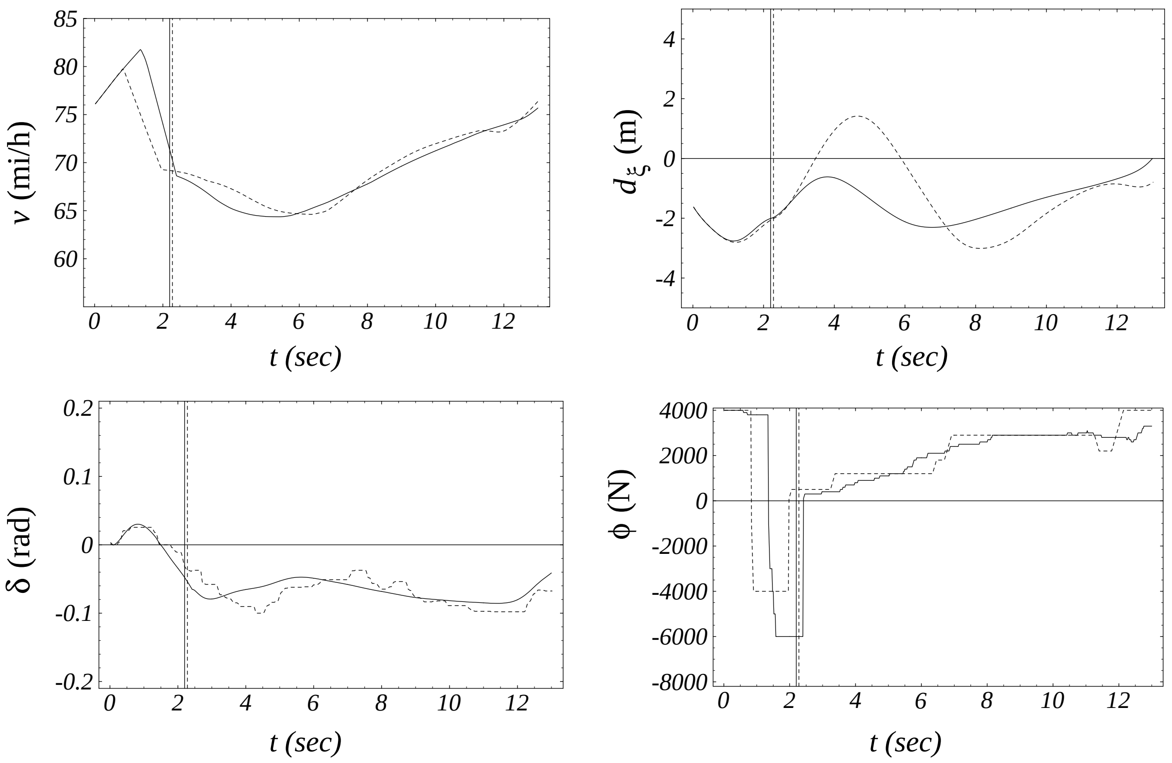


Figure 5-7: Larry's (dashed) and his hybrid model's (solid) control through a given turn.

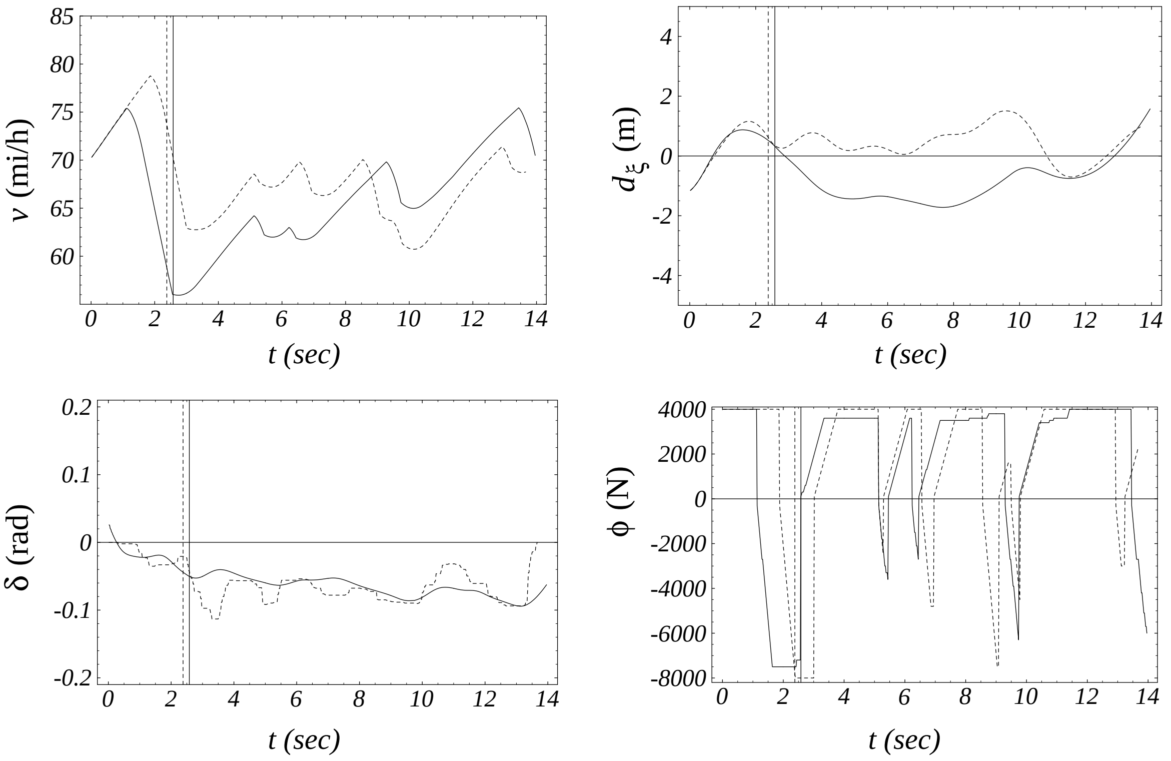


Figure 5-8: Moe's (dashed) and his hybrid model's (solid) control through a given turn.

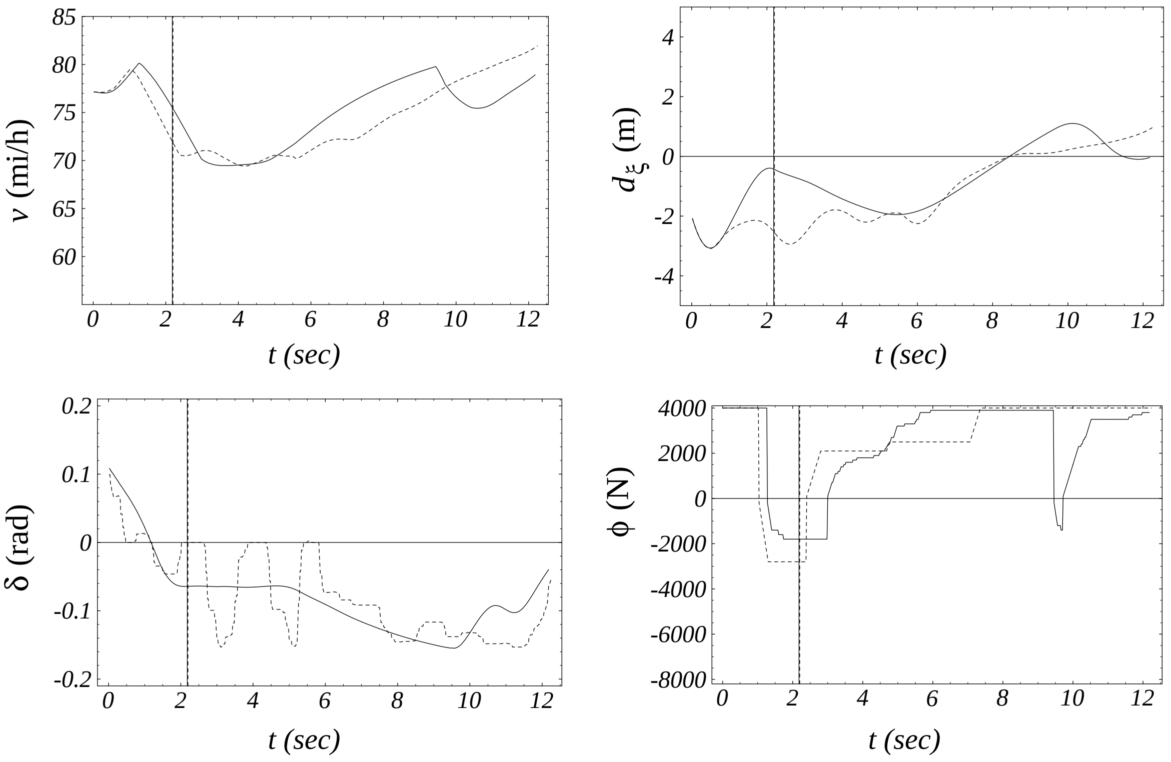


Figure 5-9: Groucho's (dashed) and hybrid model's (solid) control through a given turn.

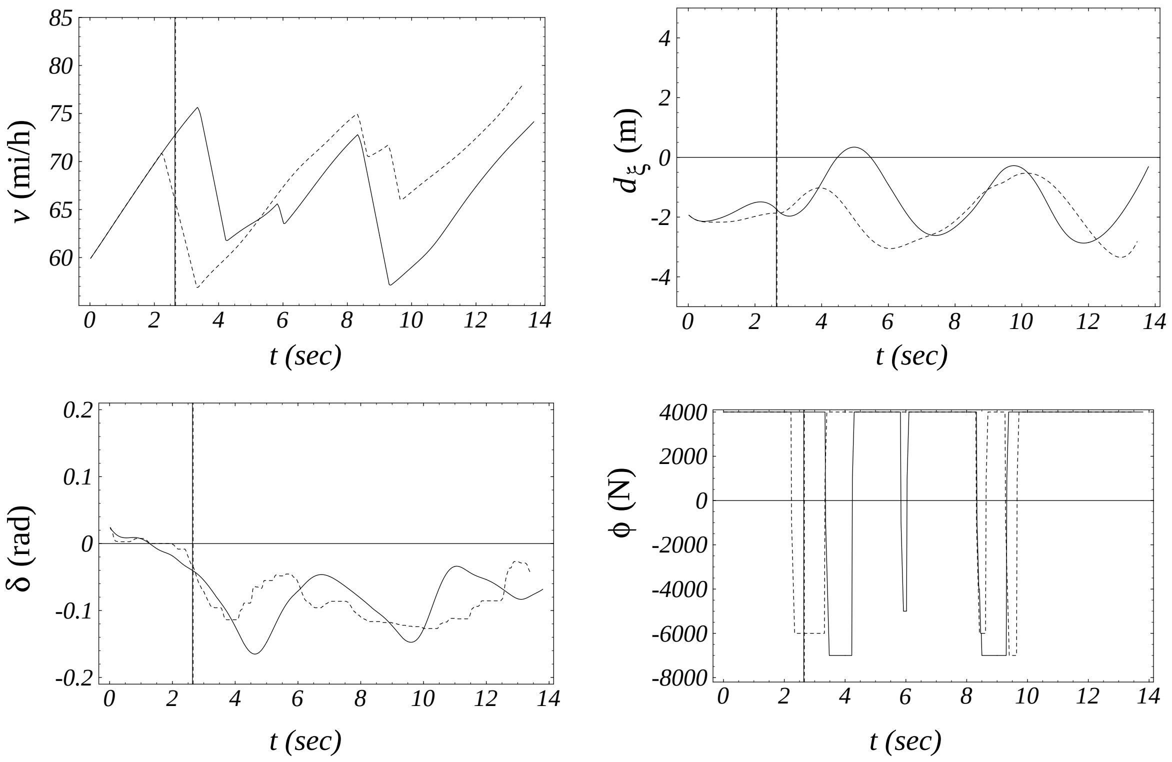


Figure 5-10: Harpo's (dashed) and his hybrid model's (solid) control through a given turn.

From these Figures we make a few general observations. First, each hybrid model completes the curve in almost exactly the same time as its respetive human. The largest difference in completion times — 2.5% (0.34 sec) — occurs for Harpo; in two of the other cases (Larry and Groucho), the time difference between the actual and hybrid controls is less than 0.5%.

Second, since the models' steering is handled through a linear, continuous mapping, the steering δ profiles for the models vary more smoothly than their human counterparts. Consequently, the lateral offset from the road median (d_ξ) also differs between the model and human control trajectories[3].

3. Although lateral offset from the road median is an important criterion for distinguishing between drivers in *real* driving, we shall see later that in our type of *simulated* driving, drivers pay little attention to d_ξ as long as they maintain contact with the road. Since drivers are inconsistent in their lateral lane position from one curve to the next, we cannot expect that a model will very closely track lateral lane position in any specific instance.

Finally, while the applied force ϕ profiles between the humans and corresponding hybrid models are not identical, they are, in fact, similar. For Larry, the hybrid model's initial brake heading into the curve occurs approximately 1/2 second after Larry's initial brake. Thus, the model is slightly faster when first braking; to compensate for the higher speed, the model brakes slightly harder, and thereafter tracks Larry's applied force profile closely.

Groucho's case is similar to Larry's. The initial brake for the model occurs approximately 1/4 second after Groucho's initial brake. In this case, however, the model compensates not by braking harder, but by braking longer. Thereafter, the main difference between Groucho's control and the model is a quick brake maneuver while in the turn to compensate for the model's somewhat larger acceleration in the turn. Harpo's model also initially brakes after Harpo (by about 1 second), and consequently is also forced to brake more while in the turn to compensate for the higher speed going into the turn.

Moe is perhaps the most interesting of the four cases. This time, the model initially brakes approximately 1/2 second before Moe himself does, albeit with somewhat less force than Moe. Thereafter, the hybrid model closely emulates Moe's strategy of rapid switching between the brake and the accelerator while in the turn.

In summary, we observe that each hybrid controller, while not replicating the human's control strategy exactly, does a good job of emulating its respective human's turn maneuver. The following section examines the underlying reason for this success.

5.3.2 Probability profile

The most important reason behind the success of the hybrid controller is that it is able to successfully model the switching behavior between the gas and brake pedals as a probabilistic event, since the *precise time* that a switch occurs is not that important (as we observed in the previous section). What is more important is that the switch take place in some *time interval* around the time that the human operator would have executed the switch. Consider, for example, Figure 5-11, which plots the posterior probabilities $P(A_i|O)$ for a small segment of Groucho's hybrid model control. We see that switches between the gas and brake pedals (actions A_4 and A_8), while never very likely for any individual time step, are modeled as intervals where,

$$P(A_4|O) = p > 0\text{, or } P(A_8|O) = p > 0\,. \tag{5-28}$$

The probability that a switch will occur after m time steps given the constant probability p is given by,

$$1 - (1 - p)^m \tag{5-29}$$

Figure 5-12 plots this probability as a function of time (at 50 Hz) for $p = 0.1$ and $p = 0.05$. Thus, we see that even for small values of p, the likelihood of a switch rises quickly as a function of time.

Because we train *separate* models λ_i for each action A_i, the hybrid modeling approach does not encounter the same one-to-many mapping problem, illustrated in Figures 4-19 and 4-20, that the continuous cascade networks encounters. The relatively few occurrences of switching in each control data set are sufficient training data, since the switching models λ_4 and λ_8 see *only* that data during training. Including the priors $P(A_i)$ in the action selection criterion (5-5) then ensures that the model is not overly biased towards switching.

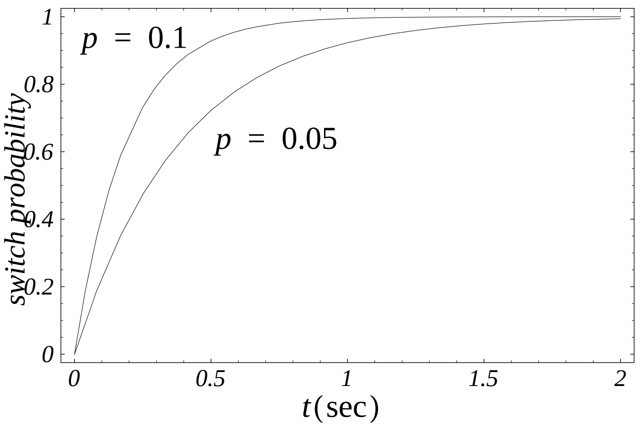


Figure 5-12: Probability of a switch after t seconds (at 50 Hz) when the probability of a switch at each time step is p.

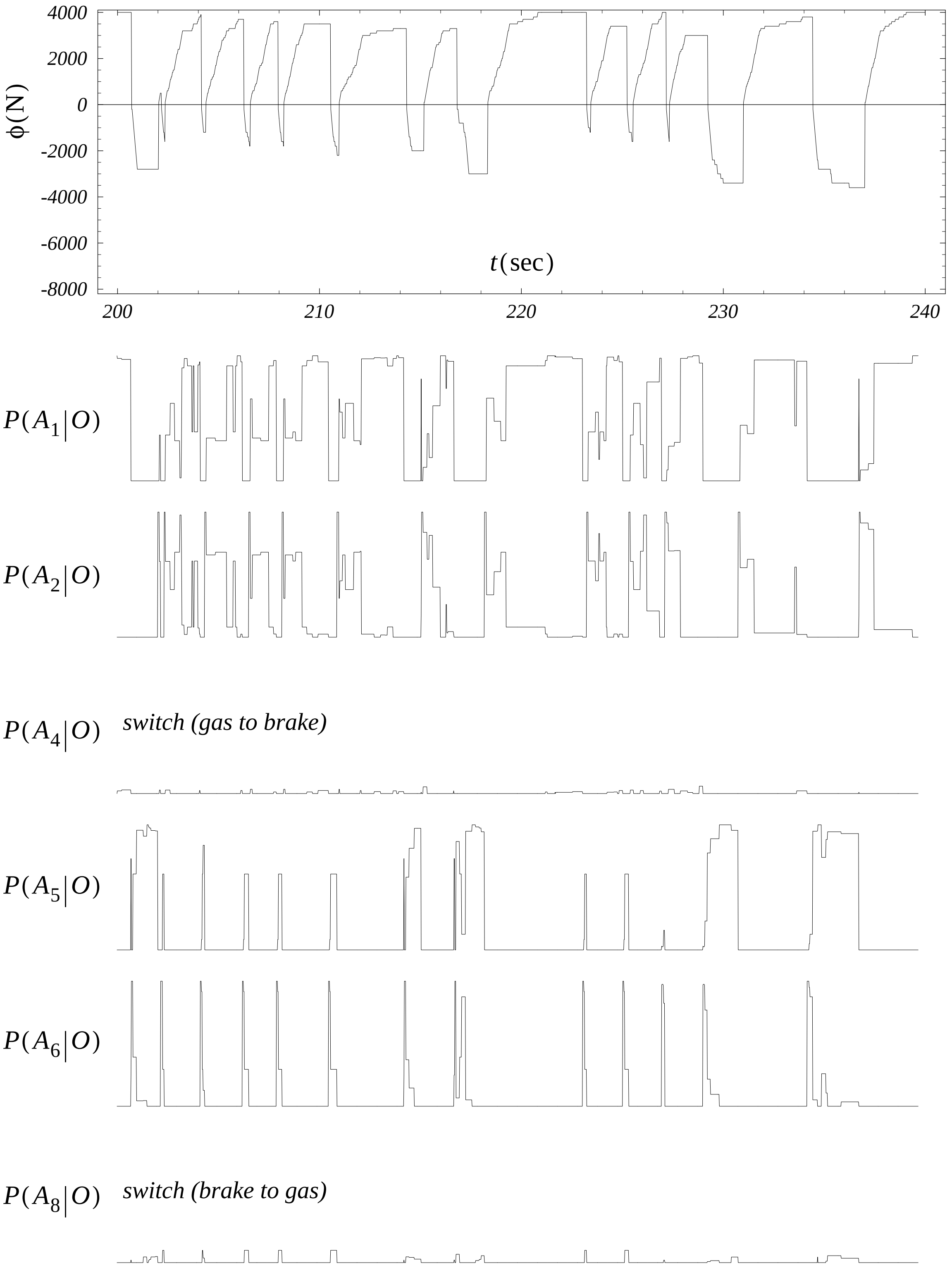


Figure 5-11: Posterior probs. for Groucho's model control ($P(A_3) = P(A_7) = 0$).

5.3.3 Modeling extension

Suppose the acceleration control ϕ were not constrained by equations (2-20) and (2-21), and thus were not as readily expressible through discrete actions. For example, suppose that the separate gas and brake commands could change by an arbitrary amount for each time step, not just by $\Delta\phi_g$ and $\Delta\phi_b$. How would this change the proposed control framework?

Figure 5-13 suggests one possible solution. Initially, we train two separate continuous controllers, the first corresponding to $\phi(k) \geq 0$, and the second corresponding to $\phi(k) < 0$. Since these controllers would not be required to model switches between braking and accelerating, the control outputs will vary continuously and smoothly with model inputs; hence a continuous function approximator should be well suited for these two modeling tasks.

Then, we train four statistical models $\tilde{\lambda}_i$, corresponding to actions $\tilde{A}_i$, $i \in \{1, 2, 3, 4\}$, where actions $\tilde{A}_1$ and $\tilde{A}_2$ correspond to *no switch* at the next time step for $\phi(k) \geq 0$ and $\phi(k) < 0$, respectively, and actions $\tilde{A}_3$ and $\tilde{A}_4$ correspond to *a switch* at the next time step for $\phi(k) \geq 0$ and $\phi(k) < 0$, respectively. This discontinuous action model would then regulate which of the continuous models is active at each time step k. Although the discontinuous controller's func-

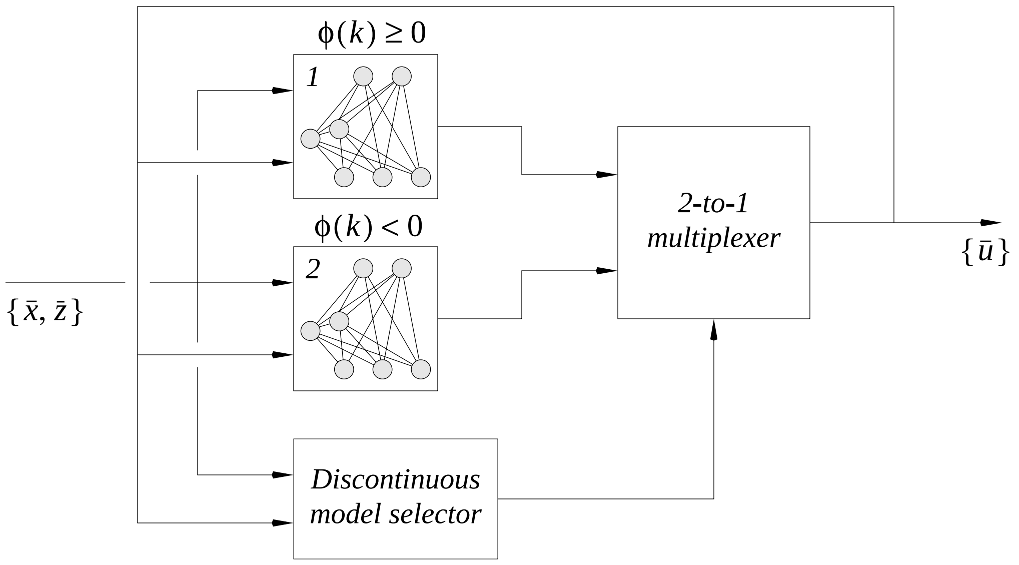


Figure 5-13: Alternative architecture for discontinuous strategies.

tion in this scheme is reduced, it does preserve the critical role of the discontinuous controller in properly modeling the switching behavior, without the introduction of high-frequency noise. In fact, Figure 5-13 offers a modeling architecture which is applicable whenever discrete events or actions disrupt the continuous mapping from inputs to outputs.

Of course, the proposed statistical framework does have some limitations in comparison to functional modeling approaches. Because we vector quantize the input space, the stable region of operation for the hybrid controller is strictly limited to the input space spanned by the VQ codes. In fact, we observe from the modeling results that the continuous *Ck* models are more stable than the hybrid discontinuous/continuous models. A second limitation of the approach is the inclusion of the prior probabilities $P(A_i)$ in the stochastic selection criterion (5-5). By including the priors, we are assuming environmental conditions similar to the training environment. Radically different environmental conditions during testing presumably change the values of the priors, and therefore make the action selection criterion less appropriate.

In summary, we have developed a discontinuous modeling framework for abstracting discontinuous human control strategies, and have compared the proposed approach to competing continuous function approximators. Which control approach is preferred ultimately depends on the specific application for the HCS model. If the model is being developed towards the eventual control of a real robot or vehicle, then the continuous modeling approach might be preferred as a good starting point. Continuous models extrapolate control strategies to a greater range of inputs, show greater inherent stability, and lend themselves more readily to theoretical performance analysis. If, on the other hand, the model is being developed in order to simulate different human behaviors in a virtual reality simulation or game, then the discontinuous control approach might be preferred, since fidelity to the human training data and random variations in behavior would be the desired qualities of the HCS model. Thus, depending on the application, we believe a need exists for both types of modeling approaches.

In the next several chapters, we develop a stochastic similarity measure as the first step in a post-training model-validation procedure. In Chapter 8, we then use this similarity measure to quantify our previous qualitative observations about the level of similarity (or dissimilarity) between the original human control data and the model-generated trajectories.

Chapter 6

Model validation

In previous chapters, we investigated different machine learning techniques for abstracting models of human control strategy. Each of these methods learns, to varying degrees, stable HCS models from the experimental data. As we observed, however, the different modeling techniques — *Cq*, *Ck* and discontinuous learning — generate control trajectories that are qualitatively quite different from one another, despite training from identical human control data. This is true not only because the modeling capacity of each approach differs, but also because modeling errors can feed back on themselves to generate state and command trajectories that are uncharacteristic of the source process. Therefore, for feedback control tasks, such as human driving, we suggest that post-training model validation is not only desirable, but essential to establish the degree to which the human and model-generated trajectories are similar.

In this chapter, we first demonstrate the need for model validation with some illustrative examples. We then propose a stochastic similarity measure — based on Hidden Markov Model analysis — for comparing stochastic, dynamic, multi-dimensional trajectories. This similarity measure can then be applied towards validating a learned model's fidelity to its training data by comparing the model's dynamic trajectories in the feedback loop to the human's dynamic trajectories. Finally, we derive and demonstrate some general properties of the similarity measure for known stochastic systems. In Chapter 7, we will test the similarity measure by comparing human control strategies across different individuals, and will show that the proposed similarity measure outperforms the more traditional Bayes classifier in correctly grouping driving data from the same individual. Chapter 8 then applies the similarity measure towards comparing the HCS models learned in the previous chapters to their respective human control data.

6.1 Need for model validation

The main strength of modeling by learning, is that no explicit physical model is required; this also represents its biggest weakness, however. On the one hand, we are not restricted by the limitations of current scientific knowledge, and are able to model human control strategies for which we have not yet developed adequate biological or psychological understanding. On the other hand, the lack of scientific justification detracts from the confidence that we can show in these learned models. This is especially true when the unmodeled process is (1) dynamic and (2) stochastic in nature, as is the case for human control strategy. For a dynamic process, model errors can feed back on themselves to produce trajectories which are not characteristic of the source process or are even potentially unstable. For a stochastic process, a static error criterion (such as RMS error), based on the difference between the training data and predicted model outputs is inadequate to gauge the fidelity of a learned model to the source process. Yet, for the static modeling techniques studied in this thesis, some static error measure usually serves as the test of convergence. While this measure is very useful during training, it offers no guarantees, theoretical or otherwise, about the dynamic behavior of the learned model in the feedback control loop.

To illustrate this problem, we consider two examples. In the first example, suppose that we wish to learn a dynamic process represented by the simple difference equation,

$$u(k+1) = 0.75u(k) + 0.24u(k-1) + x(k) \tag{6-1}$$

where $u(k)$, $x(k)$ represent the output and input of the system, respectively, at time step k. For the input/output training data in Table 6-1, at least three different linear models yield the same RMS error (6.16×10^{-3}) over the training set:

Table 6-1: Sample input-output training data

Input			*Output*
$u(k-1)$	$u(k)$	$x(k)$	$u(k+1)$
-0.1	0.1	0.4	0.349
0.1	0.1	0.5	0.599
-0.3	0.2	0.3	0.123
0.3	0.2	0.4	0.673
0.2	0.0	0.5	0.650
0.0	0.2	0.3	0.348

$$\#1\text{: } u(k+1) = 0.76u(k) + 0.25u(k-1) + x(k) \tag{6-2}$$

$$\#2\text{: } u(k+1) = 0.76u(k) + 0.23u(k-1) + x(k) \tag{6-3}$$

$$\#3\text{: } u(k+1) = 0.74u(k) + 0.23u(k-1) + x(k) \tag{6-4}$$

The dynamic trajectories for these models, however, differ markedly. As an example, consider the time-dependent input,

$$x(k) = 0.1\sin(k\pi/100) \tag{6-5}$$

and initial conditions $u(-1) = u(0) = 0$. Figure 6-1 plots the system as well as the model trajectories for $0 \le k \le 300$. We see that model #1 diverges to an unstable trajectory; model #3 remains stable, but approximates the system with significantly poorer accuracy; and model #2 matches the system's response very closely. These responses are predicted by the dominant pole for each difference equation (Table 6-2). Except for the unstable model (#1), each model's dominant pole lies inside the unit circle, thus ensuring stability.

Table 6-2: Dominant poles for each difference equation

system	*model #1*	*model #2*	*model #3*
(0.992, 0)	(1.008, 0)	(0.992, 0)	(0.976, 0)

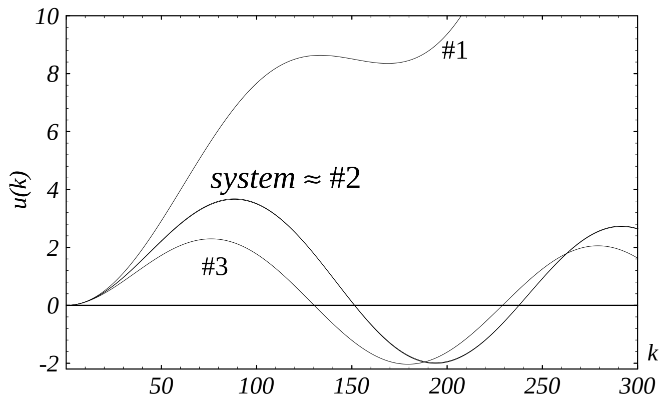


Figure 6-1: The three models result in dramatically different (even unstable) trajectories.

It is apparent from this example that a biased estimator of a marginally stable system may well result in an unstable model, despite RMS errors which appear to be equivalent to those of better models. Not only biased models are a problem, however. Static models of the type shown in Figure 2-4 can achieve deceptively low RMS errors by confusing *causation* with *correlation* between the model's input and output spaces. In our second example, we illustrate this problem with some human driving data collected from Groucho.

In the driving simulator (Figure 2-1), we ask Groucho to drive over road #1 shown in Figure 2-2(a). We simplify the problem by fixing the acceleration command at $\phi = 300\mathrm{N}$, keeping the velocity around 40mph, and requiring only control of the steering angle δ. Now, we train two linear models Γ_1 and Γ_2 with input representations $\{r_x^{10}\}$ and $\{\delta^3\}$, respectively. Note that model Γ_2 receives no road information as input, and is therefore guaranteed to be unstable.

For Γ_1, the RMS error over the data set converges to 9.70×10^{-3}, while for Γ_2, the RMS error converges to 0.71×10^{-3}, an order of magnitude smaller. All that the Γ_2 model has "learned," however, is a *correlation* between the previous values of δ and the next value of δ. In fact, the full model,

$$\delta(k+1) = 0.9997\delta(k) + 0.6647\delta(k-1) - 0.6655\delta(k-2) \tag{6-6}$$

simplifies to (approximately),

$$\delta(k+1) = \delta(k) \tag{6-7}$$

Despite the larger RMS error over the training data, model Γ_1, on the other hand, converges to a stable control strategy, as is shown in Figure 6-2. It learned a *causal* relationship between the curvature of the road ahead and the steering command $\delta(k+1)$.

The two examples above are the extremes. Not all models will be either a good or an unstable approximation of the human control data. In general, similarity between model-generated trajectories and the human control data will vary continuously for different models, from completely dissimilar to nearly identical. Furthermore, for stochastic systems (such as humans), one cannot expect equivalent trajectories for the system and the learned model, given equivalent initial conditions. Therefore, we require a stochastic similarity measure, with sufficient representational power and flexibility to compare multi-dimensional, stochastic trajectories.

6.2 Stochastic similarity measure

Similarity measures or metrics have been given considerable attention in computer vision [12, 20, 121], image database retrieval [54], and 2D or 3D shape analysis [62, 107]. These methods, however, generally rely on the special properties of images, and are therefore not appropriate for analyzing sequential trajectories.

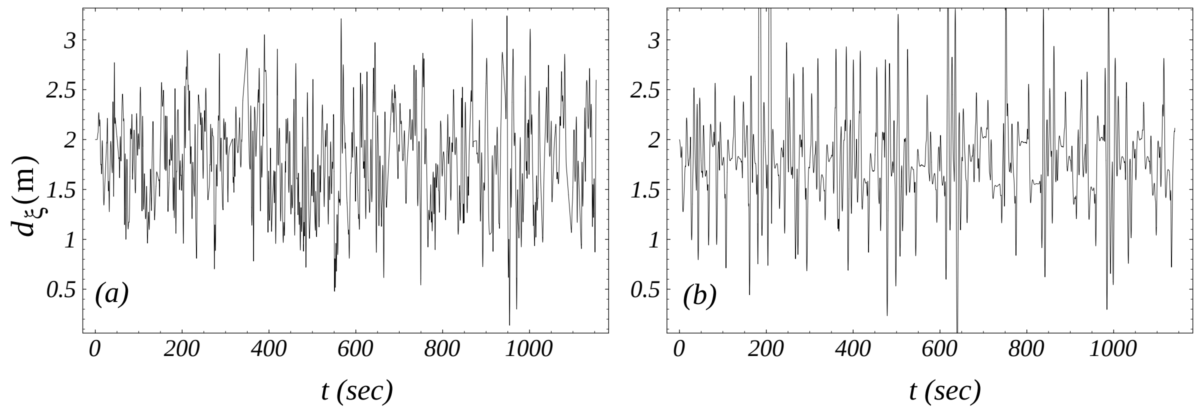


Figure 6-2: Lateral offset (e.g. distance from road median) over time, for (a) Groucho's control strategy, and (b) the first model's control trajectory.

Many parametric methods have been developed to analyze and predict time-series data. One of the more well known, autoregressive-moving average (ARMA) modeling [21], predicts the current signal based on a *linear* combination of previous time histories and Gaussian noise assumptions. Since we have already observed in Chapter 4 that a linear model is insufficient to qualitatively replicate switching, nonlinear control strategies, ARMA models may form a poor foundation upon which to develop a similarity measure. Other work has focussed on classifying temporal patterns using Bayesian statistics [30], wavelet and spectral analysis [114], neural networks (both feedforward and recurrent) [47, 112], and Hidden Markov Models (see discussion below). Much of this work, however, analyzes only short-time trajectories or patterns, and, in many cases, generates only a binary classification, rather than a continuously valued similarity measure. Prior work has not addressed the problem of comparing long, multi-dimensional, stochastic trajectories, especially of human control data. Thus, we propose to evaluate *stochastic similarity* between two dynamic, multi-dimensional trajectories using *Hidden Markov Model (HMM)* analysis.

6.2.1 Hidden Markov Models

Rich in mathematical structure, HMMs are trainable statistical models, with two appealing features: (1) no *a priori* assumptions are made about the statistical distribution of the data to be analyzed, and (2) a degree of sequential structure can be encoded by the Hidden Markov Models. As such, they have been applied for a variety of stochastic signal processing. In speech recognition, where HMMs have found their widest application, human auditory signals are analyzed as speech patterns [51, 94]. Transient sonar signals are classified with HMMs for ocean surveillance in [61]. Radons, *et. al.* [96] analyze 30-electrode neuronal spike activity in a monkey's visual cortex with HMMs. Hannaford and Lee [45] classify task structure in teleoperation based on HMMs. In [123, 124], HMMs are used to characterize sequential images of human actions. Finally, Yang and Xu apply Hidden Markov Models to open-loop action skill learning [126] and human gesture recognition [127].

A Hidden Markov Model consists of a set of *n* states, interconnected through probabilistic transitions; each of these states has some output probability distribution associated with it. Although algorithms exist for training HMMs with both discrete and continuous output probability distributions, and although most applications of HMMs deal with real-valued signals, discrete HMMs are preferred to continuous or semi-continuous HMMs in practice, due to their relative computational simplicity (orders of magnitude more efficient) and lesser sensitivity to initial random parameter settings [95]. In Section 6.2.5 below, we describe how we use discrete HMMs for analysis of real-valued signals by converting the data to discrete symbols through pre-processing and vector quantization. Section 6.2.7 follows with methods for minimizing the detrimental effects of discretization.

A discrete HMM is completely defined by the following triplet [94],

$$\lambda = \{A, B, \pi\} \tag{6-8}$$

where A is the probabilistic $n_s \times n_s$ state transition matrix, B is the $L \times n_s$ output probability matrix with L discrete output symbols $l \in \{1, 2, \ldots, L\}$, and π is the n-length initial state probability distribution vector for the HMM. Figure 6-3, for example, represents a 5-state HMM, where each state emits one of 16 discrete symbols.

We define the notion of *equivalent* HMMs for two HMMs λ_1 and λ_2 such that,

$$\lambda_1 \sim \lambda_2, \text{ iff. } P(O|\lambda_1) = P(O|\lambda_2), \forall O \tag{6-9}$$

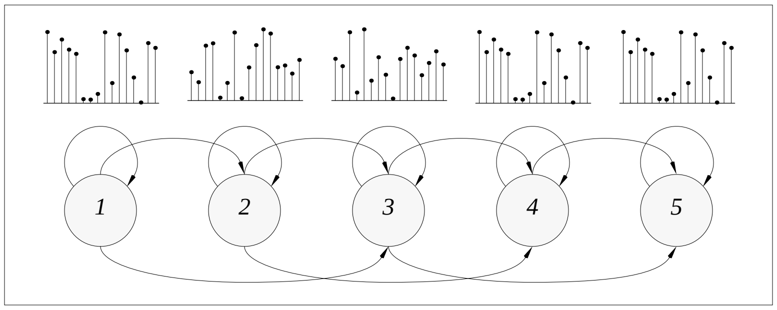


Figure 6-3: A 5-state Hidden Markov Model, with 16 observable symbols in each state.

Note that λ_1 and λ_2 need not be identical to be equivalent. The following two HMMs are, for example, equivalent, but not identical:

$$\lambda_1 = \left\{ \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T, \begin{bmatrix} 1 \end{bmatrix} \right\}, \lambda_2 = \left\{ \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T \right\} \tag{6-10}$$

Finally, we note that for an observation sequence O of discrete symbols and an HMM λ, we can locally maximize $P(\lambda|O)$ using the well-known Baum-Welch algorithm (see Section 6.2.8) [94, 16]. We can also evaluate $P(O|\lambda)$ using the computationally efficient forward-backward algorithm.

6.2.2 Similarity measure

Here, we derive a stochastic similarity measure, based on discrete-output HMMs. Assume that we wish to compare observation sequences from two stochastic processes Γ_1 and Γ_2. Let $\bar{O}_i = \{O_i^{(k)}\}$, $k \in \{1, 2, \ldots, n_i\}$, $i \in \{1, 2\}$, denote the set of n_i observation sequences of discrete symbols generated by process Γ_i. Each observation sequence is of length $T_i^{(k)}$, so that the total number of symbols in set $\bar{O}_i$ is given by,

$$\bar{T}_i = \sum_{k=1}^{n_i} T_i^{(k)}, \; i \in \{1, 2\}. \tag{6-11}$$

Also let $\lambda_j = \{A_j, B_j, \pi_j\}$, $j \in \{1, 2\}$, denote a discrete HMM locally optimized with the Baum-Welch algorithm to maximize,

$$P(\lambda_j|\bar{O}_j) = \prod_{k=1}^{n_i} P(\lambda_j|O_j^{(k)}), \; j \in \{1, 2\}, \tag{6-12}$$

and let,

$$P(\bar{O}_i|\lambda_j) = \prod_{k=1}^{n_i} P(O_i^{(k)}|\lambda_j) \tag{6-13}$$

$$P_{ij} = P(\bar{O}_i|\lambda_j)^{1/\bar{T}_i},\ i, j \in \{1, 2\} \tag{6-14}$$

denote the probability of the observation sequences $\bar{O}_i$ given the model λ_j, normalized with respect to the sequence lengths $\bar{T}_i$[1].

Using the definition in (6-14), Figure 6-4 illustrates our overall approach to evaluating similarity between two observation sequences. Each observation sequence is first used to train a corresponding HMM; this allows us to evaluate P_{11} and P_{22}. We then cross-evaluate each observation sequence on the other HMM (i.e. $P(\bar{O}_1|\lambda_2)$, $P(\bar{O}_2|\lambda_1)$) to arrive at P_{12} and P_{21}. Given, these four normalized probability values, we define the following similarity measure between $\bar{O}_1$ and $\bar{O}_2$:

$$\sigma(\bar{O}_1, \bar{O}_2) = \sqrt{\frac{P_{21}P_{12}}{P_{11}P_{22}}} \tag{6-15}$$

6.2.3 Properties

In order for the similarity measure to obey certain important properties, we restrict the HMMs λ_1 and λ_2 to have the same number of states such that,

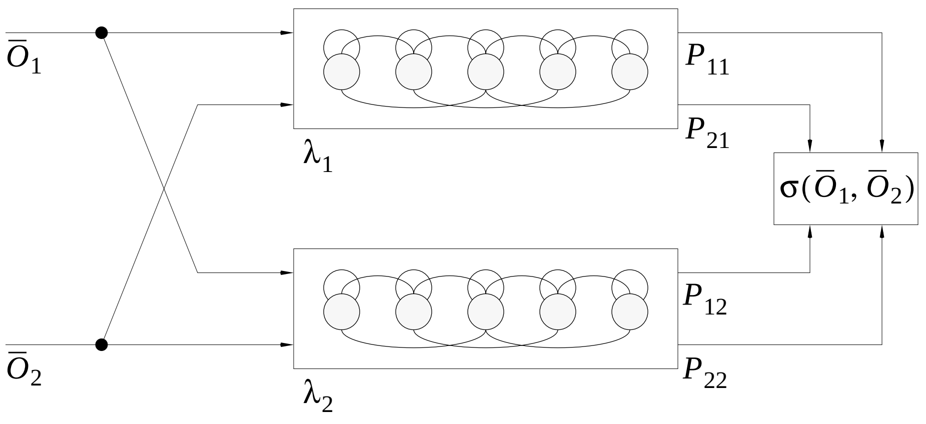


Figure 6-4: Four normalized probability values make up the similarity measure.

1. In practice, we calculate P_{ij} as $10^{\log P(\bar{O}_i|\lambda_j)/\bar{T}_i}$ to avoid problems of numerical underflow for long observation sequences.

$$n_{s,1} = n_{s,2} \tag{6-16}$$

where $n_{s,j}$, $j \in \{1, 2\}$ denotes the number of states in model λ_j.

Now, let us assume that the P_{ii} are global (rather than just a local) maxima[2]. We define model λ_i to be a *global maximum* if and only if,

$$P(O_i|\lambda_i) \geq P(O_i|\lambda), \forall\lambda, n_s = n_{s,i}, \tag{6-17}$$

where n_s is the number of states in model λ. With this assumption, we have that,

$$1/L \leq P_{ii}, \text{ and} \tag{6-18}$$

$$0 \leq P_{ij} \leq P_{ii}.^3 \tag{6-19}$$

The lower bound for P_{ii} in (6-18) is realized for single-state discrete HMMs, and a uniform distribution of symbols in $\bar{O}_i$. From (6-15) to (6-19), we derive the following properties for $\sigma(\bar{O}_1, \bar{O}_2)$:

$$\text{Property \#1: } \sigma(\bar{O}_1, \bar{O}_2) = \sigma(\bar{O}_2, \bar{O}_1) \text{ (by definition)} \tag{6-20}$$

$$\text{Property \#2: } 0 \leq \sigma(\bar{O}_1, \bar{O}_2) \leq 1 \tag{6-21}$$

$$\text{Property \#3: } \sigma(\bar{O}_1, \bar{O}_2) = 1 \text{ if (a) } \lambda_1 \sim \lambda_2 \text{ or (b) } \bar{O}_1 = \bar{O}_2 \tag{6-22}$$

Below, we illustrate the behavior of the similarity measure for some simple HMMs. First, for the class of single-state, discrete HMMs given by,

2. Theoretically, the Baum-Welch algorithm guarantees that P_{ii} is a local maximum only. In practice, this is not a significant concern, however, since the Baum-Welch algorithm converges to near-optimal solutions for discrete-output HMMs, when the algorithm is initialized with random model parameters [94, 95]. We have verified this near-optimal convergence property experimentally in two ways. First, for a given set of observation sequences $\bar{O}$, we trained n different HMMs $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ from different initial random parameter settings. We then observed that the probabilities $P(\bar{O}|\lambda_i)$, $i \in \{1, 2, \ldots, n\}$, were approximately equivalent. Second, for a given model λ, we *generated* a set of observation sequences $\hat{O}$. We then trained a second Hidden Markov Model $\hat{\lambda}$ (with initial random model parameters) on $\hat{O}$. Finally, we observed that $P(\hat{O}|\hat{\lambda})/P(\hat{O}|\lambda) \approx 1$ provided that $\hat{O}$ is sufficiently large. Both procedures suggest that the Baum-Welch algorithm does indeed converge to optimal or near-optimal solutions in practice.

3. Note that without condition (6-16), equation (6-19) does not necessarily hold.

$$\lambda_j = \{A_j, B_j, \pi_j\} = \left\{ \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} b_{j1} & b_{j2} & \dots & b_{jL} \end{bmatrix}^T, \begin{bmatrix} 1 \end{bmatrix} \right\}, \tag{6-23}$$

the similarity measure reduces to[4],

$$\sigma(\bar{O}_1, \bar{O}_2) = \prod_{k=1}^{L} \left(\frac{b_{1k}}{b_{2k}}\right)^{\frac{(b_{2k} - b_{1k})}{2}} \tag{6-24}$$

which reaches a maximum when $b_{1k} = b_{2k}$, or simply, $B_1 = B_2$, and that maximum is equal to one. Figure 6-5 shows a contour plot for, $B_1 = \begin{bmatrix} p_1 & 1 - p_1 \end{bmatrix}^T$, $B_2 = \begin{bmatrix} p_2 & 1 - p_2 \end{bmatrix}^T$, and $0 < p_1, p_2 < 1$.

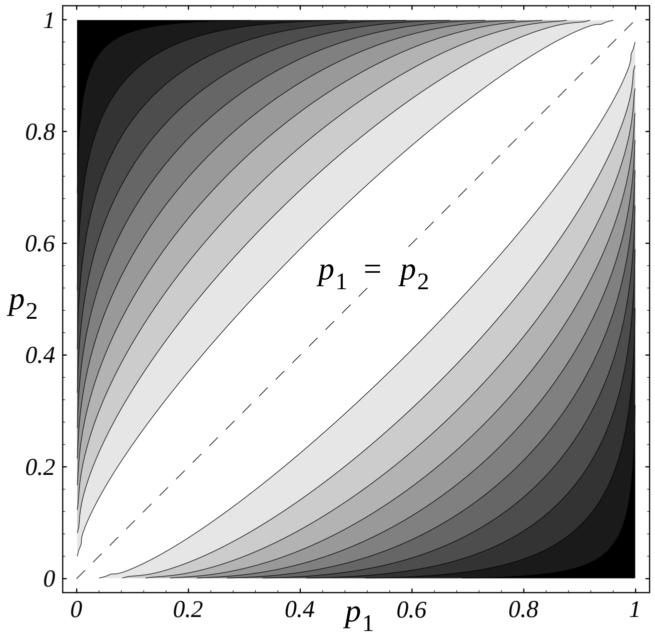


Figure 6-5: Similarity measure for two binomial distributions. Lighter colors indicate higher similarity.

4. $$P_{ij} = \left(\prod_{k=1}^{L} (b_{jk})^{(\bar{T}_i b_{ik})} \right)^{1/\bar{T}_i} = \prod_{k=1}^{L} (b_{jk})^{b_{ik}}$$

$$\sigma(\bar{O}_1, \bar{O}_2) = \sqrt{\frac{P_{21}P_{12}}{P_{11}P_{22}}} = \left[\frac{\left(\prod_{k=1}^{L} (b_{1k})^{b_{2k}} \right)\left(\prod_{k=1}^{L} (b_{2k})^{b_{1k}} \right)}{\left(\prod_{k=1}^{L} (b_{1k})^{b_{1k}} \right)\left(\prod_{k=1}^{L} (b_{2k})^{b_{2k}} \right)} \right]^{\frac{1}{2}} = \prod_{k=1}^{L} \left(\frac{b_{1k}}{b_{2k}}\right)^{\frac{b_{2k}}{2}} \prod_{k=1}^{L} \left(\frac{b_{2k}}{b_{1k}}\right)^{\frac{b_{1k}}{2}} = \prod_{k=1}^{L} \left(\frac{b_{1k}}{b_{2k}}\right)^{\frac{(b_{2k} - b_{1k})}{2}}$$

Second, we give an example of how the proposed similarity measure changes, not as a function of different symbol distributions, but rather as a function of varying HMM structure. Consider the following Hidden Markov Model,

$$\lambda(\alpha) = \left\{ \begin{bmatrix} \frac{1+\alpha}{2} & \frac{1-\alpha}{2} \\ \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T \right\}, \ 0 \le \alpha < 1 \tag{6-25}$$

and corresponding observation sequences, $O(\alpha)$, stochastically generated from model $\lambda(\alpha)$. For all $\alpha \in [0, 1)$, $O(\alpha)$ will have an equivalent aggregate distribution of symbols 0 and 1 — namely 1/2 and 1/2. As α increases, however, $O(\alpha)$ will become increasingly structured. For example,

$$\lambda(0) = \left\{ \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T, \begin{bmatrix} 1 \end{bmatrix} \right\} \text{ (equivalent to unbiased coin toss)} \tag{6-26}$$

$$\lim_{\alpha \to 1} O(\alpha) = \{\ldots, 1, 1, 1, 0, 0, 0, \ldots, 0, 0, 0, 1, 1, 1, \ldots\} \tag{6-27}$$

Figure 6-6 graphs $\sigma[O(\alpha_1), O(\alpha_2)]$ as a contour plot for $0 \le \alpha_1, \alpha_2 < 1$, where each observation sequence $O(\alpha)$ of length $T = 10,000$ is generated stochastically from the corresponding HMM $\lambda(\alpha)$[5]. Greatest similarity is indicated for $\alpha_1 = \alpha_2$, while greatest dissimilarity occurs for $(\alpha_1 \to 1, \alpha_2 = 0)$, and $(\alpha_1 = 0, \alpha_2 \to 1)$.

6.2.4 Distance measure

In some cases, it may be more convenient to represent the similarity between two sets of observation sequences through a *distance measure* $d(\bar{O}_1, \bar{O}_2)$, rather than a similarity measure. Given the similarity measure $\sigma(\bar{O}_1, \bar{O}_2)$, such a measure is easily derived. Let,

5. This procedure only approximates our similarity measure definition, since $\lambda(\alpha)$ is only optimal for $O(\alpha)$ as $T \to \infty$.

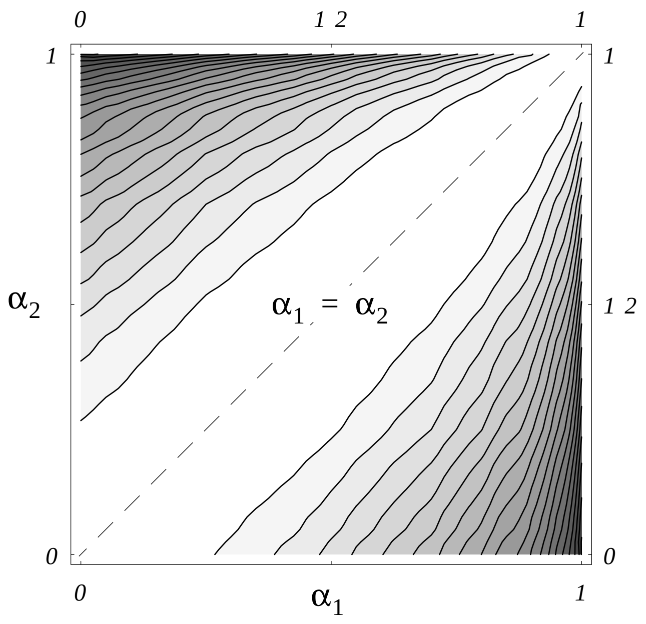


Figure 6-6: The similarity measure changes predictably as a function of HMM structure.

$$d(\bar{O}_1, \bar{O}_2) = -\log\sigma(\bar{O}_1, \bar{O}_2) = \frac{1}{2}[\log(P_{11}P_{22}) - \log(P_{21}P_{12})] \quad (6\text{-}28)$$

such that,

$$d(\bar{O}_1, \bar{O}_2) = d(\bar{O}_2, \bar{O}_1), \quad (6\text{-}29)$$

$$d(\bar{O}_1, \bar{O}_2) \geq 0, \quad (6\text{-}30)$$

$$d(\bar{O}_1, \bar{O}_2) = 0 \text{ if (a) } \lambda_1 \sim \lambda_2 \text{ or (b) } \bar{O}_1 = \bar{O}_2. \quad (6\text{-}31)$$

The distance measure $d(\bar{O}_1, \bar{O}_2)$ *between two sets of observation sequences* defined in (6-28) is closely related to the dual notion of distance *between two Hidden Markov Models*, as proposed in [55].

Let $\hat{O}_i$ denote a set of random observation sequences of total length $\hat{T}_i$ *generated by* the HMM $\hat{\lambda}_i$, and let,

$$\hat{P}_{ij} = P(\hat{O}_i|\hat{\lambda}_i)^{1/\hat{T}_i} \tag{6-32}$$

Then, [55] defines the following distance measure between two Hidden Markov Models, $\hat{\lambda}_1$ and $\hat{\lambda}_2$:

$$d(\hat{\lambda}_1, \hat{\lambda}_2) = \frac{1}{2}[\log(\hat{P}_{11}\hat{P}_{22}) - \log(\hat{P}_{21}\hat{P}_{12})] \tag{6-33}$$

Unlike the observation sequences $\bar{O}_i$, the sequences $\hat{O}_i$ are not unique, since they are stochastically generated from $\hat{\lambda}_i$. Hence, $d(\hat{\lambda}_1, \hat{\lambda}_2)$ is uniquely determined only in the limit as $\hat{T}_i \to \infty$. Likewise for $d(\bar{O}_1, \bar{O}_2)$, the HMMs λ_1 and λ_2 are not unique, since P_{11} and P_{22} are in general guaranteed to be only local, not global maxima. Hence, $d(\bar{O}_1, \bar{O}_2)$ is uniquely determined only when P_{11} and P_{22} represent global maxima.

While in general, $d(\hat{\lambda}_1, \hat{\lambda}_2)$ and $d(\bar{O}_1, \bar{O}_2)$ are not equivalent, the discussion above suggests sufficient conditions for which the two notions — distance between HMMs and distance between observation sequences — do converge to equivalence. Specifically, $d(\bar{O}_1, \bar{O}_2) = d(\hat{\lambda}_1, \hat{\lambda}_2)$ if,

(1) $\lambda_1 \sim \hat{\lambda}_1, \lambda_2 \sim \hat{\lambda}_2$, (6-34)

(2) P_{11}, P_{22} are global maxima, and (6-35)

(3) $\hat{T}_i \to \infty$. (6-36)

6.2.5 Data preprocessing

Assume that we wish to analyze the similarity of N control data sets, each of which is either a human control data set or a model-generated data set. Denote these data sets as $X^n = \begin{bmatrix}\bar{x}_1^n & \bar{x}_2^n & \dots & \bar{x}_D^n\end{bmatrix}$, $n \in \{1, 2, \dots, N\}$, where,

$$\bar{x}_d^n = \begin{bmatrix}x_{1d}^n & x_{2d}^n & \dots & x_{t_i d}^n\end{bmatrix}^T, \ d \in \{1, 2, \dots, D\}, \tag{6-37}$$

denotes the *d*th t_n-length column vector for data set X^n. Since we use discrete-output HMMs in our similarity measure, we need to convert these multi-dimensional, real-valued data sets to sequences of discrete symbols O_n. We follow two steps in this conversion: (1) data preprocessing and (2) vector quantization, as illustrated in Figure 6-7. The primary purpose of the data preprocessing (described below) is to extract meaningful feature vectors for the vector quantizer. For our case, the preprocessing proceeds in three steps: (1) normalization, (2) spectral conversion, and (3) power spectral density (PSD) estimation.

In the normalization step, we want to scale the columns in each data set, so that each dimension takes on the same range of values, namely $[-1, 1]$. Note that the scale factor for a given dimension has to be the same across data sets X^n. Let,

$$U = N^m(X, \bar{s}) = \left[(\bar{x}_1/s_1)\ (\bar{x}_2/s_2)\ \ldots\ (\bar{x}_D/s_D)\right] \tag{6-38}$$

define a matrix-to-matrix *normalization* transform for a $t \times D$ matrix X and a *D*-length scale vector,

$$\bar{s} = \left[s_1\ s_2\ \ldots\ s_D\right]^T,\ s_d > 0,\ d \in \{1, 2, \ldots, D\}. \tag{6-39}$$

To perform the normalization on our data sets X^n, we choose the $\bar{s}$ vector,

$$s_d = \max_{\forall n,\, t} \left|x^n_{td}\right| \qquad ,\ d \in \{1, 2, \ldots, D\}, \tag{6-40}$$

such that the normalized data sets U^n,

$$U^n = N^m(X^n, \bar{s}) = \left[\bar{u}^n_1\ \bar{u}^n_2\ \ldots\ \bar{u}^n_D\right],\ n \in \{1, 2, \ldots, N\}, \tag{6-41}$$

$$\bar{u}^n_d = \left[u^n_{1d}\ u^n_{2d}\ \ldots\ u^n_{t_i d}\right]^T,\ d \in \{1, 2, \ldots, D\}, \tag{6-42}$$

satisfy,

$$\left|u^n_{td}\right| \le 1,\ \forall n, d, t. \tag{6-43}$$

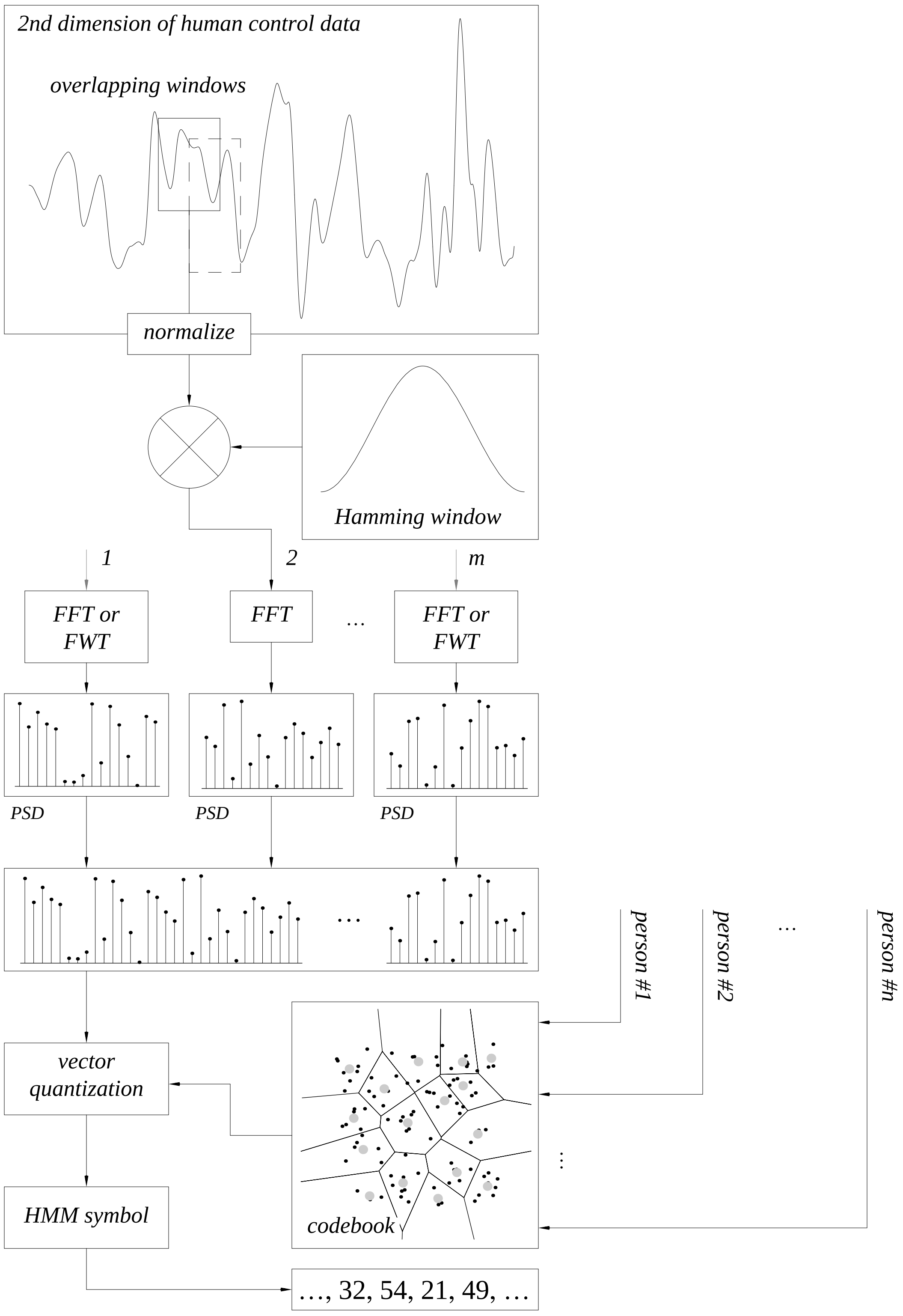


Figure 6-7: Conversion of multi-dimensional human control data to a sequence of discrete symbols.

After normalization, we perform spectral conversion on the columns of the normalized data sets U^n. For each column, we segment the data into possibly overlapping window frames, and apply either the Discrete Fourier Transform (DFT) or the Discrete Walsh Transform (DWT) to each frame[6].

The *Discrete Fourier Transform* $T_F^v(\cdot)$ maps a *k*-length real vector $\bar{y} = \left[y_1\ y_2\ \dots\ y_k\right]^T$ to a *k*-length complex vector $\bar{z}$ and is defined as,

$$\bar{z} = T_F^v(\bar{y}) = \left[F_0(\bar{y})\ F_1(\bar{y})\ \dots\ F_{k-1}(\bar{y})\right]^T, \text{ where} \tag{6-44}$$

$$F_p(\bar{y}) = \sum_{q=0}^{k-1} y_{q+1} e^{2\pi i p q / k},\ p \in \{0, 1, \dots, k-1\}. \tag{6-45}$$

Prior to applying the Fourier transform, we filter each frame through a Hamming window in order to minimize spectral leakage caused by the data windowing [91]. The *Hamming* transform $T_H^v(\cdot)$ maps a *k*-length real vector $\bar{y} = \left[y_1\ y_2\ \dots\ y_k\right]^T$ to a *k*-length real vector $\bar{h}$ and is defined as,

$$\bar{h} = T_H^v(\bar{y}) = \left[H_1 y_1\ H_2 y_2\ \dots\ H_k y_k\right], \text{ where} \tag{6-46}$$

$$H_p = 0.54 - 0.46\cos\left[\frac{2\pi(p-1)}{k-1}\right],\ p \in \{1, 2, \dots, k\} \text{ (see Figure 6-7)} \tag{6-47}$$

For notational convenience let $T_{HF}^v(\bar{y}) = T_F^v[T_H^v(\bar{y})]$.

Instead of sinusoidal basis functions, the *Discrete Walsh Transform* decomposes a signal based on the orthonormal *Walsh functions* [97]. The first eight Walsh-ordered Walsh functions are shown in Figure 6-8(a). In Figure 6-8(b), we show an example of human control data which can be characterized better through the Walsh transform, rather than the Fourier transform, due to

6. In practice, we calculate the DFT and DWT through the fast Fourier transform (FFT) and the fast Walsh transform (FWT), the $O(k \log k)$ algorithmic counterparts of the DFT and DWT, respectively. This restricts k to be of the form 2^m, $m \in \{1, 2, \dots\}$.

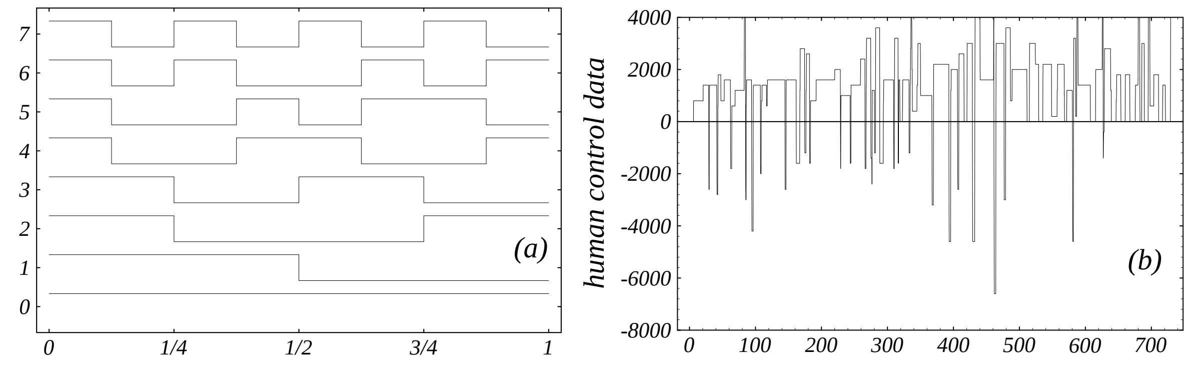


Figure 6-8: (a) The first eight Walsh-ordered Walsh functions, and (b) some sample human control data.

its discontinuous profile. Consider, for example, the power spectral densities (PSDs) for the square wave in Figure 6-9(a). The Walsh PSD in Figure 6-9(b) is a more concise feature vector than the corresponding Fourier PSD in Figure 6-9(c).

The *Discrete Walsh Transform* (DWT) $T_W^v(\cdot)$ maps a *k*-length real vector $\bar{y} = \begin{bmatrix} y_1 \; y_2 \; \dots \; y_k \end{bmatrix}^T$ to a *k*-length real vector $\bar{w}$ and is defined as,

$$T_W^v(\bar{y}) = \begin{bmatrix} W_0(\bar{y}) \; W_1(\bar{y}) \; \dots \; W_k(\bar{y}) \end{bmatrix}^T \text{, where,} \tag{6-48}$$

$$W_p(\bar{x}) = \sum_{q=0}^{k-1} y_{q+1}\omega(q,p) \text{, } p \in \{0, 1, \dots, k-1\} \text{,} \tag{6-49}$$

$$\omega(q,p) = (-1)^{\left\{ b(q,k-1)b(p,0) + \sum_{i=1}^{k-1} [b(q,k-i) + b(q,k-i-1)]b(p,i) \right\}} \text{, and} \tag{6-50}$$

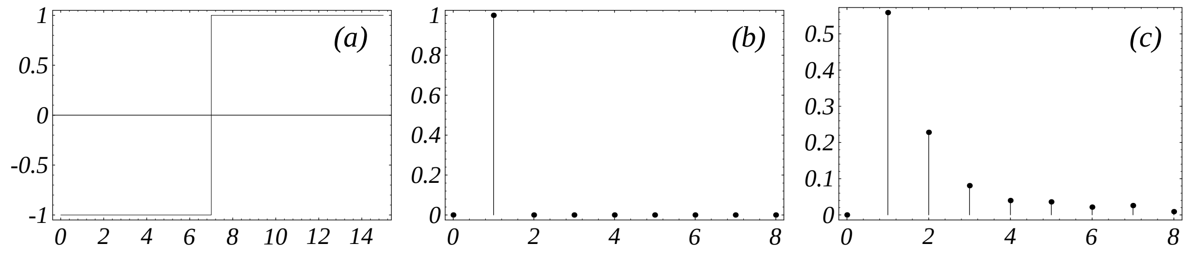


Figure 6-9: (a) Sample square wave and it’s corresponding (b) Walsh and (c) Fourier PSD’s.

$b(p, i)$ is the ith bit in the binary representation of p. (6-51)

Now, let us define the power spectral density (PSD) estimates for the Hamming-Fourier ($T^v_{HF}(\cdot)$) and Walsh transforms ($T^v_W(\cdot)$). For $\bar{f} = T^v_{HF}(\bar{y}) = \left[f_0\ f_1\ \dots\ f_{k-1}\right]^T$, the Fourier PSD is given by,

$$P^v_{HF}(\bar{f}) = \left[P_{HF0}(\bar{f})\ P_{HF1}(\bar{f})\ \dots\ P_{HF(k/2)}(\bar{f})\right] \text{, where,} \tag{6-52}$$

$$P_{HF0}(\bar{f}) = \frac{1}{H_{ss}}|f_0|^2, \tag{6-53}$$

$$P_{HFp}(\bar{f}) = \frac{1}{H_{ss}}(|f_p|^2 + |f_{k-p}|^2),\ p \in \{1, 2, \dots, k/2 - 1\}, \tag{6-54}$$

$$P_{HF(k/2)}(\bar{f}) = \frac{1}{H_{ss}}|f_{(k/2)}|^2 \text{, and} \tag{6-55}$$

$$H_{ss} = k \sum_{q=1}^{k} H_k^2 \tag{6-56}$$

For $\bar{w} = T^v_W(\bar{y}) = \left[w_0\ w_1\ \dots\ w_{k-1}\right]^T$, the Walsh PSD is given by,

$$P^v_W(\bar{w}) = \left[P_{W0}(\bar{w})\ P_{W1}(\bar{w})\ \dots\ P_{W(k/2)}(\bar{w})\right] \text{, where,} \tag{6-57}$$

$$P_{W0}(\bar{w}) = |w_0|^2, \tag{6-58}$$

$$P_{Wp}(\bar{w}) = |w_{2p-1}|^2 + |w_{2p}|^2,\ p \in \{1, 2, \dots, k/2 - 1\}, \tag{6-59}$$

$$P_{W(k/2)}(\bar{w}) = |w_{(k/2)}|^2. \tag{6-60}$$

Finally, for notational convenience, define two unity transforms,

$$P^v_G(\bar{y}) = T^v_G(\bar{y}) = \bar{y}, \tag{6-61}$$

and let,

$$\bar{y}_{[\tau,k]} = \begin{bmatrix} y_\tau & y_{\tau+1} & \dots & y^i_{\tau+k-1} \end{bmatrix}^T \tag{6-62}$$

be the k-length segment, beginning at element τ, for the t-length vector $\bar{y}$. Using equation (6-62), let us define the vector-to-matrix transform $T^{vm}_{(\varphi,[k_1,k_2])}(\cdot)$,

$$T^{vm}_{(\varphi,[k_1,k_2])}(\bar{y}) = \begin{bmatrix} P^v_\varphi\{T^v_\varphi(\bar{y}_{[1,k_1]})\} \\ P^v_\varphi\{T^v_\varphi(\bar{y}_{[k_2+1,k_1]})\} \\ P^v_\varphi\{T^v_\varphi(\bar{y}_{[2k_2+1,k_1]})\} \\ \vdots \end{bmatrix}, \ \varphi \in \{F, HF, W, G\}. \tag{6-63}$$

Furthermore, given a matrix $U = \begin{bmatrix} \bar{u}_1 & \bar{u}_2 & \dots & \bar{u}_d \end{bmatrix}$, let us define the matrix-to-matrix transform $T^{mm}_{(\bar{\varphi},[k_1,k_2])}(\cdot)$,

$$T^{mm}_{(\bar{\varphi},[k_1,k_2])}(U) = \begin{bmatrix} T^{vm}_{(\varphi_1,[k_1,k_2])}(\bar{u}_1) & T^{vm}_{(\varphi_2,[k_1,k_2])}(\bar{u}_2) & \dots & T^{vm}_{(\varphi_D,[k_1,k_2])}(\bar{u}_D) \end{bmatrix},$$
$$\bar{\varphi} = \begin{bmatrix} \varphi_1 & \varphi_2 & \dots & \varphi_D \end{bmatrix}, \ \varphi_d \in \{F, HF, W, G\}, \ d \in \{1, 2, \dots, D\} \tag{6-64}$$

The spectral conversion and PSD estimation can now be concisely expressed as,

$$V^n = T^{mm}_{(\bar{\varphi},[\kappa_1,\kappa_2])}(U^n), \tag{6-65}$$

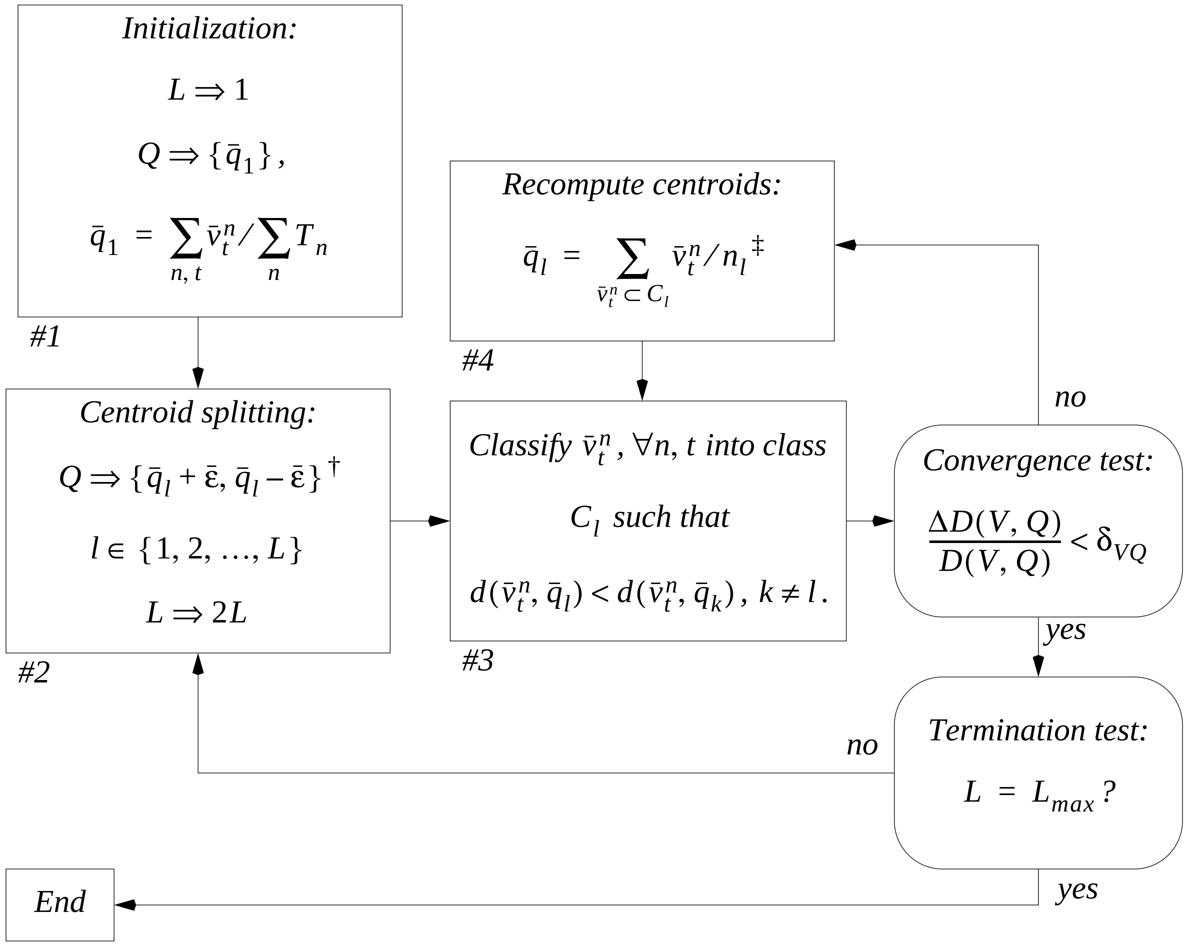
where the input matrix U^n has dimensions $t_n \times D$, and the output matrix V^n has dimensions $T_n \times K$,

$$T_n = \mathrm{floor}\left(\frac{t_n - \kappa_1}{\kappa_2} + 1\right), \ K = \sum_{\varphi_d \in \{F, HF, W\}} (\kappa_1/2 + 1) + \sum_{\varphi_d = G} \kappa_1 . \tag{6-66}$$

The integer constants $\kappa_1 \ge \kappa_2 > 0$, define the length of each window frame as κ_1 and the window overlap as $\kappa_2 - \kappa_1$. Furthermore, the transformation vector $\bar{\varphi}$ selects which transform to apply to each dimension of the control data. Generally, we select the Fourier PSD for *state trajectories*, and the Walsh PSD for *command trajectories*, since these trajectories tend to be non-smooth and, in part, discontinuous (see Appendix A).

6.2.6 Vector quantization

In the previous section, we define the transformation from the data sets X^n to the feature matrices V^n. Let, $V = \{\bar{v}_t^n\}$, $t \in \{1, 2, \ldots, T_n\}$, $n \in \{1, 2, \ldots, N\}$, denote the set of all feature vectors, where $\bar{v}_t^n$ is the *j*th row of the *i*th feature matrix. In order to apply discrete-output HMMs, we now need to convert the feature vectors V to L discrete symbols, where L is the number of output observables in our HMM models. In other words, we want to replace the many $\bar{v}_t^n$ with L prototype vectors $Q_L = \{\bar{q}_l\}$, $l \in \{1, 2, \ldots, L\}$, known as the *codebook*, such that we minimize the total distortion $D(V, Q_L)$,



Initialization:
$L \Rightarrow 1$
$Q \Rightarrow \{\bar{q}_1\}$,
$\bar{q}_1 = \sum_{n,t} \bar{v}_t^n / \sum_n T_n$
#1
Recompute centroids:
$\bar{q}_l = \sum_{\bar{v}_t^n \subset C_l} \bar{v}_t^n / n_l$ ‡
#4
no
Centroid splitting:
$Q \Rightarrow \{\bar{q}_l + \bar{\varepsilon}, \bar{q}_l - \bar{\varepsilon}\}$ †
$l \in \{1, 2, \ldots, L\}$
$L \Rightarrow 2L$
#2
Classify $\bar{v}_t^n$, $\forall n, t$ into class C_l such that $d(\bar{v}_t^n, \bar{q}_l) < d(\bar{v}_t^n, \bar{q}_k)$, $k \neq l$.
#3
Convergence test:
$\frac{\Delta D(V, Q)}{D(V, Q)} < \delta_{VQ}$
yes
no
Termination test:
$L = L_{max}$?
yes
End

† $\bar{\varepsilon} = [\varepsilon \ \ldots \ \varepsilon]^T$ ‡ n_l *is the total number of* $\bar{v}_t^n$ *in class* C_l.

Figure 6-10: The LBG VQ algorithm generates VQ codebooks of increasing size.

$$D(V, Q_L) = \sum_{n,t} \min_l d(\bar{v}_t^n, \bar{q}_l), \text{ where } d(\bar{v}_t^n, \bar{q}_l) = (\bar{q}_l - \bar{v}_t^n) \cdot (\bar{q}_l - \bar{v}_t^n), \tag{6-67}$$

over all feature vectors. We choose the well known LBG vector quantization (VQ) algorithm [69] to perform this quantization. Figure 6-10 illustrates the algorithm, which generates codebooks of size $L = 2^m$, $m \in \{0, 1, 2, \ldots\}$, and can be stopped at an appropriate level of discretization given the amount of available data and the complexity of the system trajectories. For our data, we set the split offset $\varepsilon = 0.0001$ and the convergence criterion $\delta_{VQ} = 0.01$ [7]. With these parameter settings, the centroids $\{\bar{q}_l\}$ usually converge within only a few iterations of the #3-#4 loop in Figure 6-10. As an example, Figure 6-11 illustrates the LBQ vector quantization for some random 2D data and $L \in \{1, 2, 4, 8, 16, 32\}$, while Figure 6-11 illustrates the quick convergence of the algorithm after centroid splitting for the same data and $L = 4$.

Given a trained VQ codebook Q_L, we convert the feature vectors V^n to a sequence of discrete symbols $O_n = \{o_1^n, o_2^n, \ldots, o_{T_n}^n\}$,

$$O_n = T_{VQ}^m(V^n, Q_L) = \{T_{VQ}^v(\bar{v}_1^n, Q_L), T_{VQ}^v(\bar{v}_2^n, Q_L), \ldots, T_{VQ}^v(\bar{v}_{T_n}^n, Q_L)\}, \text{ where,} \tag{6-68}$$

$$o_t^n = T_{VQ}^v(\bar{v}_t^n, Q_L) = \text{index}[\min_l d(\bar{v}_t^n, \bar{q}_l)] \tag{6-69}$$

This completes the conversion from the multi-dimensional, real-valued data sets X^n to the discrete observation sequences $O_n = \{o_1^n, o_2^n, \ldots, o_{T_n}^n\}$. Combining equations (6-40), (6-65) and (6-68), we can summarize the signal-to-symbol conversion of the data sets X^n as,

$$O_n = T_{VQ}^m\{T_{(\bar{\varphi}, [\kappa_1, \kappa_2])}^{mm}[N^m(X^n, \bar{s})], Q_L\} = T_{all}(X^n, \bar{s}, \bar{\varphi}, [\kappa_1, \kappa_2], Q_L) \tag{6-70}$$

6.2.7 Discretization compensation

We have stated previously that we choose to use discrete-output HMMs in our similarity analysis because they involve *significantly* less computation in training than either continuous-out-

7. These parameters are selected to achieve low distortion levels, while minimizing the number of iterations of the VQ algorithm. After the last split of the codebook $L = L_{max}$, we set $\delta_{VQ} \to \delta_{VQ}/10$.

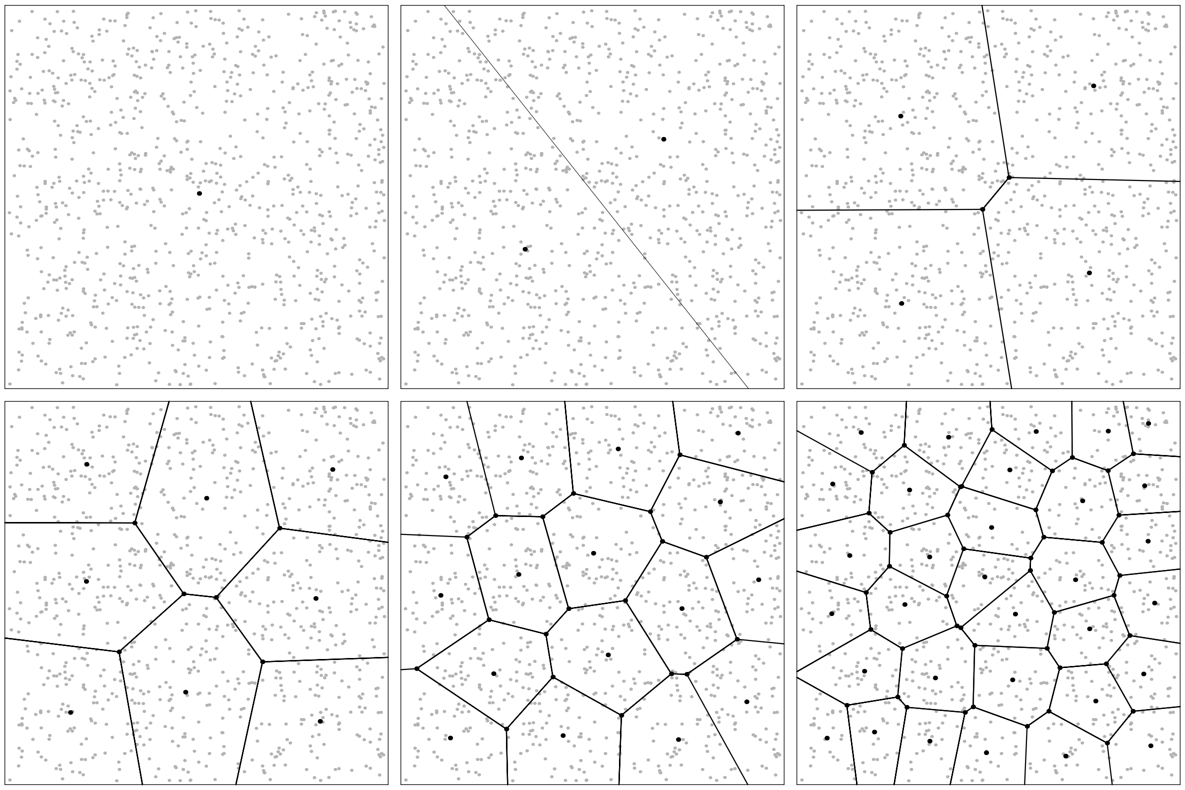


Figure 6-11: The LBG vector quantization for some random 2D data, as L equals 1, 2, 4, 8, 16 and 32.

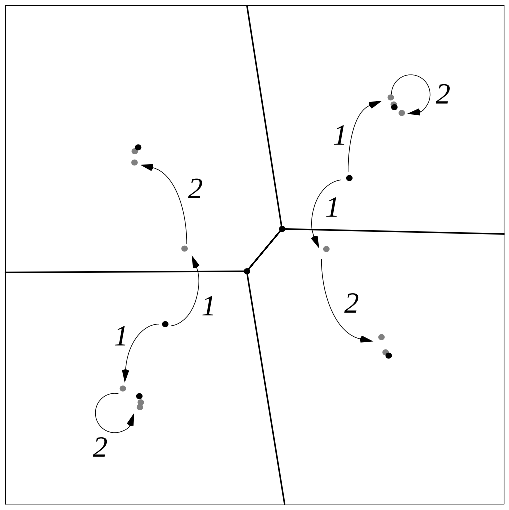


Figure 6-12: For a given codebook size L, the LBG algorithm converges in only a few iterations after centroid splitting.

put or semicontinuous-output HMMs. While computationally efficient, discretization of the output space, can have some negative consequences when analyzing real-valued data. Consider the following example.

Assume that we want to determine the similarity between two control data sets, X^1 and X^2. We follow the signal-to-symbol conversion procedure described in the previous two sections, and convert the data sets to discrete observation sequences O_1 and O_2,

$$O_k = \{o_t^k\},\ t \in \{1, 2, \ldots, T_k\},\ k \in \{1, 2\}. \quad (6\text{-}71)$$

We also train corresponding n-state HMMs, λ_1 and λ_2, where

$$A_k = \begin{bmatrix} a_{11}^k & a_{12}^k & \ldots & a_{1n}^k \\ a_{21}^k & a_{22}^k & \ldots & a_{2n}^k \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1}^k & a_{n2}^k & \ldots & a_{nn}^k \end{bmatrix},\ B_k = \begin{bmatrix} b_1^k(1) & b_2^k(1) & \ldots & b_n^k(1) \\ b_1^k(2) & b_2^k(2) & \ldots & b_n^k(2) \\ \vdots & \vdots & \vdots & \vdots \\ b_1^k(L) & b_2^k(L) & \ldots & b_n^k(L) \end{bmatrix},\ \pi_k = \begin{bmatrix} \pi_1^k \\ \pi_2^k \\ \vdots \\ \pi_n^k \end{bmatrix},\ k \in \{1, 2\}. \quad (6\text{-}72)$$

Now suppose that symbol l appears in the O_1 observation sequence (say at $t = \tau$), but does not appear in the O_2 observation sequence. This will force,

$$b_j^2(l) = 0,\ j \in \{1, 2, \ldots, n\}, \quad (6\text{-}73)$$

during the training of λ_2. Consequently, when we try to evaluate $P(O_1|\lambda_2)$ using the forward algorithm (Appendix B.1), we get,

$$\alpha_\tau(j) = \left[\sum_{i=1}^{n} \hat{\alpha}_{\tau-1}(i) a_{ij}^2\right] b_j^2(o_\tau) = \left[\sum_{i=1}^{n} \hat{\alpha}_{\tau-1}(i) a_{ij}^2\right] b_j^2(l) = 0, j \in \{1, 2, \ldots, n\} \quad (6\text{-}74)$$

$$c_\tau = 1 \Big/ \left(\sum_{i=1}^{n} \alpha_\tau(i)\right) \to \infty,\ P(O_1|\lambda_2) = 1 \Big/ \left(\prod_{t=1}^{T} c_t\right) \to 0,\ P_{12} \to 0 \quad (6\text{-}75)$$

$$\therefore \sigma(O_1, O_2) = 0. \quad (6\text{-}76)$$

Thus, the presence of a single observable l present in one observation sequence but not the other will force the similarity measure to be 0, even if the two observation sequences are identical in every other respect. This is not desirable, since the rogue observable l might be an event that is observed only rarely, or might even be the result of a measurement error or unintended control action on the part of the process that generated control trajectory X^1. Below, we consider two parameterized post-training solutions to this singularity problem within the context of discrete-output HMMs: (1) flooring and (2) semicontinuous evaluation.

Flooring [94] defines the common practice of replacing nonzero elements in the trained HMMs by some small value $\rho > 0$ and then renormalizing the rows of A and the columns of B to satisfy the probabilistic constraints in equations (6-87) and (6-88). If there are m zero elements in a probability vector (i.e. a row of A or a column of B), this methods redistributes $(\rho m)/(1+\rho m)$ of the total probability mass to the zero elements.

Semicontinuous evaluation [51] redefines the forward algorithm. Let $O = \{o_t\}$ denote a discrete observation sequence that has been vector quantized from a sequence of real vectors $V = \{\bar{v}_t\}$, $t \in \{1, 2, \ldots, T\}$, and a VQ codebook $Q = \{\bar{q}_l\}$, $l \in \{1, 2, \ldots, L\}$. For discrete evaluation on a Hidden Markov Model λ, $P(O|\lambda)$ is computed using the forward algorithm,

$$\alpha_1(i) = \pi_i b_i(o_1),\ i \in \{1, 2, \ldots, n\} \tag{6-77}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{n} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}),\ t \in \{1, 2, \ldots, T-1\},\ j \in \{1, 2, \ldots, n\} \tag{6-78}$$

$$P(O|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \tag{6-79}$$

Semicontinuous evaluation proceeds almost identically, except that the $b_j(o_t)$ terms are replaced by $\tilde{b}_j(\bar{v}_t)$ terms,

$$\tilde{b}_j(\bar{v}_t) = \sum_{l=1}^{L} p(\bar{v}_t | C_l) b_j(l), \tag{6-80}$$

$$p(\bar{v}_t | C_l) = \exp\left[-\frac{1}{\sigma^2}(\bar{q}_l - \bar{v}_t) \cdot (\bar{q}_l - \bar{v}_t)\right], \tag{6-81}$$

where $p(\bar{v}_t | C_l)$ represents the estimated conditional probability density function that vector $\bar{v}_t$ belongs to class C_l, corresponding to the codebook vector $\bar{q}_l$, and σ is a user-defined smoothing parameter. Thus, in semicontinuous evaluation, we view the codebook vectors as the peaks of Gaussian distributions with uniform variances σ^2. The complete forward algorithm (without scaling) is given by,

$$\alpha_1(i) = \pi_i b_i(\bar{v}_1), \; i \in \{1, 2, \ldots, n\} \tag{6-82}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{n} \alpha_t(i) a_{ij}\right] \tilde{b}_j(\bar{v}_{t+1}), \; t \in \{1, 2, \ldots, T-1\}, \; j \in \{1, 2, \ldots, n\} \tag{6-83}$$

$$P(V|\lambda) = \sum_{i=1}^{n} \alpha_T(i), \text{ where } \lim_{\sigma \to 0} P(V|\lambda) = P(O|\lambda). \tag{6-84}$$

As an example, consider a single-state HMM λ with the discrete-output probability matrix B,

$$B = \begin{bmatrix} 0.0 & 0.5 & 0.3 & 0.2 & 0.0 \end{bmatrix}^T. \tag{6-85}$$

Also, let the discrete symbols l correspond to real-valued numbers v,

$$\frac{l-1}{5} \le v < \frac{l}{5}, \; l \in \{1, 2, 3, 4, 5\} \tag{6-86}$$

Figure 6-13 illustrates how the discrete probability density function (pdf) $p(v|\lambda)$ encoded by B is modified through flooring and semicontinuous evaluation, for specific values $\rho = 0.01$ and $\sigma^2 = 0.01$, respectively. Flooring of course maintains the discrete structure of the HMM, while semicontinuous evaluation smoothes between output classes.

We note that the smoothing of the output pdf achieved by semicontinuous evaluation is done so at a significant computational cost, in comparison to discrete evaluation. Assuming L output classes (i.e. symbols) and K dimensions for the $\bar{v}_t$ vectors, the computation of $\tilde{b}_j(\bar{v}_t)$ is $O(LK)$, while $b_j(\bar{o}_t)$ requires only one table lookup. Consequently, for typical values of L and K, the evaluation of $P(V|\lambda)$ will be orders of magnitude slower than the evaluation of $P(O|\lambda)$.

For the experiments in the next chapter, the similarity measure achieves roughly equivalent discrimination results with semicontinuous evaluation as with flooring. Therefore, because of the substantial computational burden of semicontinuous evaluation, unless otherwise noted, we choose flooring rather than semicontinuous evaluation to avoid the singularity problem, with $\rho = 0.0001$. For the HMMs in this thesis, this value of ρ redistributes less than 0.1% of the probability mass in the state transition matrices A, and less than 0.5% probability mass in the output probability matrices B.

6.2.8 HMM training

The last step of the similarity analysis involves training Hidden Markov Models λ corresponding to each observation sequence O. To do this we use the iterative Baum-Welch algorithm (see Appendix B). Throughout this thesis, we initialize the Baum-Welch algorithm by setting the

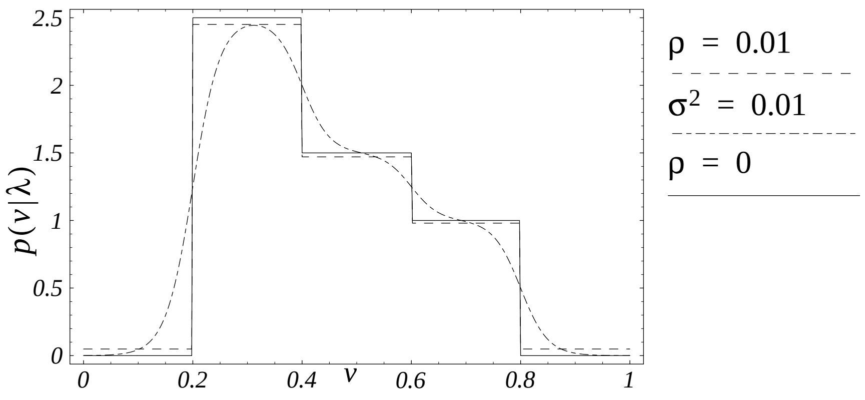


Figure 6-13: To avoid the singularity problem, the initial discrete pdf ($\rho = 0$) can be modified either through flooring $\rho = 0.01$ or semicontinuous evaluation $\sigma^2 = 0.01$.

Hidden Markov Model parameters to random, nonzero values, subject to the necessary probabilistic constraints,

$$\sum_{j=1}^{n_s} a_{ij} = 1, \; i \in \{1, 2, \ldots, n_s\}, \tag{6-87}$$

$$\sum_{l=1}^{L} b_j(l) = 1, \; j \in \{1, 2, \ldots, n_s\}, \tag{6-88}$$

where a_{ij} is the probability of transiting from state i to state j, and $b_j(l)$ is the probability of observing symbol l in state j, and n_s is the number of HMM states that we choose for the models λ. Let $\lambda^{(k)}$ denote the HMM λ after k iterations of the Baum-Welch algorithm, and let $\lambda^{(m)}$ denote the current iteration of Baum-Welch. Then, we stop training if,

$$\frac{P(O|\lambda^{(k)}) - P(O|\lambda^{(k-1)})}{P(O|\lambda^{(k)})} < \delta_{HMM}, \; k \in \{m, m-1, \ldots, m-4\}, \tag{6-89}$$

where $\delta_{HMM} = 0.000001$. This type of stringent convergence test is required, because in practice, the Baum-Welch algorithm frequently stalls over consecutive iterations. Figure 6-14, for example, plots $-\frac{1}{T}\log P(O|\lambda^{(k)})$ for some typical human data. We must be careful that we do not stop training on these types of plateaus if further improvements can be achieved. Otherwise, the assumption that the P_{ii}, defined in (6-14), represent near-optimal global maxima would be violated, along with properties #2 and #3 in equations (6-21) and (6-22), respectively.

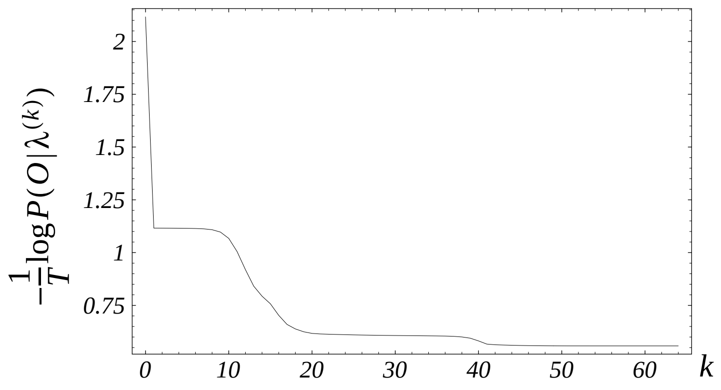


Figure 6-14: The Baum-Welch algorithm can stall over several iterations, until further improvements are realized.

Chapter 7

Human-to-human similarity

In this chapter we test the similarity measure proposed in Chapter 6 by comparing human control strategies across different individuals. We contrast the human-to-human classification results with the well-known Bayes classifier and simple spectral processing, and show that the similarity measure achieves significantly better performance than either of the alternative methods. These results confirm the similarity measure's usefulness as a model validation tool.

7.1 Comparing human control strategies

7.1.1 Experimental data

Appendix A describes driving control data from six different individuals — (1) Larry, (2) Curly, (3) Moe, (4) Groucho, (5) Harpo and (6) Zeppo across three different roads, roads #1, #2 and #3 in Figures 2-2(a), (b) and (c), respectively. For notational convenience, let $X^{(i,\,j)}$, $i \in \{1, 2, 3, 4, 5, 6\}$, $j \in \{1, 2, 3\}$, denote the run from person (i) on road #j, sampled at 50Hz.

Figure 7-1 plots the means and standard deviations for the two dimensions — the velocity v and the acceleration control ϕ — with the greatest variance between runs (see Table A-1). From this diagram, we observe that there is significant overlap in the data for runs from different individuals. Appendix A offers a more complete picture of each person's control data, where the velocity v, the lateral offset from the road median d_ξ, the steering control δ and the acceleration control ϕ over time are plotted for all 18 runs.

7.1.2 Classification experiments

Here, we investigate how well the similarity measure is able to classify the driving data $X^{(i,j)}$ belonging to the same individual, while discriminating driving data from different individuals.

Let $\bar{s}^j = \begin{bmatrix} s_1^j & s_2^j & \dots & s_5^j \end{bmatrix}$, where,

$$s_d^j = \max_{\forall i, t} \left| x_{td}^{(i,j)} \right|, \; d \in \{1, 2, \dots, D\}, \; j \in \{1, 2, 3\}. \tag{7-1}$$

Also let,

$$V_k^{(i,j)} = T^{mm}_{(\bar{\varphi}, [\kappa_1, \kappa_2])}[N^m(X^{(i,j)}, \bar{s}^k)], \; i \in \{1, 2, \dots, 6\}, \; j, k \in \{1, 2, 3\}, \tag{7-2}$$

be the feature vectors corresponding to data set $X^{(i,j)}$ and scale vector $\bar{s}^k$; let Q_L^j be an L-length VQ codebook trained on $\{V_j^{(i,j)}\}$, $i \in \{1, 2, \dots, 6\}$; and let,

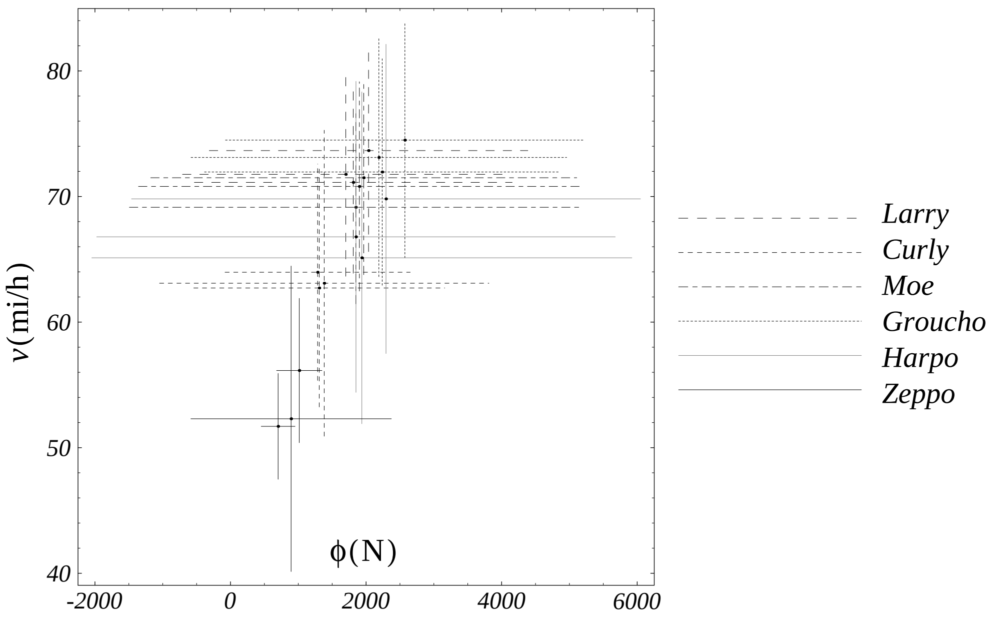


Figure 7-1: There is significant overlap between runs from different individuals.

$$O^k_{(i,j)} = O(i,j,k) = T_{all}(X^{(i,j)}, \bar{s}^k, \bar{\varphi}, [\kappa_1, \kappa_2], Q^k_L),\ i \in \{1, 2, \ldots, 6\},$$
$$j, k \in \{1, 2, 3\}. \qquad (7\text{-}3)$$

We view the observation sequences $O^k_{(i,k)}$ as *labeled* data, which has already been collected and processed without any prior information about $O^k_{(i,j)}$, $j \neq k$. Each $O^k_{(i,k)}$ represents a known class for individual i. Similarly, we view the observation sequences $O^k_{(i,j)}$, $j \neq k$ as *unlabeled* data which needs to be classified as belonging to one individual. This emulates the common classification scenario, where we have labeled data from which we define our classes (represented here by the sequences $O^k_{(i,k)}$), and unknown or unlabeled data to which we wish to assign a label (represented here by the sequences $O^k_{(i,j)}$, $j \neq k$).

We now use our similarity measure σ to perform this classification, where we consider $O^k_{(i,j)}$ to be classified correctly if and only if,

$$\sigma[O^k_{(i,k)}, O^k_{(i,j)}] > \sigma[O^k_{(l,k)}, O^k_{(i,j)}],\ \forall(j \neq k, l \neq i) \qquad (7\text{-}4)$$

In other words, we expect that two runs from the same individual will yield a higher similarity measure than two runs from two different individual.

The system trajectory for the driving task is defined by the three state trajectories $\{\bar{v}_\xi, \bar{v}_\eta, \bar{\omega}\}$ and the two control trajectories $\{\bar{\delta}, \bar{\phi}\}$. Hence, we let,

$$X^{(i,j)} = \left[\bar{v}_\xi\ \bar{v}_\eta\ \bar{\omega}\ \bar{\delta}\ \bar{\phi}\right]^{(i,j)},\ i \in \{1, 2, \ldots, 6\},\ j \in \{1, 2, 3\}. \qquad (7\text{-}5)$$

We select the following parameters:

$$\bar{\varphi} = \left[HF\ HF\ HF\ W\ W\right]^T\ ^{1},\ \kappa_1 = 16\ ^{2},\ \kappa_2 = \kappa_1/2,\ L = 128\ ^{3},\ n_s = 8. \qquad (7\text{-}6)$$

1. As we have noted before, the Walsh PSD is chosen for the command variables $\{\delta, \phi\}$, due their nonsmooth and discontinuous profile.
2. The transform length $\kappa_1 = 16$ represents a length of 0.32 seconds for our recording frequency of 50Hz.
3. The vector quantization level L is chosen so that the number of nonzero parameters in the trained Hidden Markov Models λ is much larger than the length $T_{(i,j)}$ of each observation sequence.

Tables 7-1, 7-2, and 7-3 report the similarity results for $O^k_{(i,k)}$, $k \in \{1, 2, 3\}$, respectively. Note that the largest similarity value is highlighted in each row, and that the similarity measure in fact classifies all 36 $O^k_{(i,j)}$, $j \neq k$, correctly.

Since the similarity measure achieves 100% correct classification, we would like to see the degree to which we can discriminate between individuals when — rather than include all five dimensions ($\{v_\xi, v_\eta, \omega\}$ and $\{\delta, \phi\}$) in the similarity analysis — we give the algorithm only a single dimension off which to classify the driving data. Figure 7-2 plots the correct-classification percentages for each dimension, as well as d_ξ, the lateral offset from the road median. The lateral offset d_ξ — for these data sets — does not discriminate nearly as well as the state and control variables included in the similarity analysis above. From qualitative observations of the simulated driving runs, we observed that individuals payed relatively little attention to their lateral position as long as they maintained contact with the road; it is therefore not unexpected that lateral road position would be a poor discriminator between individuals. Consequently, we chose not include this dimension in the similarity analysis.

7.1.3 Bayes classification

A legitimate question of course is whether or not a simpler statistical technique, like the Bayes optimal classifier [30], can achieve similar positive results. Let class $C_{(i,k)}$,

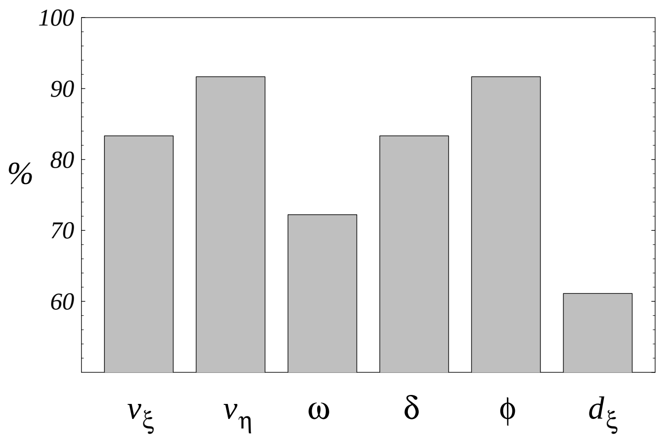


Figure 7-2: Single-dimensional correct classification percentages (36 classifications).

Table 7-1: Similarity results for road #1 data

σ	*O(1, 1, 1)*	*O(2, 1, 1)*	*O(3, 1, 1)*	*O(4, 1, 1)*	*O(5, 1, 1)*	*O(6, 1, 1)*
O(1, 2, 1)	**0.800**	0.288	0.399	0.325	0.125	0.034
O(1, 3, 1)	**0.720**	0.234	0.397	0.391	0.152	0.015
O(2, 2, 1)	0.272	**0.729**	0.131	0.102	0.090	0.326
O(2, 3, 1)	0.277	**0.531**	0.098	0.082	0.047	0.254
O(3, 2, 1)	0.381	0.220	**0.849**	0.368	0.288	0.013
O(3, 3, 1)	0.376	0.160	**0.754**	0.402	0.273	0.009
O(4, 2, 1)	0.253	0.233	0.445	**0.863**	0.364	0.016
O(4, 3, 1)	0.152	0.149	0.224	**0.756**	0.368	0.010
O(5, 2, 1)	0.044	0.076	0.207	0.302	**0.672**	0.006
O(5, 3, 1)	0.040	0.061	0.168	0.346	**0.651**	0.005
O(6, 2, 1)	0.003	0.036	0.003	0.003	0.004	**0.357**
O(6, 3, 1)	0.027	0.085	0.008	0.009	0.009	**0.580**

Table 7-2: Similarity results for road #2 data

σ	*O(1, 2, 2)*	*O(2, 2, 2)*	*O(3, 2, 2)*	*O(4, 2, 2)*	*O(5, 2, 2)*	*O(6, 2, 2)*
O(1, 1, 2)	**0.812**	0.338	0.390	0.257	0.046	0.002
O(1, 3, 2)	**0.774**	0.192	0.317	0.336	0.076	0.002
O(2, 1, 2)	0.285	**0.731**	0.223	0.257	0.079	0.034
O(2, 3, 2)	0.289	**0.733**	0.151	0.113	0.011	0.054
O(3, 1, 2)	0.423	0.152	**0.850**	0.412	0.182	0.003
O(3, 3, 2)	0.411	0.104	**0.784**	0.325	0.161	0.002
O(4, 1, 2)	0.338	0.127	0.359	**0.817**	0.298	0.003
O(4, 3, 2)	0.163	0.065	0.152	**0.673**	0.253	0.003
O(5, 1, 2)	0.135	0.093	0.307	0.368	**0.652**	0.007
O(5, 3, 2)	0.044	0.017	0.136	0.282	**0.689**	0.001
O(6, 1, 2)	0.027	0.321	0.013	0.018	0.006	**0.419**
O(6, 3, 2)	0.022	0.170	0.011	0.011	0.002	**0.540**

Table 7-3: Similarity results for road #3 data

σ	*O(1, 3, 3)*	*O(2, 3, 3)*	*O(3, 3, 3)*	*O(4, 3, 3)*	*O(5, 3, 3)*	*O(6, 3, 3)*
O(1, 1, 3)	**0.671**	0.339	0.396	0.169	0.054	0.029
O(1, 2, 3)	**0.782**	0.298	0.424	0.174	0.049	0.030
O(2, 1, 3)	0.228	**0.578**	0.174	0.138	0.069	0.091
O(2, 2, 3)	0.175	**0.739**	0.091	0.068	0.024	0.168
O(3, 1, 3)	0.422	0.108	**0.793**	0.241	0.170	0.010
O(3, 2, 3)	0.357	0.114	**0.768**	0.176	0.149	0.011
O(4, 1, 3)	0.392	0.091	0.384	**0.696**	0.344	0.011
O(4, 2, 3)	0.365	0.113	0.381	**0.652**	0.309	0.016
O(5, 1, 3)	0.171	0.061	0.311	0.314	**0.604**	0.013
O(5, 2, 3)	0.091	0.012	0.178	0.290	**0.690**	0.002
O(6, 1, 3)	0.013	0.320	0.008	0.014	0.011	**0.608**
O(6, 2, 3)	0.001	0.082	0.001	0.007	0.003	**0.631**

$$C_{(i,k)} = C(i,k) = \{\mu_{(i,k)}, \Sigma_{(i,k)}\},\ i \in \{1, 2, \ldots, 6\},\ k \in \{1, 2, 3\}, \tag{7-7}$$

correspond to run $X^{(i,k)}$, where $\mu_{(i,k)}$ is the mean vector for,

$$U^{(i,k)} = T^{mm}_{(\varphi, [\kappa_1, \kappa_2])}[N^m(X^{(i,j)}, \bar{s})], \tag{7-8}$$

and $\Sigma_{(i,k)}$ is the covariance matrix for $U^{(i,k)}$. Now, for each vector $\bar{u}_t^{(i,j)}$ and each class $C_{(l,k)}$, $j \neq k$, we can calculate [30],

$$P(C_{(l,k)} | \bar{u}_t^{(i,j)}) = \frac{p(\bar{u}_t^{(i,j)} | C_{(l,k)}) P(C_{(l,k)})}{p(\bar{u}_t^{(i,j)})},\ j \neq k, \tag{7-9}$$

(i.e. the probability that vector $\bar{u}_t^{(i,j)}$ belongs to class $C_{(l,k)}$), where $P(C_{(l,k)})$ is the prior probability of class $C_{(l,k)}$,

$$p(\bar{u}_t^{(i,j)}) = \sum_{k=1}^{6} p(\bar{u}_t^{(i,j)} | C_{(l,k)}) P(C_{(l,k)}), \tag{7-10}$$

is just a normalization factor, and,

$$p(\bar{u}_t^{(i,j)}|C_{(l,k)}) = \frac{1}{(2\pi)^{D/2}|\Sigma_{(l,k)}|^{1/2}}\exp\left[-\frac{1}{2}(\bar{u}_t^{(i,j)}-\mu_{(l,k)})^T\Sigma_{(l,k)}^{-1}(\bar{u}_t^{(i,j)}-\mu_{(l,k)})\right]. \tag{7-11}$$

Given equation (7-9), we define the Bayes classification measure,

$$\zeta_{(l,k)}[X^{(i,j)}] = \frac{1}{T_{(i,j)}}\sum_t P(C_{(l,k)}|\bar{u}_t^{(i,j)}),\ j \neq k, \tag{7-12}$$

for run $X^{(i,j)}$. The measure $\zeta_{(l,k)}$ gives the probability of class $C_{(l,k)}$ given $U^{(i,j)}$, averaged over all vectors $\bar{u}_t^{(i,j)}$. We consider run $X^{(i,j)}$ classified correctly if and only if,

$$\zeta_{(i,k)}[U^{(i,j)}] > \zeta_{(l,k)}[U^{(i,j)}],\ \forall(j \neq k, l \neq i). \tag{7-13}$$

Tables 7-4, 7-5, and 7-6 report Bayes classification results analogous to the similarity measure results in Tables 7-1, 7-2 and 7-3, respectively, for,

$$\bar{\varphi} = \begin{bmatrix} G & G & G & G & G \end{bmatrix}^T,\ \kappa_1 = \kappa_2 = 1, \tag{7-14}$$

assuming equal priors $P(C_{(l,k)})$. Note that the Bayes classifier misclassifies 9 out of 36, or 25% of all runs. An alternate Bayes classification criterion, which measures the percentage of vectors $\bar{u}_t^{(i,j)}$ that fall into class $C_{(l,k)}$ performs even worse, misclassifying 13 of 36, or 36% of all runs. The similarity measure (with 0% error) compares quite favorably to both these results.

Providing additional inputs to the Bayes classifier in the form of time histories $\kappa_1 > 1$ does not help its performance. Consider, for example, $X^{(3,2)}$ and $X^{(3,3)}$, which are badly misclassified as $C_{(1,1)}$ rather than $C_{(3,1)}$. Let Δ_ζ,

$$\Delta_\zeta = -\log\left\{\frac{\max_l(\zeta_{(l,k)}[U^{(i,j)}])}{\zeta_{(i,k)}[U^{(i,j)}]}\right\},\ l \neq i, \tag{7-15}$$

Table 7-4: Bayes classification results for road #1 data

ζ	*C(1, 1)*	*C(2, 1)*	*C(3, 1)*	*C(4, 1)*	*C(5, 1)*	*C(6, 1)*
X(1, 2)	**0.297**	0.165	0.189	0.122	0.146	0.081
X(1, 3)	**0.294**	0.142	0.197	0.158	0.158	0.051
X(2, 2)	0.210	0.223	0.119	0.059	0.116	**0.273**
X(2, 3)	0.217	0.219	0.114	0.069	0.096	**0.286**
X(3, 2)	**0.241**	0.165	0.201	0.122	0.212	0.061
X(3, 3)	**0.248**	0.143	0.211	0.153	0.201	0.043
X(4, 2)	0.167	0.112	0.178	**0.327**	0.175	0.040
X(4, 3)	0.114	0.072	0.143	**0.468**	0.180	0.023
X(5, 2)	0.139	0.156	0.149	0.188	**0.288**	0.079
X(5, 3)	0.119	0.109	0.147	**0.292**	0.286	0.046
X(6, 2)	0.030	0.194	0.021	0.011	0.059	**0.685**
X(6, 3)	0.081	0.217	0.056	0.050	0.072	**0.524**

Table 7-5: Bayes classification results for road #2 data

ζ	*C(1, 2)*	*C(2, 2)*	*C(3, 2)*	*C(4, 2)*	*C(5, 2)*	*C(6, 2)*
X(1, 1)	**0.297**	0.244	0.207	0.133	0.111	0.009
X(1, 3)	**0.323**	0.168	0.207	0.172	0.127	0.003
X(2, 1)	0.194	**0.344**	0.167	0.082	0.145	0.068
X(2, 3)	0.187	**0.356**	0.135	0.070	0.068	0.185
X(3, 1)	**0.252**	0.164	0.237	0.178	0.167	0.003
X(3, 3)	**0.270**	0.153	0.246	0.170	0.160	0.002
X(4, 1)	0.196	0.096	0.177	**0.345**	0.184	0.002
X(4, 3)	0.133	0.057	0.125	**0.463**	0.221	0.001
X(5, 1)	0.177	0.192	0.208	0.141	**0.280**	0.001
X(5, 3)	0.141	0.097	0.165	0.278	**0.319**	0.000
X(6, 1)	0.053	0.260	0.051	0.042	0.146	**0.448**
X(6, 3)	0.045	0.206	0.047	0.052	0.077	**0.572**

Table 7-6: Bayes classification results for road #3 data

ζ	*C(1, 3)*	*C(2, 3)*	*C(3, 3)*	*C(4, 3)*	*C(5, 3)*	*C(6, 3)*
X(1, 1)	0.248	**0.281**	0.225	0.084	0.104	0.059
X(1, 2)	0.262	**0.269**	0.237	0.081	0.104	0.048
X(2, 1)	0.155	**0.333**	0.181	0.058	0.140	0.133
X(2, 2)	0.134	**0.362**	0.154	0.037	0.093	0.220
X(3, 1)	0.249	0.152	**0.274**	0.125	0.171	0.028
X(3, 2)	0.242	0.172	**0.290**	0.090	0.171	0.034
X(4, 1)	0.231	0.082	0.214	**0.267**	0.194	0.011
X(4, 2)	0.213	0.094	0.209	**0.269**	0.199	0.016
X(5, 1)	0.196	0.159	0.243	0.107	**0.288**	0.007
X(5, 2)	0.178	0.114	0.220	0.142	**0.345**	0.001
X(6, 1)	0.025	0.231	0.031	0.021	0.121	**0.572**
X(6, 2)	0.004	0.154	0.005	0.002	0.013	**0.823**

define a discrimination measure, where $\Delta_\zeta < 0$ indicates misclassification, while $\Delta_\zeta > 0$ indicates correct classification. Figure 7-3 plots this measure for $i = 3$, $j \in \{2, 3\}$ and $k = 1$, as a function of time history lengths $(\kappa_1 = \kappa_2) \in \{1, 2, 3\}$. We see that providing time histories actually hurts, rather than helps performance.

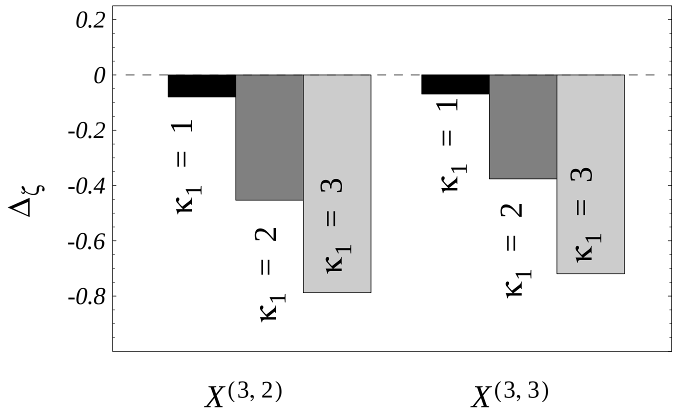


Figure 7-3: Providing time histories to the Bayes classifier hurts, rather than help performance.

7.1.4 Spectral classification

Another legitimate question is whether or not simple spectral processing on the data sets, such as the FFT, can achieve classification results as good as the HMM-based similarity measure. As an example let,

$$X^{*(i,j)} = \begin{bmatrix}\bar{x}_1{}^* & \bar{x}_2{}^* & \bar{x}_3{}^* & \bar{x}_4{}^* & \bar{x}_5{}^*\end{bmatrix}^{(i,j)} = \begin{bmatrix}\bar{v}_\xi & \bar{v}_\eta & \bar{\omega} & \bar{\delta} & \bar{\phi} \\ \bar{0} & \bar{0} & \bar{0} & \bar{0} & \bar{0}\end{bmatrix}^{(i,j)}, \; i \in \{1, 2, \ldots, 6\},$$

$$j \in \{1, 2, 3\} \tag{7-16}$$

be a $2^{16} \times 5$ matrix, where the upper $t_{(i,j)} \times 5$ submatrix is the *i*th person's run on road #*j*, and the $[2^{16} - t_{(i,j)}] \times 5$ lower submatrix is the zero matrix. Furthermore let,

$$U^{*(i,j)} = N^m(X^{*(i,j)}, \bar{s}), \tag{7-17}$$

where the scale vector $\bar{s}$ is taken over all data sets $X^{*(i,j)}$ (similar to (7-1)). Then define $\bar{v}^{*(i,j)}$, such that,

$$\bar{v}^{*(i,j)} = T^{mm}_{(\bar{\varphi}, [2^{16}, 2^{16}])}[N^m(X^{*(i,j)}, \bar{s}^k)], \; \bar{\varphi} = \begin{bmatrix}F & F & F & F & F\end{bmatrix}^T. \tag{7-18}$$

In other words, $\bar{v}^{*(i,j)}$ is a $2^{16} \times 5$ vector of FFT PSD coefficients for the normalized columns of $X^{*(i,j)}$. The original data runs $X^{(i,j)}$ are padded with ending zeros, so that each $X^{*(i,j)}$, and consequently each $\bar{v}^{*(i,j)}$ is of equal dimension. This allows us to define the following spectral distance measure,

$$d_s(\bar{v}^{*(i,j)}, \bar{v}^{*(p,q)}) = \sqrt{\frac{[\bar{v}^{*(i,j)} - \bar{v}^{*(p,q)}] \cdot [\bar{v}^{*(i,j)} - \bar{v}^{*(p,q)}]}{2^{16}}}, \tag{7-19}$$

and corresponding spectral similarity measure,

$$\sigma_s(\bar{v}^{*(i,j)}, \bar{v}^{*(p,q)}) = 10^{-d_s(\bar{v}^{*(i,j)}, \bar{v}^{*(p,q)})}. \tag{7-20}$$

Tables 7-7, 7-8 and 7-9 report spectral classification results — based on σ_s — analogous to the HMM-based similarity results (σ) in Tables 7-1, 7-2 and 7-3, respectively. Note that the spec-

Table 7-7: Spectral similarity results for road #1 data

σ_s	$\bar{v}*^{(1,1)}$	$\bar{v}*^{(2,1)}$	$\bar{v}*^{(3,1)}$	$\bar{v}*^{(4,1)}$	$\bar{v}*^{(5,1)}$	$\bar{v}*^{(6,1)}$
$\bar{v}*^{(1,2)}$	**0.689**	0.673	0.654	0.653	0.584	0.631
$\bar{v}*^{(1,3)}$	**0.696**	0.653	0.656	0.681	0.593	0.606
$\bar{v}*^{(2,2)}$	**0.695**	0.691	0.645	0.656	0.576	0.681
$\bar{v}*^{(2,3)}$	0.690	**0.693**	0.634	0.644	0.561	0.663
$\bar{v}*^{(3,2)}$	0.635	0.639	**0.644**	0.616	0.581	0.582
$\bar{v}*^{(3,3)}$	0.670	0.638	**0.671**	0.646	0.602	0.581
$\bar{v}*^{(4,2)}$	0.636	0.630	0.633	**0.654**	0.586	0.596
$\bar{v}*^{(4,3)}$	0.626	0.593	0.602	**0.665**	0.574	0.564
$\bar{v}*^{(5,2)}$	0.540	0.549	0.557	0.558	**0.566**	0.517
$\bar{v}*^{(5,3)}$	0.573	0.560	**0.597**	0.588	0.590	0.514
$\bar{v}*^{(6,2)}$	0.630	0.616	0.568	0.585	0.494	**0.690**
$\bar{v}*^{(6,3)}$	0.593	0.589	0.549	0.581	0.490	**0.661**

Table 7-8: Spectral similarity results for road #2 data

σ_s	$\bar{v}*^{(1,2)}$	$\bar{v}*^{(2,2)}$	$\bar{v}*^{(3,2)}$	$\bar{v}*^{(4,2)}$	$\bar{v}*^{(5,2)}$	$\bar{v}*^{(6,2)}$
$\bar{v}*^{(1,1)}$	0.689	**0.695**	0.635	0.636	0.540	0.630
$\bar{v}*^{(1,3)}$	0.675	**0.683**	0.639	0.654	0.563	0.604
$\bar{v}*^{(2,1)}$	0.673	**0.691**	0.639	0.630	0.549	0.616
$\bar{v}*^{(2,3)}$	0.689	**0.731**	0.620	0.632	0.524	0.681
$\bar{v}*^{(3,1)}$	**0.654**	0.645	0.644	0.633	0.557	0.568
$\bar{v}*^{(3,3)}$	**0.667**	0.651	0.657	0.637	0.561	0.564
$\bar{v}*^{(4,1)}$	0.653	**0.656**	0.616	0.654	0.558	0.585
$\bar{v}*^{(4,3)}$	0.622	0.614	0.597	**0.638**	0.557	0.554
$\bar{v}*^{(5,1)}$	0.584	0.576	0.581	**0.586**	0.566	0.494
$\bar{v}*^{(5,3)}$	0.572	0.558	**0.586**	0.585	0.555	0.486
$\bar{v}*^{(6,1)}$	0.631	0.681	0.582	0.596	0.517	**0.690**
$\bar{v}*^{(6,3)}$	0.592	0.638	0.537	0.574	0.481	**0.697**

Table 7-9: Spectral similarity results for road #3 data

σ_s	$\bar{v}*^{(1,3)}$	$\bar{v}*^{(2,3)}$	$\bar{v}*^{(3,3)}$	$\bar{v}*^{(4,3)}$	$\bar{v}*^{(5,3)}$	$\bar{v}*^{(6,3)}$
$\bar{v}*^{(1,1)}$	**0.696**	0.690	0.670	0.626	0.573	0.593
$\bar{v}*^{(1,2)}$	0.675	**0.689**	0.667	0.622	0.572	0.592
$\bar{v}*^{(2,1)}$	0.653	**0.693**	0.638	0.593	0.560	0.589
$\bar{v}*^{(2,2)}$	0.683	**0.731**	0.651	0.614	0.558	0.638
$\bar{v}*^{(3,2)}$	0.656	0.634	**0.671**	0.602	0.597	0.549
$\bar{v}*^{(3,3)}$	0.639	0.620	**0.657**	0.597	0.586	0.537
$\bar{v}*^{(4,2)}$	**0.681**	0.644	0.646	0.665	0.588	0.581
$\bar{v}*^{(4,3)}$	**0.654**	0.632	0.637	0.638	0.585	0.574
$\bar{v}*^{(5,2)}$	0.593	0.561	**0.602**	0.574	0.590	0.490
$\bar{v}*^{(5,3)}$	**0.563**	0.524	0.561	0.557	0.555	0.481
$\bar{v}*^{(6,2)}$	0.606	**0.663**	0.581	0.564	0.514	0.661
$\bar{v}*^{(6,3)}$	0.604	0.681	0.564	0.554	0.486	**0.697**

tral similarity measure misclassifies 15 out of 36, or 42% of all runs. Since these results are not competitive with either the Bayes classifier or the HMM-based similarity measure, we do not consider this method further in this chapter.

7.1.5 Task-based classification

Here we present results for task-based classification. We first divide data sets $X^{(i,j)}$, $j \in \{1, 2, 3\}$, into the set of left-hand maneuvers $\alpha^{(i)}$ and the set of right-hand maneuvers $\beta^{(i)}$, $i \in \{1, 2, \ldots, 6\}$ contained in runs $X^{(i,j)}$. We then split each set $\alpha^{(i)}$, $\beta^{(i)}$ into two sets,

$$\alpha_k^{(i)}, \beta_k^{(i)}, k \in \{1, 2\}, \tag{7-21}$$

so that half the maneuvers are in sets $\alpha_1^{(i)}$, $\beta_1^{(i)}$, while the remaining maneuvers are in sets $\alpha_2^{(i)}$, $\beta_2^{(i)}$, respectively.

Table 7-10: Left-turn similarity classification

σ	$\alpha_1^{(1)}$	$\alpha_2^{(1)}$	$\alpha_3^{(1)}$	$\alpha_4^{(1)}$	$\alpha_5^{(1)}$	$\alpha_6^{(1)}$
$\alpha_1^{(2)}$	**0.751**	0.338	0.365	0.364	0.063	0.014
$\alpha_2^{(2)}$	0.265	**0.462**	0.095	0.064	0.020	0.329
$\alpha_3^{(2)}$	0.358	0.155	**0.722**	0.209	0.138	0.008
$\alpha_4^{(2)}$	0.103	0.098	0.157	**0.611**	0.175	0.013
$\alpha_5^{(2)}$	0.016	0.030	0.110	0.293	**0.573**	0.006
$\alpha_6^{(2)}$	0.013	0.124	0.009	0.016	0.008	**0.694**

Table 7-11: Right-turn similarity classification

σ	$\beta_1^{(1)}$	$\beta_2^{(1)}$	$\beta_3^{(1)}$	$\beta_4^{(1)}$	$\beta_5^{(1)}$	$\beta_6^{(1)}$
$\beta_1^{(2)}$	**0.625**	0.200	0.353	0.375	0.071	0.025
$\beta_2^{(2)}$	0.277	**0.451**	0.085	0.069	0.010	0.387
$\beta_3^{(2)}$	0.344	0.141	**0.710**	0.267	0.152	0.012
$\beta_4^{(2)}$	0.201	0.114	0.269	**0.612**	0.275	0.015
$\beta_5^{(2)}$	0.031	0.035	0.120	0.253	**0.638**	0.003
$\beta_6^{(2)}$	0.019	0.087	0.007	0.012	0.003	**0.737**

Using the parameters given in (7-6), Tables 7-10 and 7-11 report the similarity results for $\sigma[\alpha_1^{(i)}, \alpha_2^{(j)}]$ and $\sigma[\beta_1^{(i)}, \beta_2^{(j)}]$, $\forall i, j$. Similarly, Tables 7-12 and 7-13 report analogous results for the Bayes classifier. We see once again that the similarity measure classifies all 12 sets $\alpha_2^{(i)}$, $\beta_2^{(i)}$ correctly, while the Bayes classifier misclassifies 4 out of 12, or 33%.

7.1.6 Classification with performance drift

Finally, we examine classification performance for control data where a single individual slowly improves over time. For this experiment, we ask Groucho to drive over the *same* road (road #1 in Figure 2-2) on two different days, twice each day. This generates a total of four runs

Table 7-12: Left-turn Bayes classification

ζ	$\alpha_1^{(1)}$	$\alpha_2^{(1)}$	$\alpha_3^{(1)}$	$\alpha_4^{(1)}$	$\alpha_5^{(1)}$	$\alpha_6^{(1)}$
$\alpha_1^{(2)}$	**0.443**	0.127	0.145	0.134	0.105	0.047
$\alpha_2^{(2)}$	**0.287**	0.196	0.074	0.038	0.053	**0.353**
$\alpha_3^{(2)}$	**0.309**	0.135	0.253	0.133	0.138	0.031
$\alpha_4^{(2)}$	0.090	0.073	0.125	**0.517**	0.182	0.013
$\alpha_5^{(2)}$	0.113	0.108	0.126	**0.320**	0.311	0.022
$\alpha_6^{(2)}$	0.049	0.187	0.026	0.041	0.051	**0.647**

Table 7-13: Right-turn Bayes classification

ζ	$\beta_1^{(1)}$	$\beta_2^{(1)}$	$\beta_3^{(1)}$	$\beta_4^{(1)}$	$\beta_5^{(1)}$	$\beta_6^{(1)}$
$\beta_1^{(2)}$	**0.387**	0.107	0.173	0.158	0.123	0.051
$\beta_2^{(2)}$	0.251	0.178	0.097	0.075	0.079	**0.319**
$\beta_3^{(2)}$	0.252	0.118	**0.271**	0.145	0.170	0.043
$\beta_4^{(2)}$	0.128	0.069	0.144	**0.412**	0.230	0.017
$\beta_5^{(2)}$	0.101	0.115	0.149	0.230	**0.391**	0.013
$\beta_6^{(2)}$	0.056	0.202	0.029	0.045	0.075	**0.594**

(#1, #2, #3 and #4). Because the runs are recorded on the same road, Groucho improves his control strategy relatively quickly over the four runs, raising his average speed per run from 65.9 mi/h to 71.9mi/h from run #1 to run #4.

Next, we take two additional data sets over the same road from Curly and Moe, respectively. Figure 7-4 plots the mean and standard deviation for the velocity v and acceleration control ϕ for all six runs. Note that Curly's and Moe's runs share some similar aggregate statistics with at least some of Groucho's runs.

Table 7-14: (a) Similarity measure classification results

σ	*Curly*	*Groucho #4*	σ	*Moe*	*Groucho #4*
Groucho #1	**0.572**	0.528	*Groucho #1*	0.315	**0.616**
Groucho #2	0.435	**0.540**	*Groucho #2*	0.495	**0.603**
Groucho #3	0.258	**0.728**	*Groucho #3*	0.550	**0.760**

Table 7-14: (b) Bayes classification results

ζ	*Curly*	*Groucho #4*	ζ	*Moe*	*Groucho #4*
Groucho #1	**0.609**	0.391	***Groucho #1***	**0.569**	0.431
Groucho #2	**0.589**	0.411	***Groucho #2***	**0.663**	0.337
Groucho #3	0.416	**0.583**	***Groucho #3***	**0.567**	0.433

Now we run six classification experiments. First, we classify Groucho's first three runs as more similar to either Groucho's fourth run or Curly's run. Second, we classify Groucho's first three runs as more similar to either Groucho's fourth run or Moe's run. Using the parameters given in (7-6)[4], Table 7-14(a) reports the similarity measure results for these comparisons. Similarly, Table 7-14(b) reports analogous results for the Bayes classifier. We observe that the similarity measure misclassifies one out of six (17%), while the Bayes classifier misclassifies five out of six (83%) experiments, some quite badly.

7.2 Comparing Navlab driving data

7.2.1 Experimental data

Finally, we present classification results for real road-driving data, collected as part of an on-going research effort geared towards the development of autonomous vehicles at Carnegie Mellon University. Data was collected from seven drivers of both genders, ranging in age from 21 to 50 in Navlab 8, a minivan equipped with RALPH, a vision-based lateral-position estimation system [15]. Each driver was asked to drive from Carnegie Mellon University, Pittsburgh, PA,

4. Here we only use $L = 64$ VQ codes, since we are comparing fewer data sets.

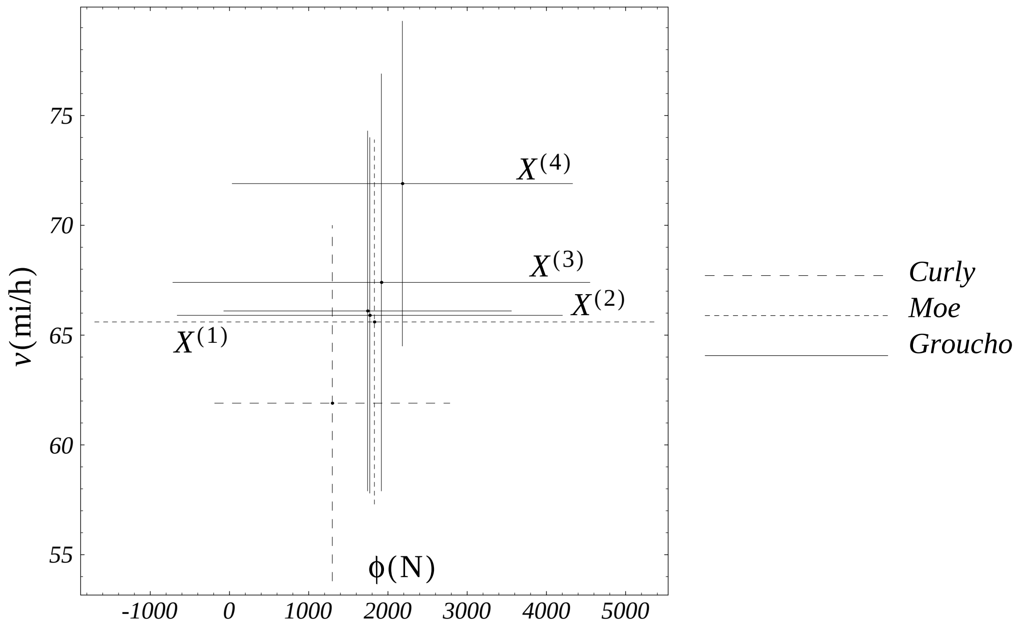


Figure 7-4: Groucho's average speed improves over time; Curly's and Moe's runs are statistically close.

to Grove City, PA, 50 miles north of Pittsburgh and back, while data was being recorded at 16Hz; his/her data was then split into two runs — $X^{(i,1)}$ (going north) and $X^{(i,2)}$ (going south), $i \in \{1, 2, \ldots, 7\}$. Since the route consists primarily of two and three lane traffic, each one-way trip took approximately 40 to 45 minutes. Throughout, drivers received no coaching or instructions other than to drive safely.

7.2.2 Classification experiments

The following state dimensions are available in each data set $X^{(i,j)}$: (1) d_ξ, the lateral position of the vehicle, (2) v, the velocity of the vehicle, and (3) ω the angular velocity of the vehicle. Using these variables as input,

$$X^{(i,j)} = \begin{bmatrix} \bar{d}_\xi & \bar{v} & \bar{\omega} \end{bmatrix}^{(i,j)} \tag{7-22}$$

we select the same similarity parameters as in equation (7-6), except that we let,

$$\bar{\varphi} = \begin{bmatrix} G & G & G \end{bmatrix}^T, \tag{7-23}$$

while, for the Bayes classification, we use (7-23) and $\kappa_1 = \kappa_2 = 1$. Now we perform two sets of experiments. In the first set, we classify each of the south-bound runs $X^{(i,2)}$ as similar to one of the north-bound runs $X^{(j,1)}$, $i, j \in \{1, 2, \ldots, 7\}$. Table 7-15 reports these classification results for the similarity measure σ, while Table 7-16 reports analogous results for the Bayes classifier. We observe that the similarity measure classifies all seven $X^{(i,2)}$ runs correctly, while the Bayes classifier misclassifies 2 out of 7, or 29%.

In the second set of experiments, we first select two runs, $X^{(i,j)}$ and $X^{(p,q)}$, $i \neq p$. We then try to classify the other runs from driver *i* and driver *p* as belonging to either of the classes represented by $X^{(i,j)}$ and $X^{(p,q)}$. This allows us to conduct $7 \times 6 \times 2 = 84$ separate comparisons. The similarity measure correctly classifies all 84 of these comparisons, while the Bayes classifier misclassifies 9 out of 84, or 11%. Thus, for the real driving data in this section, as well as the simulated driving data in the previous sections, the HMM-based similarity measure again outperforms the Bayes classifier.

7.3 Analysis

In the discussion below, we first justify the similarity measure definition in greater detail. We then specifically address why the Bayes classifier fails in some cases, where the similarity measure succeeds. Finally, we consider the similarity measure's performance as a function of the signal-to-symbol preprocessing and the number of HMM states.

7.3.1 One-sided similarity measure

The definition of the similarity measure in equation (6-15) requires that one HMM λ is trained for each control trajectory. In practice, this means that whenever we wish to compare an unknown control trajectory to a bank of known models, we first have to train an HMM on the unknown control trajectory, thus hurting potential on-line (i.e. real-time) performance. Why don't we avoid this problem by defining the following one-sided similarity measure $\tilde{\sigma}$,

$$\tilde{\sigma}(\lambda_1, \bar{O}_2) = P_{21}/P_{11} \text{ or } \tilde{\sigma}(\lambda_2, \bar{O}_1) = P_{12}/P_{22}? \tag{7-24}$$

Table 7-15: Similarity results for real driving data

σ	*X(1,1)*	*X(2,1)*	*X(3,1)*	*X(4,1)*	*X(5,1)*	*X(6,1)*	*X(7,1)*
X(1,2)	**0.606**	0.522	0.112	0.190	0.068	0.062	0.128
X(2,2)	0.356	**0.747**	0.405	0.637	0.325	0.299	0.536
X(3,2)	0.167	0.487	**0.743**	0.385	0.619	0.381	0.281
X(4,2)	0.372	0.689	0.359	**0.758**	0.342	0.432	0.563
X(5,2)	0.275	0.343	0.364	0.186	**0.431**	0.279	0.109
X(6,2)	0.155	0.374	0.255	0.612	0.309	**0.683**	0.553
X(7,2)	0.112	0.409	0.153	0.668	0.158	0.442	**0.799**

Table 7-16: Bayes classification results for real driving data

ζ	*X(1,1)*	*X(2,1)*	*X(3,1)*	*X(4,1)*	*X(5,1)*	*X(6,1)*	*X(7,1)*
X(1,2)	**0.334**	0.218	0.093	0.102	0.083	0.033	0.079
X(2,2)	0.107	**0.156**	0.139	0.137	0.140	0.087	0.129
X(3,2)	0.097	0.151	**0.220**	0.098	0.178	0.090	0.084
X(4,2)	0.106	**0.146**	0.138	0.135	0.133	0.106	0.121
X(5,2)	**0.216**	0.199	0.118	0.089	0.139	0.093	0.068
X(6,2)	0.043	0.094	0.096	0.152	0.123	**0.169**	0.162
X(7,2)	0.043	0.106	0.083	0.166	0.110	0.146	**0.182**

The short answer is that is does not work as one might expect. Specifically, $\tilde{\sigma}$ no longer obeys properties #1, #2 and #3 in equations (6-20) through (6-22). Consider the following simple example. Let,

$$O_1 = \{0, 0, 0, 1, 1, 1, 1, 1, 1, 1\}, \text{ and} \tag{7-25}$$

$$O_2 = \{0, 1, 1, 1, 1, 1, 1, 1, 1, 1\}. \tag{7-26}$$

The corresponding single-state HMMs λ_1 and λ_2 are given by,

$$\lambda_1 = \left\{ \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}, \lambda_2 = \left\{ \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \tag{7-27}$$

Recall that $P_{ij} = 10^{\log P(O_i|\lambda_j)/T_i}$. Thus,

$$P_{21} = 10^{\log[(0.7^9)(0.3)]/10} = 0.643 \tag{7-28}$$

$$P_{11} = 10^{\log[(0.3^3)(0.7^7)]/10} = 0.543 \tag{7-29}$$

$$P_{12} = 10^{\log[(0.9^7)(0.1^3)]/10} = 0.466 \tag{7-30}$$

$$P_{22} = 10^{\log[(0.1)(0.9^9)]/10} = 0.722. \tag{7-31}$$

Consequently,

$$\tilde{\sigma}(\lambda_1, \bar{O}_2) = 0.643/0.543 = 1.185, \text{ and} \tag{7-32}$$

$$\tilde{\sigma}(\lambda_2, \bar{O}_1) = 0.466/0.722 = 0.644. \tag{7-33}$$

Not only do we lose the nice properties of the original similarity measure, we also get mixed classification results for the human driving data. Tables 7-17, 7-18 and 7-19, for example, report one-sided classification results analogous to the similarity measure results in Tables 7-1, 7-2 and 7-3, respectively. Note that 4 out of 36 comparisons (11%) are misclassified by the one-sided similarity measure.

7.3.2 Bayes classifier limitations

Here, we examine why the Bayes classifier performs more poorly than the HMM-based similarity measure. Figure 7-5(a) plots the distribution (over v and ϕ) for Moe's data, and the corresponding Gaussian approximation of that distribution. Likewise, Figure 7-5(b) and (c) plot

Table 7-17: One-sided similarity results for road #1 data

$\tilde{\sigma}$	λ_{11}	λ_{21}	λ_{31}	λ_{41}	λ_{51}	λ_{61}
O(1, 2, 1)	**0.784**	0.343	0.278	0.141	0.057	0.022
O(1, 3, 1)	**0.579**	0.518	0.265	0.057	0.051	0.160
O(2, 2, 1)	0.345	**0.964**	0.185	0.125	0.098	0.797
O(2, 3, 1)	0.290	**1.100**	0.152	0.076	0.061	0.896
O(3, 2, 1)	0.590	0.252	**0.909**	0.325	0.183	0.029
O(3, 3, 1)	0.569	0.251	**0.883**	0.120	0.137	0.081
O(4, 2, 1)	0.547	0.304	0.523	**0.824**	0.373	0.030
O(4, 3, 1)	0.588	0.311	0.590	**0.752**	0.445	0.123
O(5, 2, 1)	0.166	0.191	0.351	0.347	**0.742**	0.057
O(5, 3, 1)	0.195	0.150	0.329	0.325	**0.700**	0.140
O(6, 2, 1)	0.012	0.114	0.004	0.003	0.003	**1.563**
O(6, 3, 1)	0.006	0.159	0.003	0.002	0.002	**0.966**

Table 7-18: One-sided similarity results for road #2 data

$\tilde{\sigma}$	λ_{12}	λ_{22}	λ_{32}	λ_{42}	λ_{52}	λ_{62}
O(1, 1, 2)	**0.912**	0.264	0.268	0.174	0.081	0.068
O(1, 3, 2)	0.704	**0.705**	0.311	0.066	0.046	0.216
O(2, 1, 2)	0.310	0.551	0.078	0.044	0.044	**0.806**
O(2, 3, 2)	0.148	**1.133**	0.062	0.020	0.018	**0.988**
O(3, 1, 2)	0.674	0.226	**0.767**	0.207	0.178	0.034
O(3, 3, 2)	0.523	0.327	**0.815**	0.076	0.082	0.059
O(4, 1, 2)	0.578	0.437	0.542	**0.847**	0.349	0.088
O(4, 3, 2)	0.569	0.467	0.540	**0.657**	0.310	0.129
O(5, 1, 2)	0.049	0.064	0.195	0.251	**0.603**	0.013
O(5, 3, 2)	0.098	0.013	0.180	0.246	**0.636**	0.008
O(6, 1, 2)	0.001	0.002	0.000	0.000	0.000	**0.158**
O(6, 3, 2)	0.000	0.014	0.000	0.000	0.000	**0.320**

Table 7-19: One-sided similarity results for road #3 data

$\tilde{\sigma}$	λ_{13}	λ_{23}	λ_{33}	λ_{43}	λ_{53}	λ_{63}
O(1, 1, 3)	**0.877**	0.246	0.328	0.208	0.107	0.028
O(1, 2, 3)	**0.927**	0.266	0.316	0.202	0.075	0.005
O(2, 1, 3)	0.178	0.277	0.031	0.017	0.011	**0.582**
O(2, 2, 3)	0.125	0.498	0.030	0.018	0.007	**0.792**
O(3, 1, 3)	0.640	0.250	**0.699**	0.237	0.231	0.030
O(3, 2, 3)	0.620	0.203	**0.785**	0.225	0.154	0.005
O(4, 1, 3)	0.586	0.447	0.603	**0.839**	0.476	0.155
O(4, 2, 3)	0.592	0.373	0.530	**0.813**	0.419	0.169
O(5, 1, 3)	0.062	0.115	0.260	0.337	**0.721**	0.057
O(5, 2, 3)	0.058	0.052	0.272	0.330	**0.843**	0.039
O(6, 1, 3)	0.006	0.011	0.001	0.001	0.001	**0.475**
O(6, 2, 3)	0.004	0.034	0.001	0.001	0.000	**1.454**

similar comparisons for Groucho's second run and Groucho's fourth run, respectively. From these plots, it is clear that the Bayes classifier is doomed to fail, since the human data is distributed in a decidedly non-Gaussian manner. The similarity measure, on the other hand, succeeds because the HMMs are trained on the underlying distributions of the data sets, and make no *a priori* assumptions about each individual's distribution.

We also note that despite repeated attempts at improving the Bayes classifier's performance — by only classifying on a subset of the vector $\begin{bmatrix} v_\xi & v_\eta & \omega & \delta & \phi \end{bmatrix}^T$ — we have yet to identify an example where the Bayes classifier succeeds and the similarity measure fails.

7.3.3 Similarity measure variations

Finally, we consider the similarity measure's performance as a function of (1) the signal-to-symbol preprocessing and (2) the number of HMM states. Let us first define a discrimination measure, with which we can evaluate the similarity mesure's performance. Let,

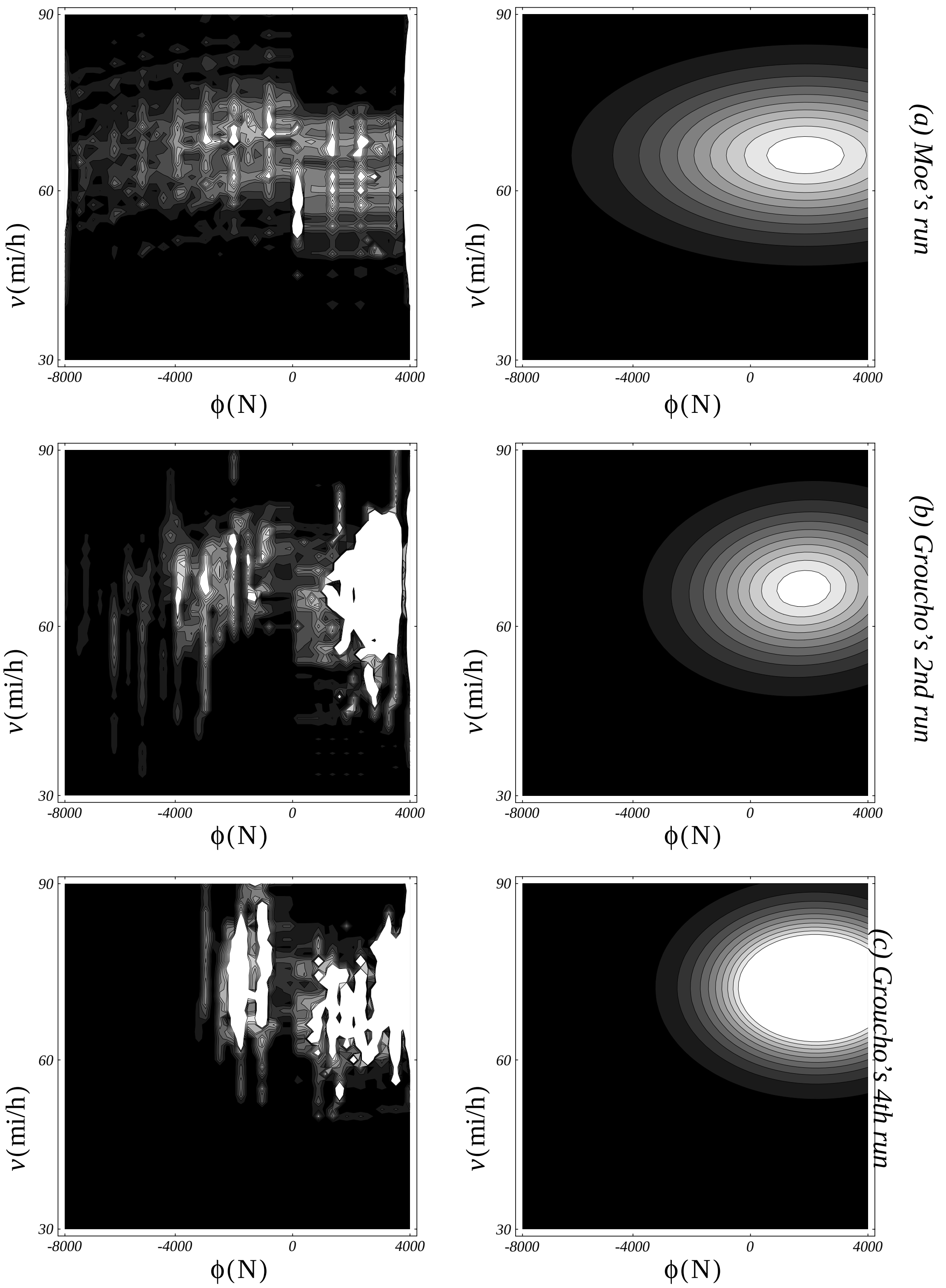


Figure 7-5: Groucho's second run (b) is badly misclassified by the Bayes classifier as Moe's run (a) and not as Groucho's fourth run (c).

$$\delta_\sigma^{(i,j,k)} = \frac{\max_{l, l \neq i}(\sigma[O^k_{(i,j)}, O^k_{(l,k)}])}{\sigma[O^k_{(i,j)}, O^k_{(i,k)}]}, \ j \neq k, \tag{7-34}$$

$$\delta_{\sigma, \log}^{(i,j,k)} = -\log[\delta_\sigma^{(i,j,k)}], \tag{7-35}$$

define a discrimination measure corresponding to row $O(i, j, k)$ of Tables 7-1, 7-2 and 7-3, where $i \in \{1, 2, \ldots, 6\}$, $j, k \in \{1, 2, 3\}$, $j \neq k$. This measure takes the maximum off-diagonal element over the self-similar element in each row. Thus, $\delta_{\sigma, \log} > 0$ indicates that sequence $O^k_{(i,j)}$ is classified correctly. Next define,

$$\Delta_\sigma = -\log\left[\frac{1}{N}\sum_k \sum_{\substack{i,j \\ j \neq k}} \delta_\sigma^{(i,j,k)}\right], \ i, l \in \{1, 2, \ldots, 6\}, \ j, k \in \{1, 2, 3\}, \ N = 36, \tag{7-36}$$

where Δ_σ simply averages the discrimination measure in equation (7-34) over all rows in Tables 7-1, 7-2 and 7-3 and takes the logarithm, so that $\Delta_\sigma > 0$ indicates better classification, while $\Delta_\sigma < 0$ indicates worse classification. We are now in a position to evaluate the similarity measure's performance as we vary different design parameters in the similarity analysis.

First, we show how the similarity measure changes when *no* spectral preprocessing is performed, such that,

$$O^k_{(i,j)} = O(i, j, k) = T_{all}(X^{(i,j)}, \bar{s}^k, \bar{\varphi}, [\kappa_1, \kappa_2], Q^k_L), \ i \in \{1, 2, \ldots, 6\}, \\ j, k \in \{1, 2, 3\}, \text{ where,} \tag{7-37}$$

$$X^{(i,j)} = \left[\bar{v}_\xi \ \bar{v}_\eta \ \bar{\omega} \ \bar{\delta} \ \bar{\phi}\right]^{(i,j)}, \ i \in \{1, 2, \ldots, 6\}, \ j \in \{1, 2, 3\}, \text{ and,} \tag{7-38}$$

$$\bar{\varphi} = \left[G \ G \ G \ G \ G\right]^T, \ L = 128, \ n_s = 8. \tag{7-39}$$

We consider two cases, $\kappa_1 = \kappa_2 = 1$, and $\kappa_1 = 2\kappa_2 = 16$.

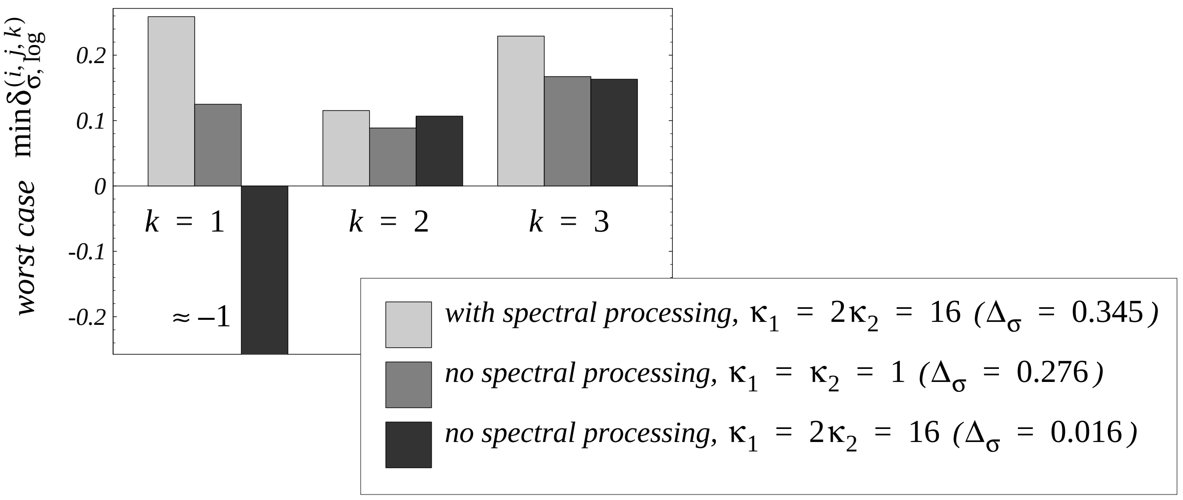


Figure 7-6: The discrimination performance worsens with no preprocessing.

With eight-state HMMs, and no spectral preprocessing, $\Delta_\sigma = 0.276$ for $\kappa_1 = \kappa_2 = 1$, $\Delta_\sigma = 0.016$ for $\kappa_1 = 2\kappa_2 = 16$, while for Tables 7-1, 7-2 and 7-3, $\Delta_\sigma = 0.345$. Figure 7-6 plots the worst discrimination example $\delta_{\sigma,\log}^{(i,j,k)}$, for $k \in \{1, 2, 3\}$, corresponding to each table. Note that for $k = 1$ and $\kappa_1 = 2\kappa_2 = 16$, we actually get two misclassifications. Thus, signal preprocessing prior to vector quantization contributes positively to the discrimination capacity of the similarity measure.

Second, we examine how the similarity measure changes as a function of the number of HMM states in our HMM models λ. Let the signal-preprocessing parameters be given by (7-6), and repeat the similarity analysis for Tables 7-1, 7-2 and 7-3 as the number of HMM states n_s is varied from 1 to 10. Figure 7-7 plots the discrimination measure Δ_σ as a function of n_s. We see that as n_s is increased initially, Δ_σ improves sharply. As n_s becomes larger, however, discrimination begins to level off and eventually declines from $n_s = 9$ to $n_s = 10$. At that point, for $\kappa_1 = 2\kappa_2 = 16$, and sampling period $\tau = 0.02\text{sec}$, common aspects of control strategies from different individuals begin to dominate the unique features of each strategy.

From these results, we suggest the following three reasons — in order of importance — for the success of our similarity measure: (1) no *a priori* distribution of the data is assumed, as HMMs

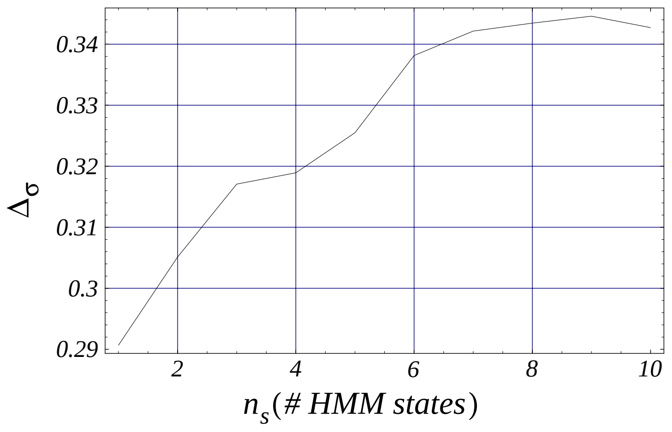


Figure 7-7: Sequential structure adds to the discrimination capacity of the similarity measure.

are capable of encoding arbitrary statistical distributions; (2) spectral preprocessing extracts useful features from the control strategy data; and (3) HMMs in part capture the sequential structure of the control strategy data.

In summary, this chapter has shown that the proposed similarity measure correctly classifies control strategy data from the same individual, while discriminating driving control data from different individuals in human-to-human comparisons, both for simulated as well as real driving data. It is these results that now justify applying the similarity measure as a *validation measure* in human-to-model comparisons. Chapter 8 will do just that, thereby quantifying the qualitative observations of HCS model fidelity in Chapters 4 and 5.

Chapter 8

Human-to-model validation

In Chapter 7, we verified the performance of the similarity measure σ in human-to-human comparisons. Now, we will apply the proposed similarity measure as a means of validating the control trajectories of the different HCS models presented in Chapters 4 and 5. In other words we will quantify our previous qualitative observations about the similarity between the learned models and their respective human control training data.

8.1 Human-to-model comparisons

We once again select the following parameters for our similarity analysis:

$$\bar{\varphi} = \left[HF\ HF\ HF\ W\ W\right]^T,\ \kappa_1 = 16,\ \kappa_2 = \kappa_1/2,\ L = 128,\ n_s = 8, \tag{8-1}$$

so that for a control trajectory $X^* = \left[\bar{v}_\xi\ \bar{v}_\eta\ \bar{\omega}\ \bar{\delta}\ \bar{\phi}\right]$, the corresponding observation sequence O^* will be given by,

$$O^* = T_{all}(X^*, \bar{s}, \bar{\varphi}, [\kappa_1, \kappa_2], Q_L), \tag{8-2}$$

where the scale vector $\bar{s}$ is chosen and the VQ codebook Q_L is trained over all control trajectories in the similarity analysis. Note that the parameters in equation (8-1) are the same as those that we used through most of Chapter 7.

Table 8-1 and Figure 8-1 report the resulting human-to-model similarity measures σ, for Larry, Moe, Groucho and Harpo, and their respective *Cq*, *Ck* and hybrid HCS models. In addition, we provide similarity results for the hybrid HCS models, where we let the statistical models λ_i be uniform, such that equation (5-19),

$$P(A_i|S_l) = b(l)_i P(A_i) \Big/ \sum_{i=1}^{N} b(l)_i P(A_i), \tag{8-3}$$

reduces to,

$$P(A_i|S_l) = P(A_i). \tag{8-4}$$

In other words, the choice of control action at each time step depends strictly on the priors $P(A_i)$, when the λ_i are uniform.

Table 8-1: Human-to-model similarity[a]

Individual	*Dimensions*	*Cq*	*Ck*	*Just* $P(A_i)$ [b]	*Hybrid*
Larry	$\{\bar{x}, \bar{u}\}$ [c]	0.100	0.161	0[d]	0.450
	$\{\bar{u}\}$ [e]	0.128	0.432	0[d]	0.657
Moe	$\{\bar{x}, \bar{u}\}$	0.087	0.088	0.046	0.555
	$\{\bar{u}\}$	0.117	0.146	0.187	0.594
Groucho	$\{\bar{x}, \bar{u}\}$	0.101	0.096	0.014	0.457
	$\{\bar{u}\}$	0.319	0.172	0.132	0.561
Harpo	$\{\bar{x}, \bar{u}\}$	0[d]	0.003	0.012	0.578
	$\{\bar{u}\}$	0[d]	0.003	0.024	0.609

a. Each individual's run is compared to the model trajectory over road #5.
b. Models λ_i are uniform over the entire input space. Condition (5-27) is enforced.
c. All state and control variables are included for the similarity analysis.
d. Model is unstable over road #5.
e. Only the control variables $\{\delta, \phi\}$ are included in the similarity analysis.

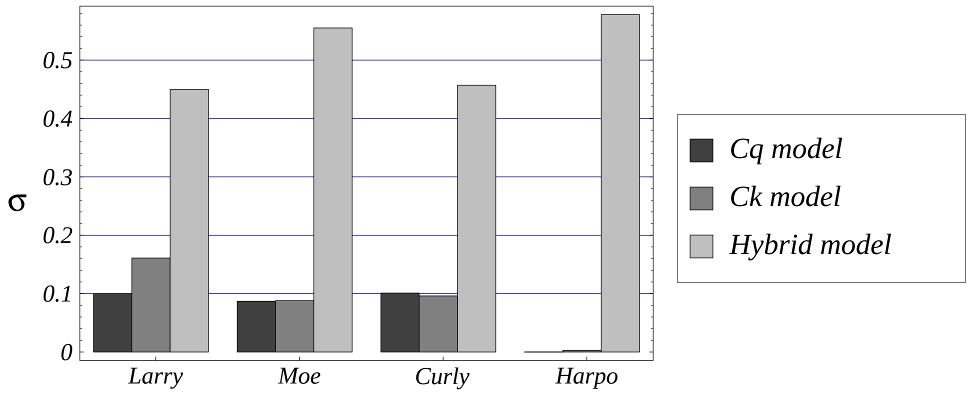


Figure 8-1: Human-to-model similarity for different modeling approaches and $\{\bar{x}, \bar{u}\}$.

From Table 8-1, we make several observations. First, the *Cq* and *Ck* models exhibit roughly the same similarity to each model's corresponding human control data. These similarity values are, however, rather low, especially when compared to the human-to-human similarity results in Chapter 7. Therefore, neither the *Cq* nor the *Ck* learning algorithm is able to model the driving control strategies with a high degree of fidelity to the source training data.

In comparison, we note that the hybrid controllers have much more in common with the source training data than do either the *Cq* and *Ck* models. In fact, the similarity values for the human-to-model comparisons in Table 8-1 approach those of the self-similar human-to-human comparisons in Chapter 7. Finally, we observe that without models λ_i, the hybrid control strategies degenerate, and are no longer representative of the human's control strategy. This confirms that the statistical models λ_i impart useful information to the hybrid control strategies, and that the improved fidelity of the hybrid controllers in not simply due to random thrashing about of the acceleration command ϕ.

8.2 Classification experiment

As an additional validation check, we now show that a particular individual's hybrid model not only closely matches the control data for that individual, but also exhibits a lesser degree of similarity with other individual's control data. Table 8-2 reports the similarity of each individ-

Table 8-2: Hybrid model-to-human matching

σ	*Larry*	*Moe*	*Groucho*	*Harpo*
Larry's model	**0.450**	0.329	0.315	0.069
Moe's model	0.126	**0.555**	0.338	0.217
Groucho's model	0.152	0.377	**0.457**	0.206
Harpo's model	0.013	0.134	0.127	**0.578**

Table 8-3: *Ck* model-to-human matching

σ	*Larry*	*Moe*	*Groucho*	*Harpo*
Larry's model	0.161	**0.166**	0.118	0.157
Moe's model	0.056	**0.088**	0.063	0.041
Groucho's model	0.056	0.066	**0.096**	0.040
Harpo's model	0.006	0.008	**0.012**	0.003

ual's model with each individual's control strategy for the hybrid HCS models. We observe that the highest degree of similarity occurs between a specific individual and his model. In contrast, we observe from Table 8-3, that the *Ck* models do not necessarily exhibit the highest degree of similarity (however low) with their respective training data.

8.3 Model inputs revisited

In Section 4.2.1, we described a method for selecting a given model's input representation (n_x, n_u) based on the performance of the *Ck* models on the cross validation road #4 (Figure 2-3(b)). Here we revisit the problem of model input selection, within the context of the similarity measure σ.

From Figure 4-5, we made the observation that *Ck* model performance does not appear to vary significantly for $n_x, n_u \geq 3$, as judged by the maximum deviation from the road median. Alternatively, we can look at,

$$d(\Gamma_k^{(i,j)}, \Gamma_{k+1}^{(i,j)}) \equiv d(O_k^{(i,j)}, O_{k+1}^{(i,j)}),\ k \in \{1, 2, \ldots, 19\}, \tag{8-5}$$

where $d(\cdot)$ indicates the distance measure defined in equation (6-28), $O_k^{(i,j)}$ denotes an observation sequence converted from the control trajectory of model $\Gamma_k^{(i,j)}$ over validation road #4, and model $\Gamma_k^{(i,j)}$, as before, corresponds to a *Ck* model trained from run $X^{(i,j)}$ (i.e. person (*i*) on road #*j*), with input space,

$$\{\bar{x}^k, \bar{u}^k, \bar{z}^{10}\},\ k \in \{1, 2, \ldots, 20\}. \tag{8-6}$$

Figure 8-2, for example, plots $d(\Gamma_k, \Gamma_{k+1})$ as k varies from 1 to 19 for Groucho's *Ck* model. Once again, we observe that model performance does not vary significantly for $k \geq 3$.

For the *Ck* models, we could not look at the similarity between the models and their respective training data as a robust indicator, since the similarity measure σ evaluates to very low values. For the hybrid models, however, the similarity measures appear to be more robust. Hence for these models, we can look directly at the human-to-model similarity,

$$d(X^{(i,j)}, \Gamma_k^{(i,j)}) = d(O^{(i,j)}, O_k^{(i,j)}). \tag{8-7}$$

Figure 8-3, plots $d(X, \Gamma_k)$ as k varies from 1 to 6.[1] We observe that for each person the minimum is located at $k = 3$. It appears that no matter what modeling approach we take, when the

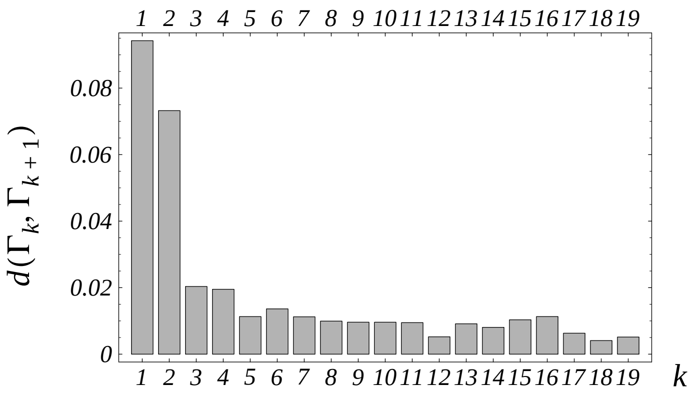


Figure 8-2: Model distance for Groucho's run as the size of the input space is varied.

1. We stop at $k = 6$ due to the increased distortion in the VQ codebook for the hybrid models as the dimensionality of the input space increases.

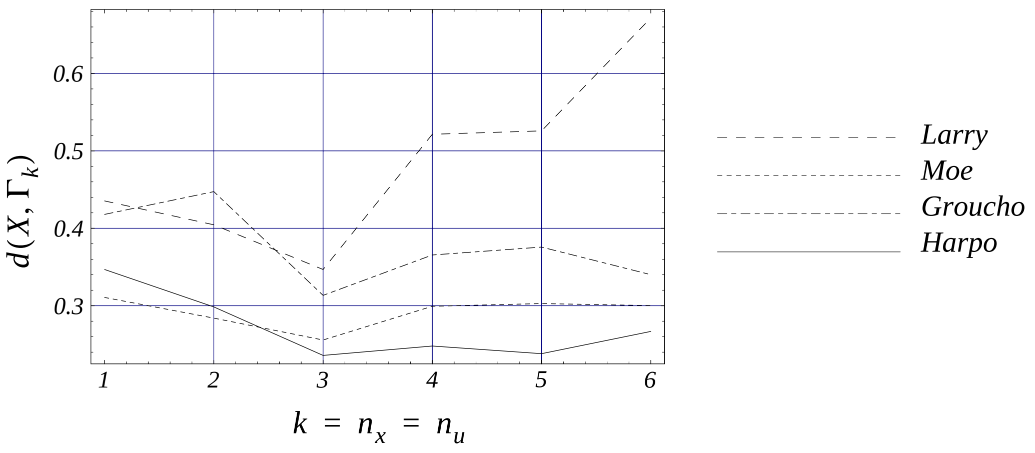


Figure 8-3: Human to model similarity over test road #5 as we vary the size of the input space.

model is presented with enough time-delayed values of each state and control variable, the model is able to build what appear to be necessary derivative dependencies between model inputs and outputs $\{\delta(k+1), \phi(k+1)\}$.

In summary, we have demonstrated that the hybrid models exhibit greater fidelity to the human training data than the either of the cascade network-based modeling approaches. Which learning approach — continuous or hybrid — is preferred may ultimately depend on the specific application for the HCS model. If the model is being developed towards the eventual control of a real robot or vehicle, then the continuous modeling approach might be preferred as a good starting point. Continuous models extrapolate control strategies to a greater range of inputs, show greater inherent stability, and lend themselves more readily to theoretical performance analysis. If, on the other hand, the model is being developed in order to simulate different human behaviors in a virtual reality simulation or game, then the discontinuous control approach might be preferred, since fidelity to the human training data and random variations in behavior would be the desired qualities of the HCS model. Thus, depending on the application, we believe a need exists for both types of modeling approaches.

Chapter 9

Conclusion

9.1 Contributions

In this dissertation, we present a coherent framework for learning and validating discrete-time models of human control strategy. We summarize the original contributions of this work below.

- We developed a neural-network-based algorithm that combines flexible cascade neural networks with extended Kalman filtering. We show that the resulting learning architecture achieves better convergence in faster time than alternative neural-network paradigms for modeling both known continuous functions and dynamic systems, as well as for modeling human control strategies from real-time human data. We also demonstrate the fundamental problem of modeling discontinuous control strategies with a continuous function approximator.

- We developed a statistical, discontinuous framework for modeling discontinuous human control strategies. The approach models control actions as probabilistic events and chooses a specific control action based on a stochastic selection criterion. We demonstrate that the resulting learning architecture is much better able to approximate discontinuous control strategies than continuous function approximators.

- As a model validation tool, we developed a stochastic similarity measured — based on Hidden Markov Models — that measures the level of similarity (or dissimilarity) between multi-dimensional, stochastic trajectories. We demonstrate and derive important properties of the similarity measure.

- We applied the similarity measure towards classifying human control data across different individuals. We demonstrate that the similarity measure does a better job of grouping human control data from the same individual than the more traditional Bayes classifier. We also analyze classification performance as a function of similarity measure parameters.

- We applied the similarity measure for comparisons between human control strategy models and their respective human training data. Thus we quantify qualitative results about model fidelity in different modeling approaches. We also apply the similarity measure for model-to-model comparisons to show how representational choices of the input space affect model performance.

- We developed a real-time graphic driving simulator, with dynamic interactions of the simulated car an the environment. This has proven to be a valuable testing tool for the learning algorithms and statistical methods developed herein. Some other researchers have also used this simulator in their work [29, 110].

9.2 Future work

While this thesis provides the foundations for modeling and analyzing human control strategies, it is certainly not the first and last word on this topic — it is only an important first step. There are a number of different directions in which the work in this thesis can be extended and applied.

Once we have abstracted a HCS model, it is important to assess the skill exhibited by the model and its corresponding human controller. In this thesis, we evaluate models based on stability and model fidelity to the human training data. There are, however, other criteria — many of them task-dependent — by which we can evaluate performance. Models or control strategies with different skill qualities may be more or less appropriate for a given situation, depending on the specific performance requirements of an application. For example, Song, *et. al*. [110] have begun to examine the problem of skill evaluation, by proposing two task-specific performance criteria for the human driving task, including a (1) *tight-turning* and (2) *obstacle avoidance* criterion. These criteria evaluate the performance of a HCS model outside its training range and quantify a particular model's suitability for specific control subtasks.

Given a specific performance requirement, it might be necessary to optimize a particular HCS model with respect to that performance measure. The unoptimized HCS model already gives an initial stable control strategy; optimization would merely refine the parameters in the model to improve performance with respect to a specific criterion J. Since, in general, we do not have an explicit representation for J in terms of the model parameters $\bar{\omega}$, model optimization can be achieved through one of a number of different stochastic algorithms, including simultaneously perturbed stochastic approximation (SPSA) [113], population-based incremental learning (PBIL) [11] and genetic optimization [40]. Initial experiments with SPSA, for example, demonstrate that learned models of human control strategy can be improved with respect to specific performance criteria [111].

Another area of application for HCS models might be as *virtual expert instructors*. A novice, when faced with learning an appropriate control strategy for a new task, is generally faced with two alternatives. Either the novice can attempt to learn the new skill through trial and error, with no on-line feedback to critique performance, or the novice can learn with the help of a human instructor. The first approach can be frustrating and cumbersome, while in the second approach, the feedback advice by the instructor is limited to certain sensor modalities and is sporadic in nature. In addition, individual one-on-one training is expensive and can tax the constrained resources of a single expert.

To alleviate these problems, it may be possible to replace an actual human expert instructor, so that an apprentice gets advice through the expert's HCS model, rather than from the expert directly. The model-generated advice can be presented continuously to the apprentice, while exploiting multiple sensor modalities. This has the potential to improve both learning speed and the quality of learning by the apprentice. In this approach, apprentice training need no longer be one-to-one. A single expert can efficiently train many apprentices through his/her HCS model, without increased demands on the expert's time; conversely, a single apprentice can efficiently benefit from diverse advice of many experts at once (Figure 9-1). Finally, since HCS models are trained on physically plausible human data, feedback advice from the HCS model does not require unreasonably high precision or control fidelity from the apprentice. It has been shown that simulated training (e.g. training on a simulation of the system rather than the real system) still improves performance once the apprentice transitions to control of the real dynamic system [41, 103]. Therefore, we would expect that this approach would prove useful, even if — for safety reasons — we replace the actual system by a simulation of that system during apprentice training.

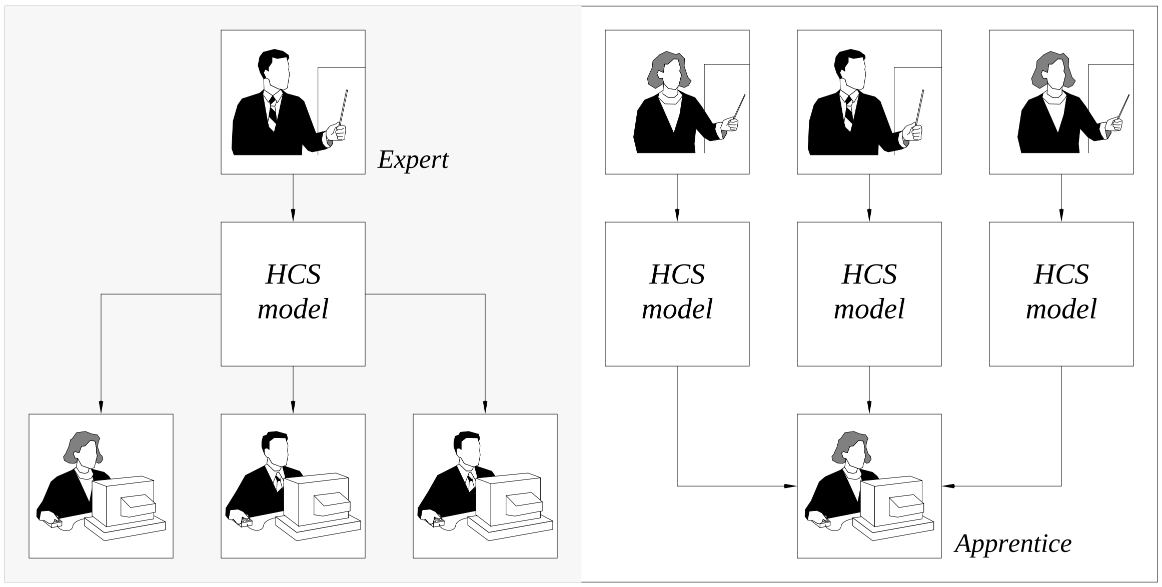


Figure 9-1: One expert can teach many apprentices (left) and many experts can contribute to the learning of a single apprentice (right).

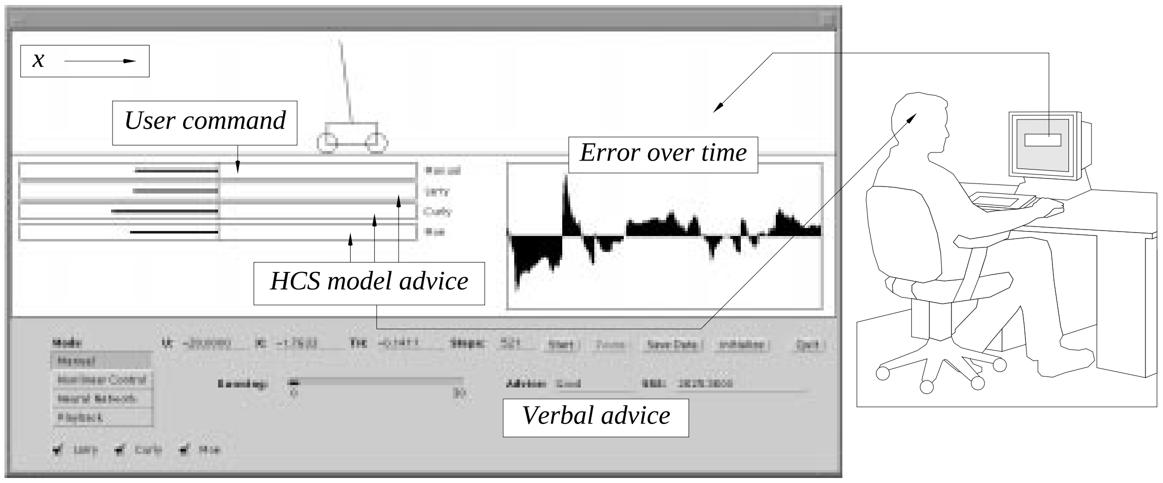


Figure 9-2: An augmented learning interface for human-to-human skill transfer.

In [80], we demonstrate the viability of applying HCS models towards human-to-human skill transfer for a simple inverted-pendulum system (see Figure 9-2). This example does not, however, address a number of important issues in human-to-human skill transfer — namely, (1) model selection for good apprentice learning, (2) multiple model learning, and (3) interface design of the feedback advice. These are all important directions for future research.

Finally, while our primary motivation for developing the similarity measure in Chapter 6 was for validation of HCS models, we believe that it may have other useful applications. Most importantly, it could be used to monitor and detect potentially dangerous control behaviors on the part of the human operator, as is done, for example, in [29] with auto-regressive models. Alternatively, it could be used to monitor an apprentice's control strategy during training to see whether or not his control strategy begins to approach that of the expert.

Appendix A:
Human control data

In this appendix, we describe the human control data sets which we use throughout the thesis for learning and validation experiments. We use the dynamic driving simulator of Section 2.2.1 to collect human control data from six individuals[1] — (1) Larry, (2) Curly, (3) Moe, (4) Groucho, (5) Harpo and (6) Zeppo. In order to become accustomed to the simulator's dynamics, we first allow each individual to practice "driving" in the simulator for up to fifteen minutes prior to recording any actual data. We then ask each person to drive over three different, randomly generated 20km roads — roads #1, #2 and #3 in Figure 2-2 —as fast as possible without veering off the road. Between runs, we allow a short break for each operator.

Sections A.1 through A.6 plot the instantaneous velocity v (mi/h), the lateral offset from the road median d_ξ (m), the steering angle δ (rad) and the acceleration command ϕ (N) for each human control data set. Table A-1 reports corresponding aggregate statistics for each of the 18 runs.

1. All human subjects are males in their mid-20s.

Table A-1: Aggregate statistics for human driving data[a]

	Run	v *(mi/h)*	ω *(rad/s)*[b]	δ *(rad)*[b]	ϕ *(* $\times 10^3$ *N)*	*off road %*
Larry	$X^{(1,1)}$	71.8 (8.1)	(0.183)	(0.064)	1.70 (2.41)	1.31
	$X^{(1,2)}$	71.1 (7.2)	(0.194)	(0.072)	1.81 (2.35)	0.80
	$X^{(1,3)}$	73.7 (8.0)	(0.200)	(0.081)	2.04 (2.35)	2.05
Curly	$X^{(2,1)}$	63.1 (12.2)	(0.174)	(0.057)	1.38 (2.43)	2.94
	$X^{(2,2)}$	62.7 (9.5)	(0.174)	(0.056)	1.31 (1.85)	2.33
	$X^{(2,3)}$	64.0 (8.6)	(0.178)	(0.056)	1.29 (1.37)	2.43
Moe	$X^{(3,1)}$	70.8 (8.3)	(0.201)	(0.073)	1.90 (3.26)	1.75
	$X^{(3,2)}$	69.1 (7.7)	(0.194)	(0.073)	1.85 (3.34)	1.19
	$X^{(3,3)}$	71.5 (7.7)	(0.200)	(0.077)	1.97 (3.14)	0.59
Groucho	$X^{(4,1)}$	73.1 (9.5)	(0.244)	(0.092)	2.19 (2.77)	2.04
	$X^{(4,2)}$	71.9 (9.0)	(0.249)	(0.095)	2.24 (2.62)	1.02
	$X^{(4,3)}$	74.5 (9.4)	(0.285)	(0.114)	2.57 (2.65)	2.41
Harpo	$X^{(5,1)}$	66.8 (12.4)	(0.181)	(0.084)	1.85 (3.83)	4.02
	$X^{(5,2)}$	65.1 (13.2)	(0.208)	(0.095	1.94 (3.98)	5.27
	$X^{(5,3)}$	69.8 (12.3)	(0.226)	(0.111)	2.29 (3.76)	4.69
Zeppo	$X^{(6,1)}$	52.3 (12.2)	(0.171)	(0.053)	0.89 (1.48)	7.16
	$X^{(6,2)}$	51.7 (4.2)	(0.158)	(0.043)	0.70 (0.25)	1.36
	$X^{(6,3)}$	56.1 (5.7)	(0.204)	(0.058)	1.01 (0.34)	4.50

a. Numbers in parentheses are standard deviations.
b. Means for all runs is 0.000.

A.1 Larry

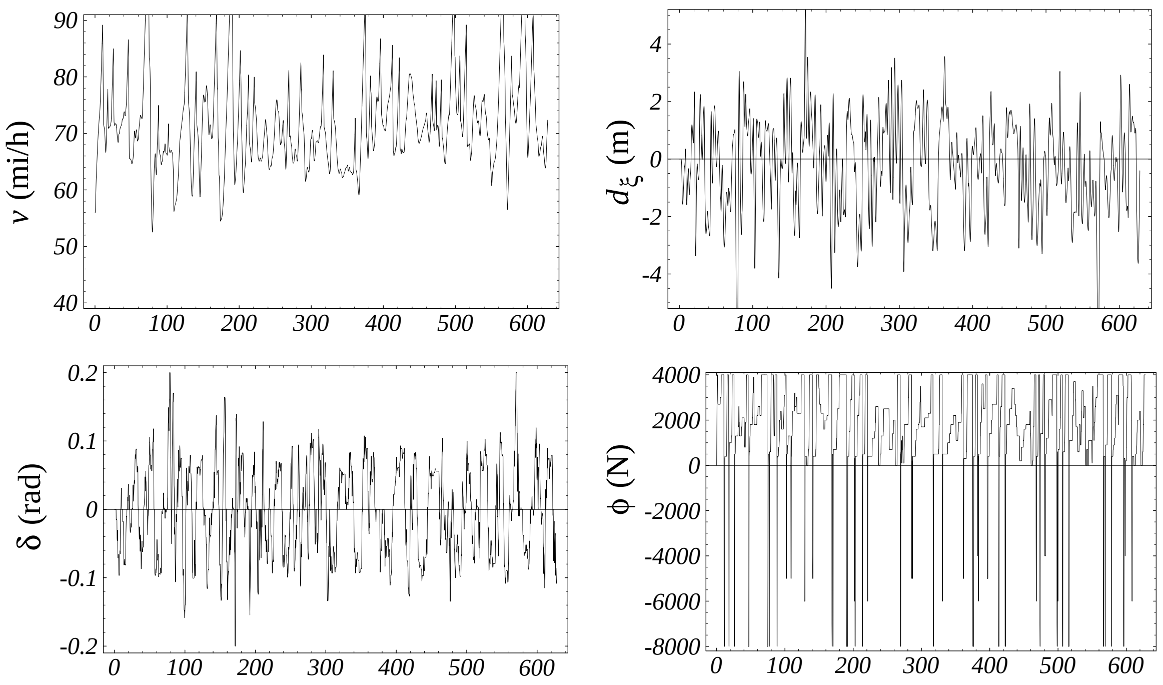


Figure A-1: Larry's run over road #1 as a function of time (sec).

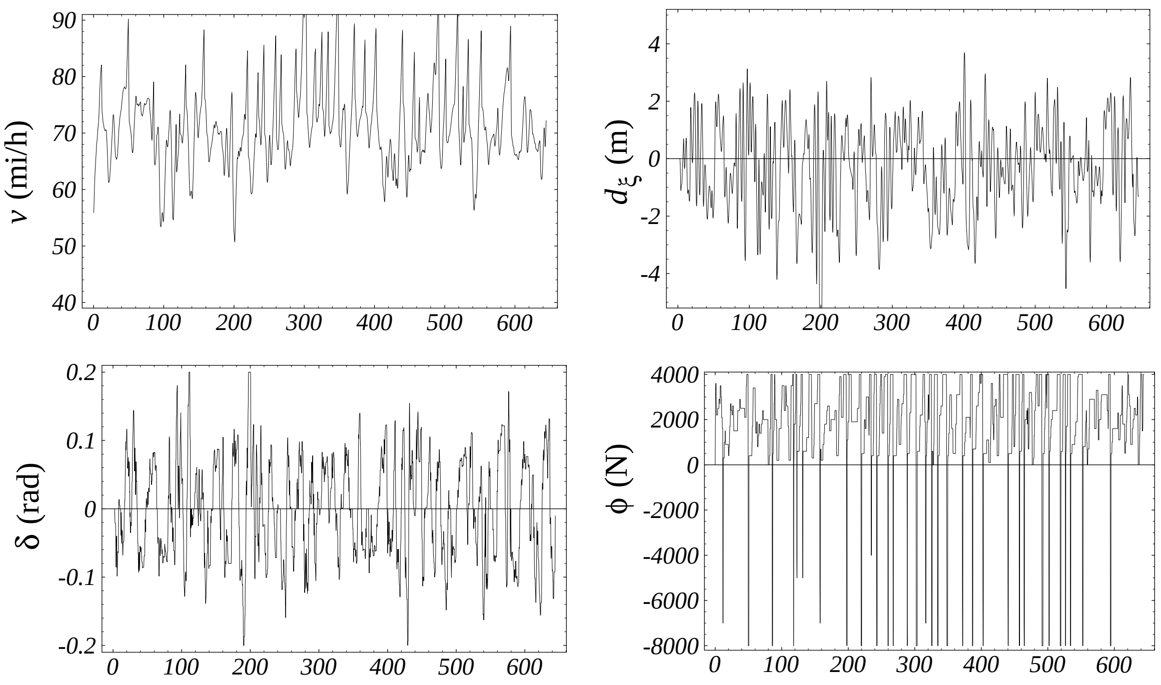


Figure A-2: Larry's run over road #2 as a function of time (sec).

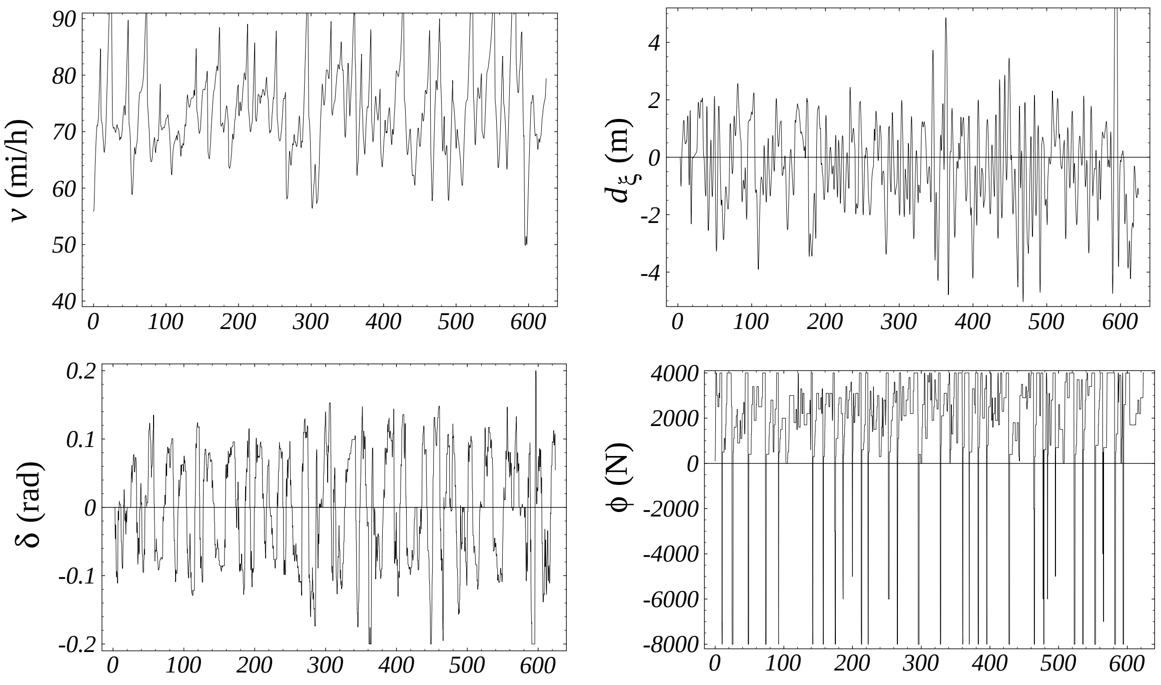


Figure A-3: Larry's run over road #3 as a function of time (sec).

A.2 Curly

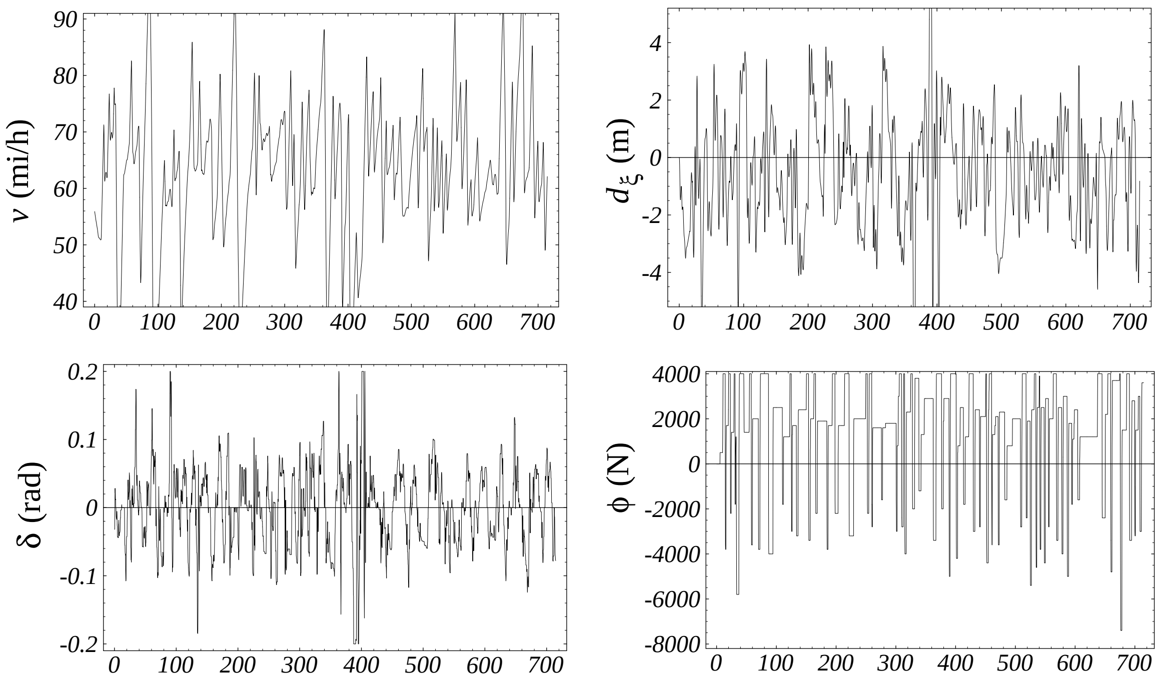


Figure A-4: Curly's run over road #1 as a function of time (sec).

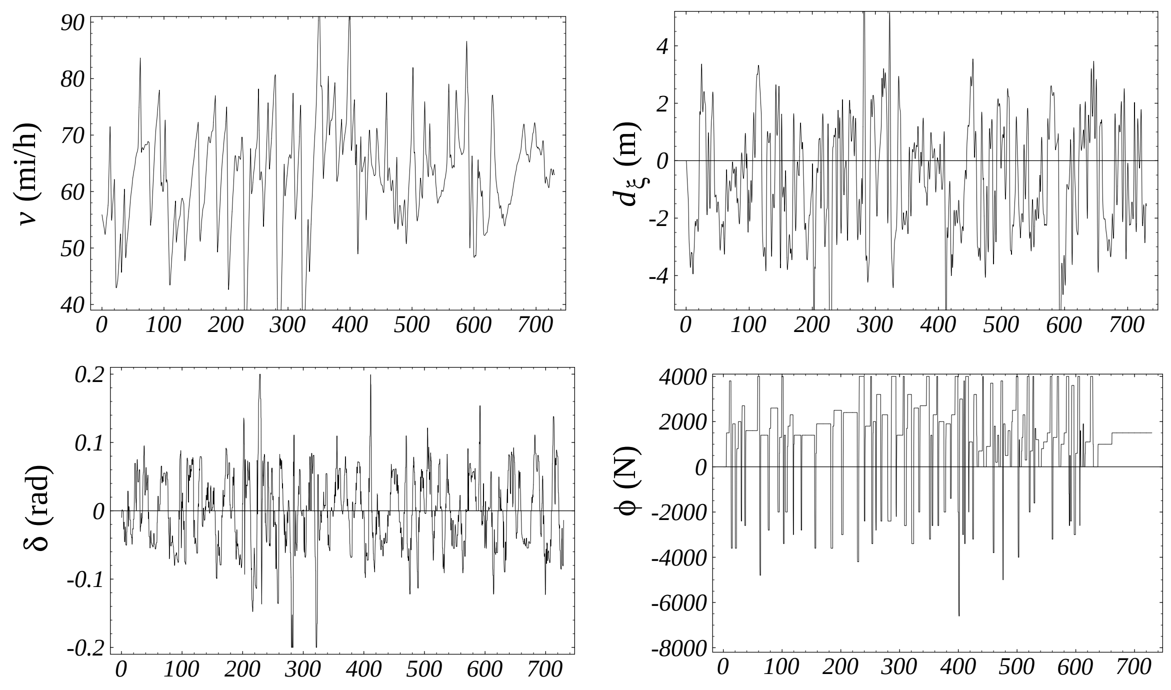


Figure A-5: Curly's run over road #2 as a function of time (sec).

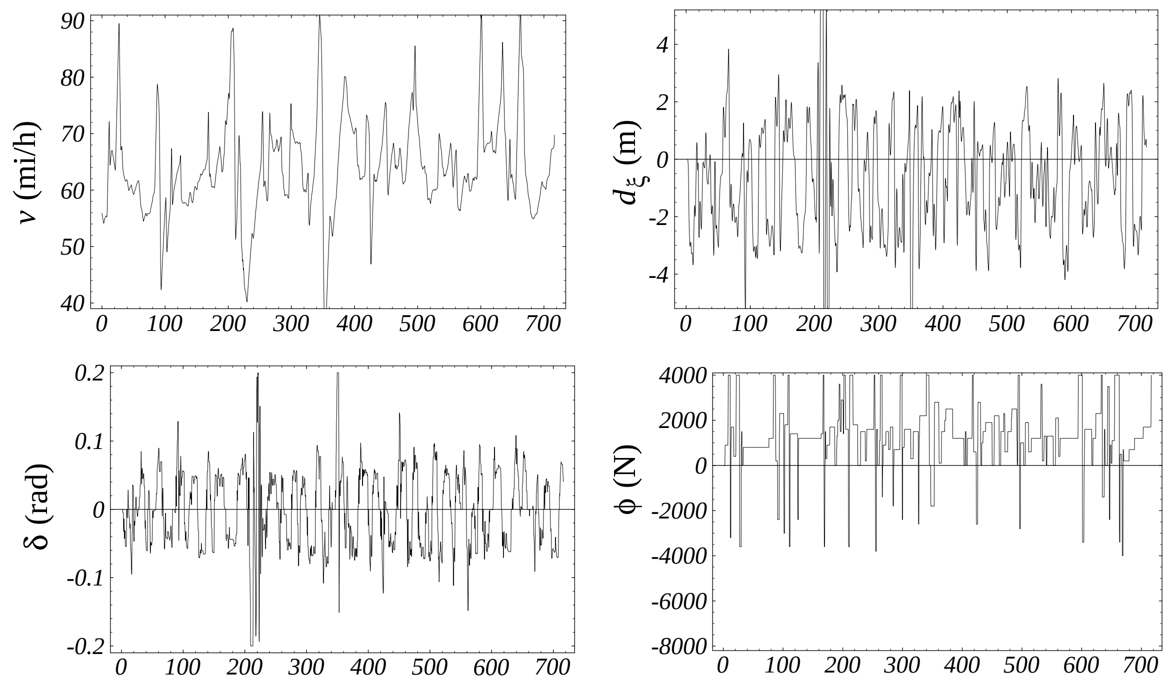


Figure A-6: Curly's run over road #3 as a function of time (sec).

A.3 Moe

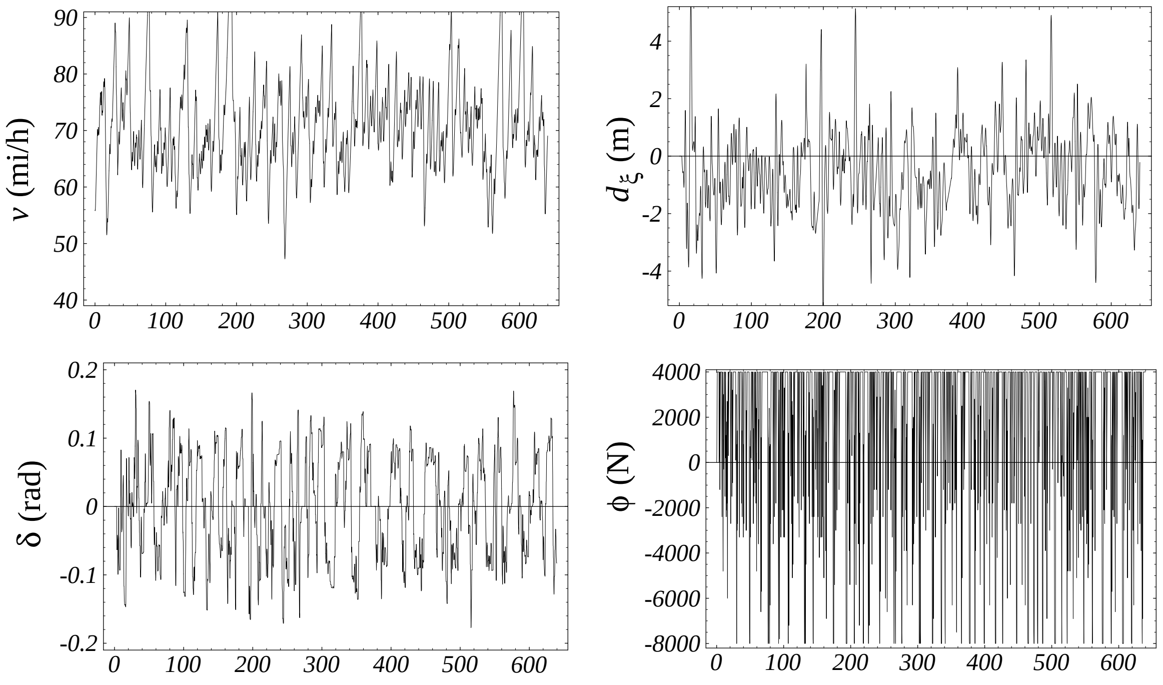


Figure A-7: Moe's run over road #1 as a function of time (sec).

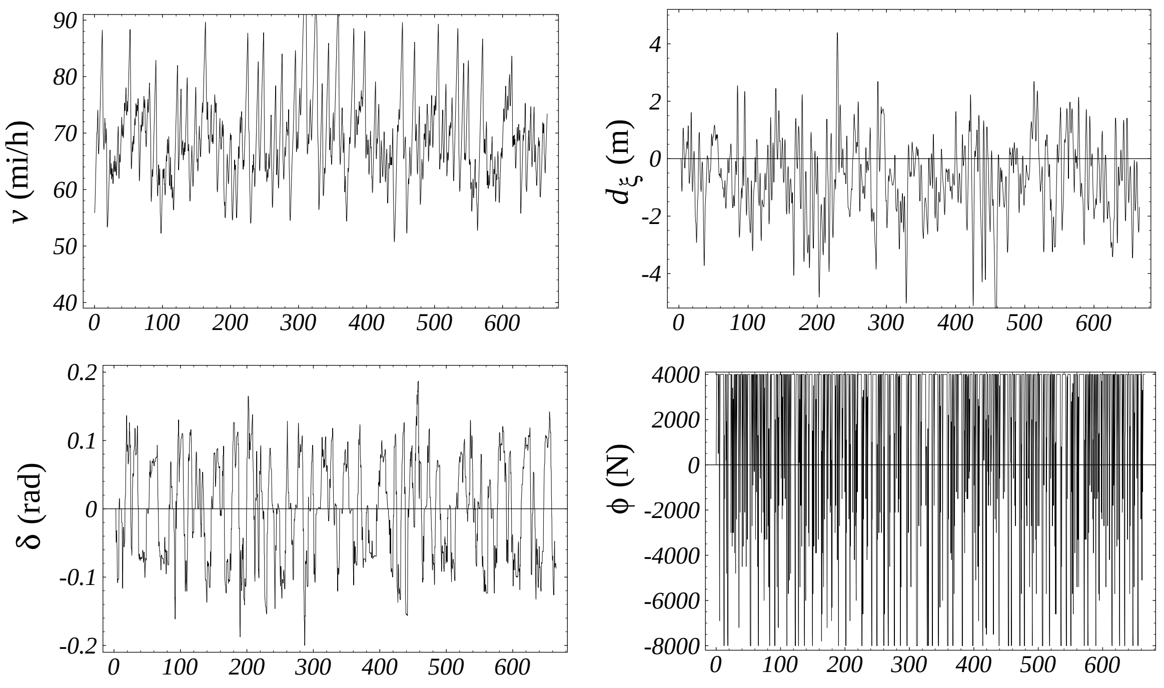


Figure A-8: Moe’s run over road #2 as a function of time (sec).

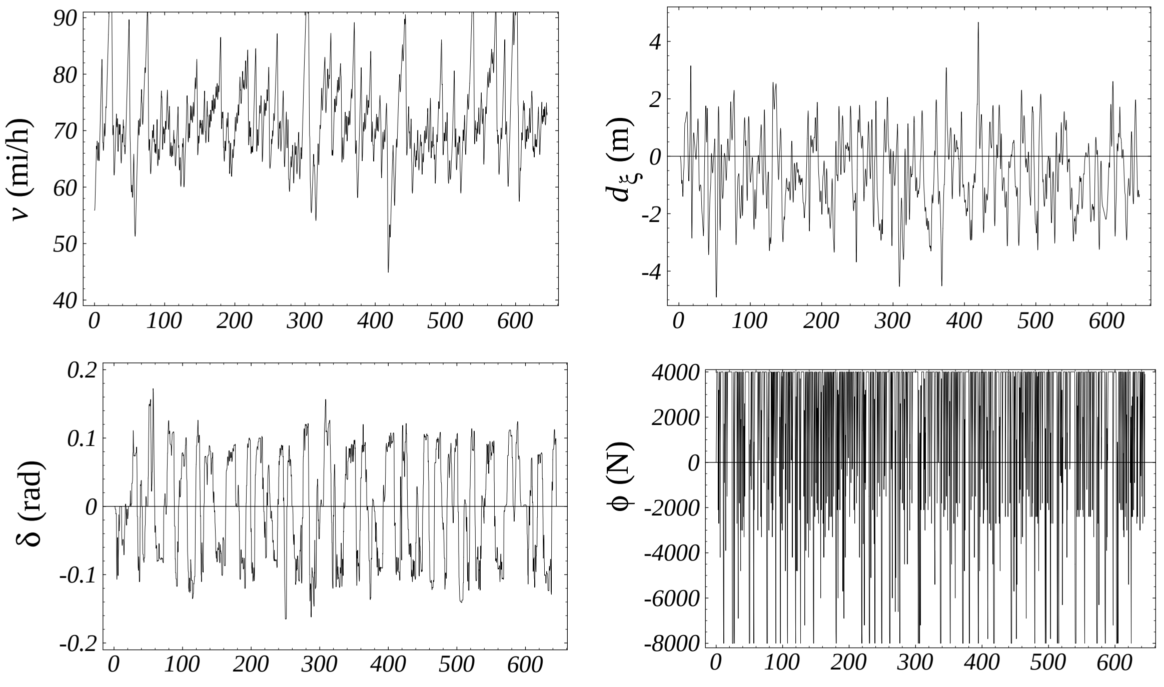


Figure A-9: Moe’s run over road #3 as a function of time (sec).

A.4 Groucho

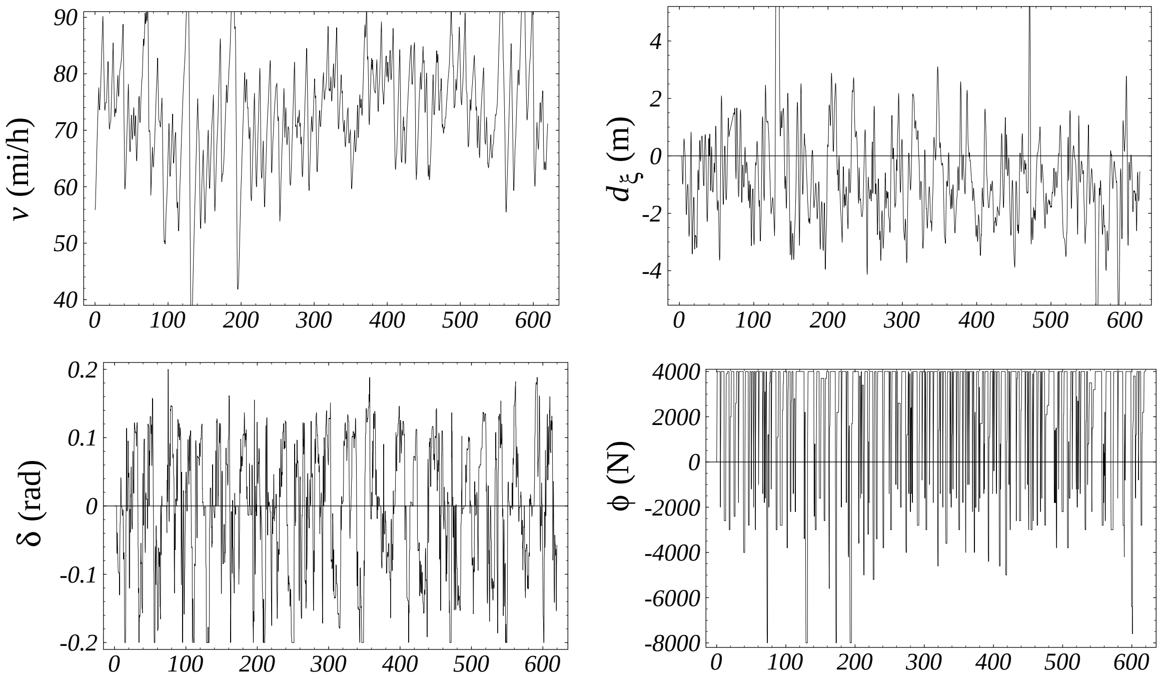


Figure A-10: Groucho's run over road #1 as a function of time (sec).

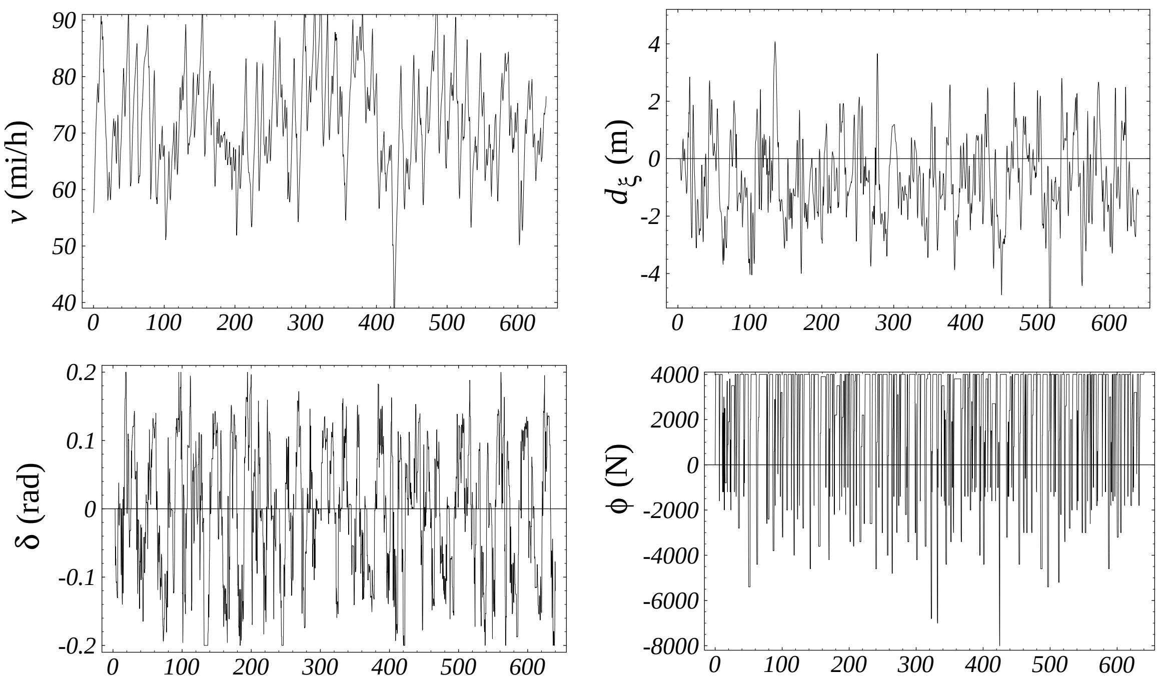


Figure A-11: Groucho's run over road #2 as a function of time (sec).

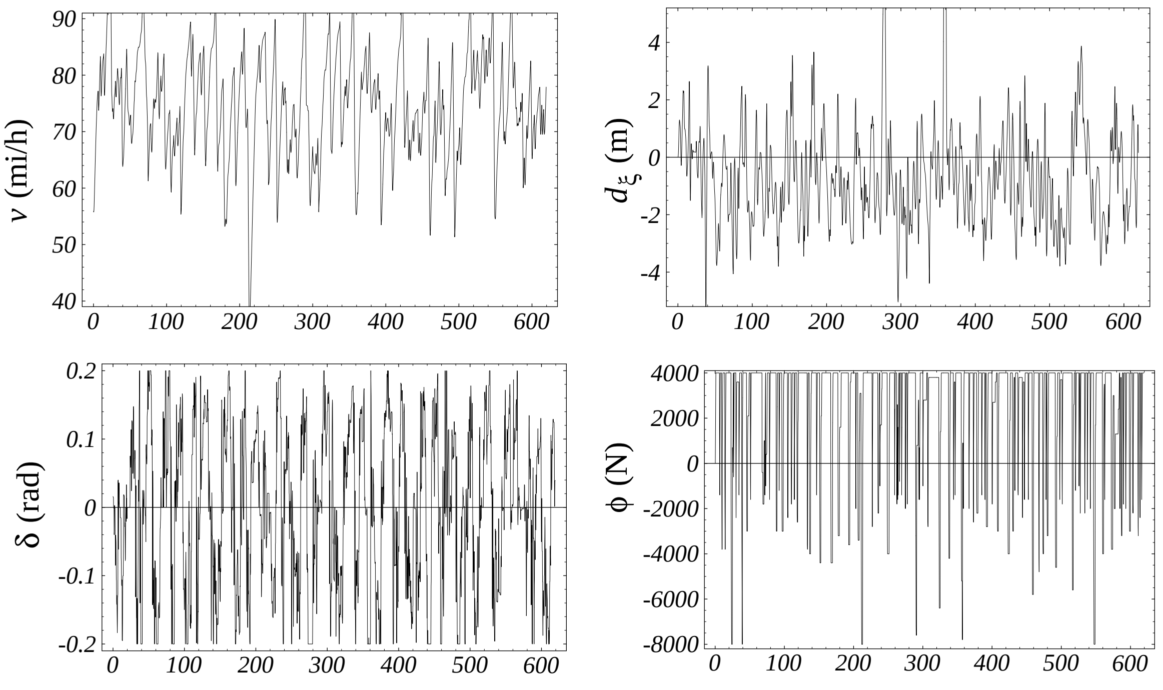


Figure A-12: Groucho's run over road #3 as a function of time (sec).

A.5 Harpo

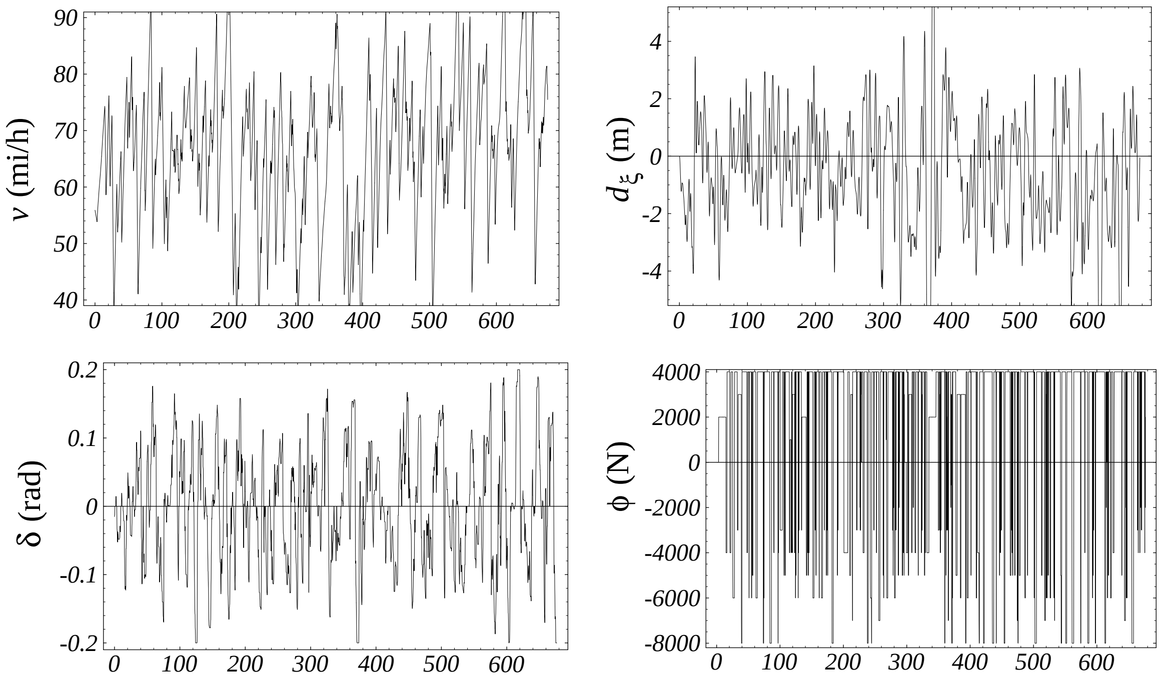


Figure A-13: Harpo's run over road #1 as a function of time (sec).

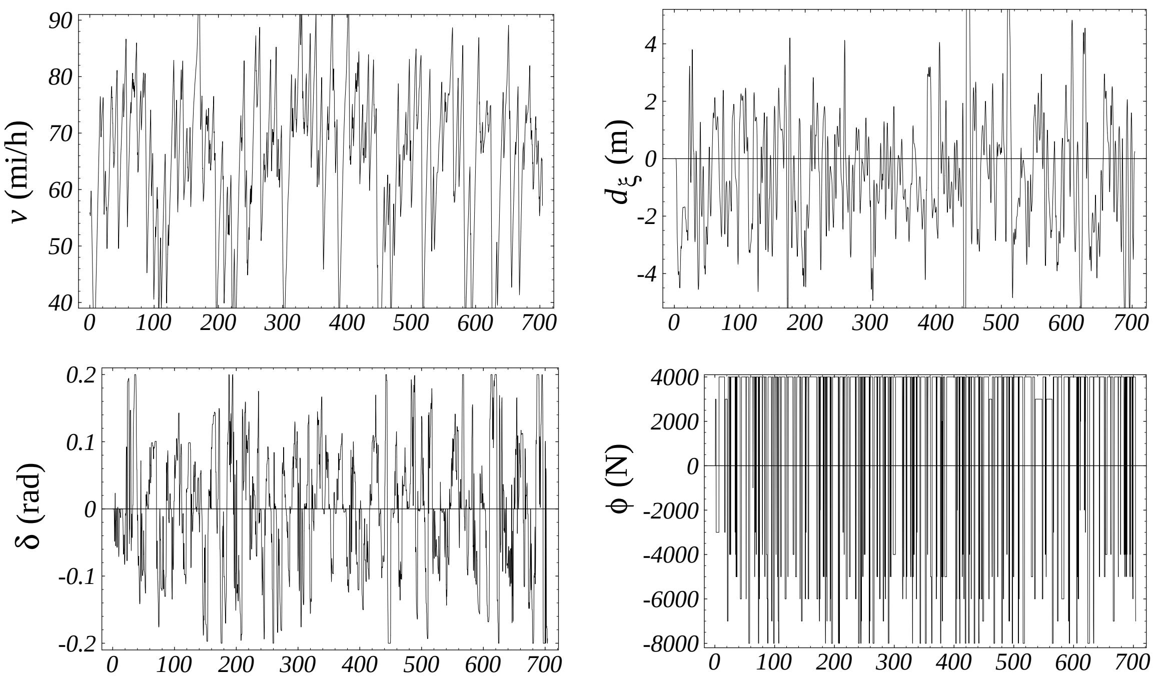


Figure A-14: Harpo's run over road #2 as a function of time (sec).

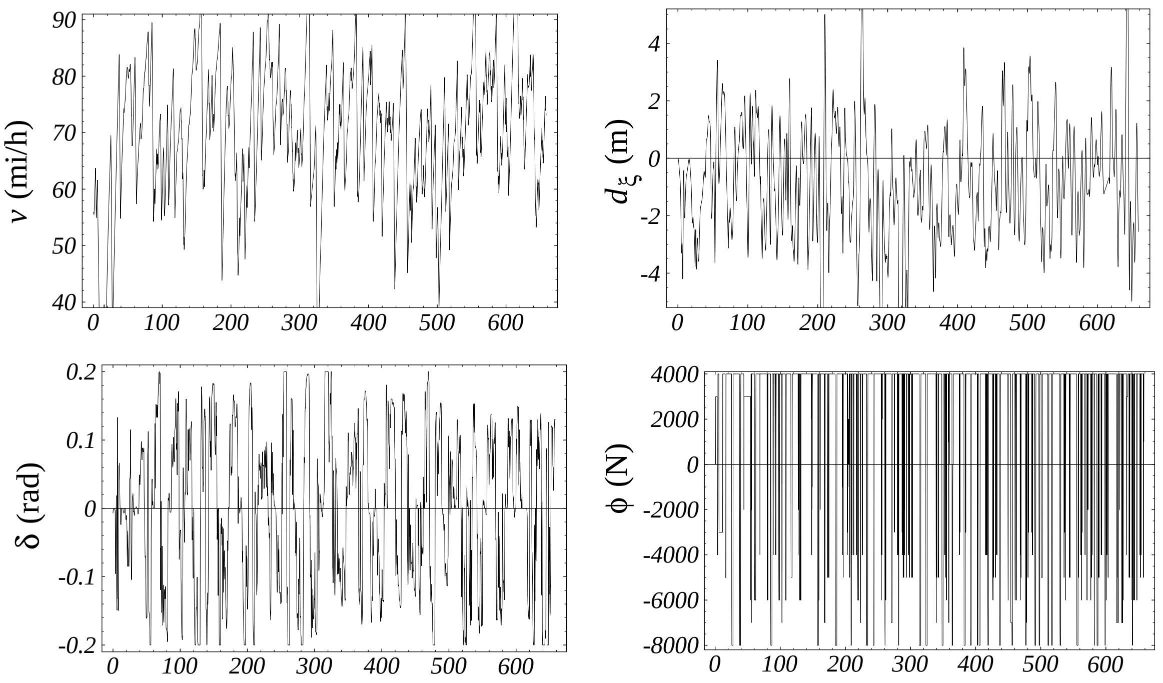


Figure A-15: Harpo's run over road #3 as a function of time (sec).

A.6 Zeppo

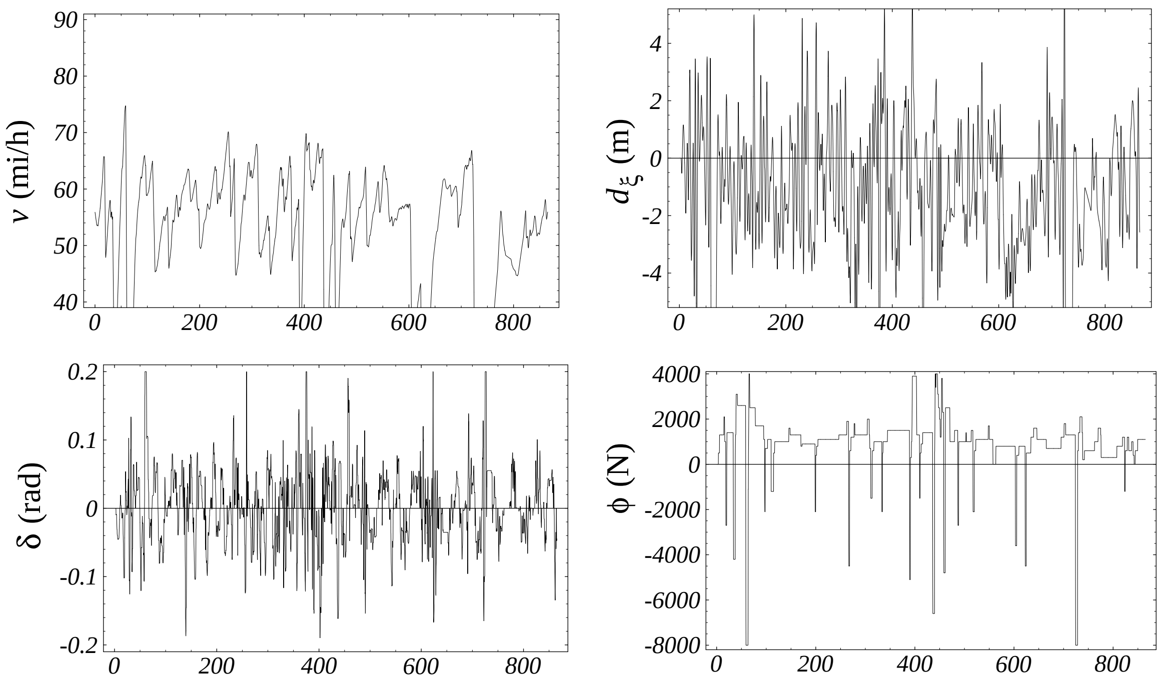


Figure A-16: Zeppo's run over road #1 as a function of time (sec).

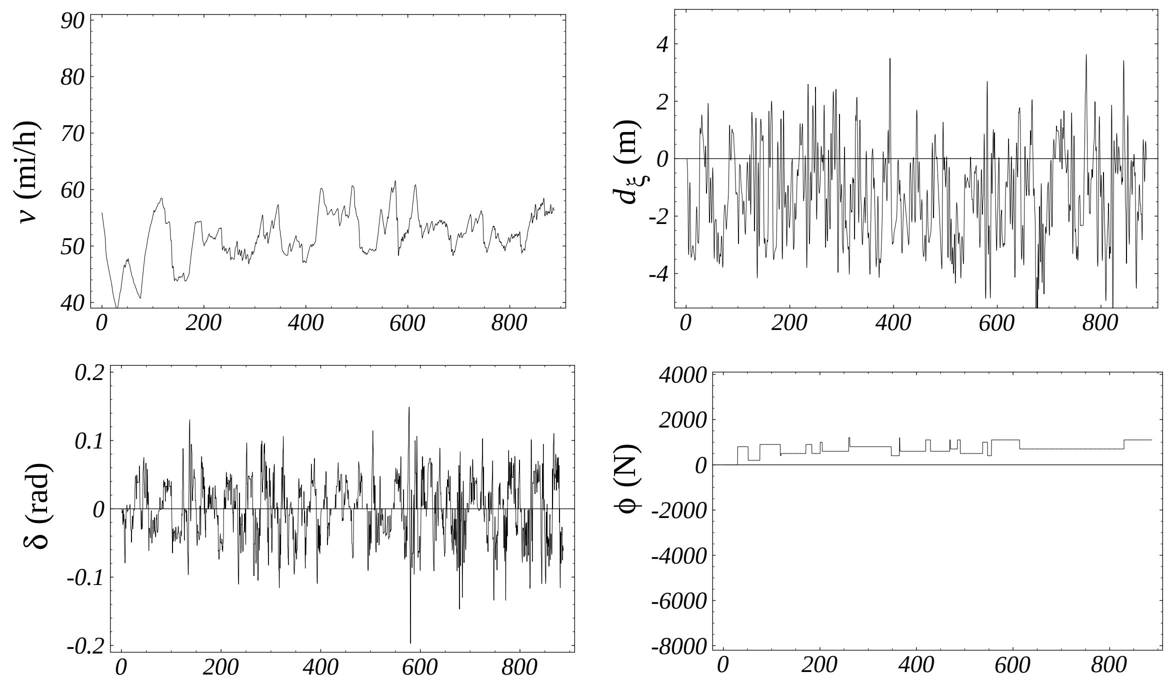


Figure A-17: Zeppo's run over road #2 as a function of time (sec).

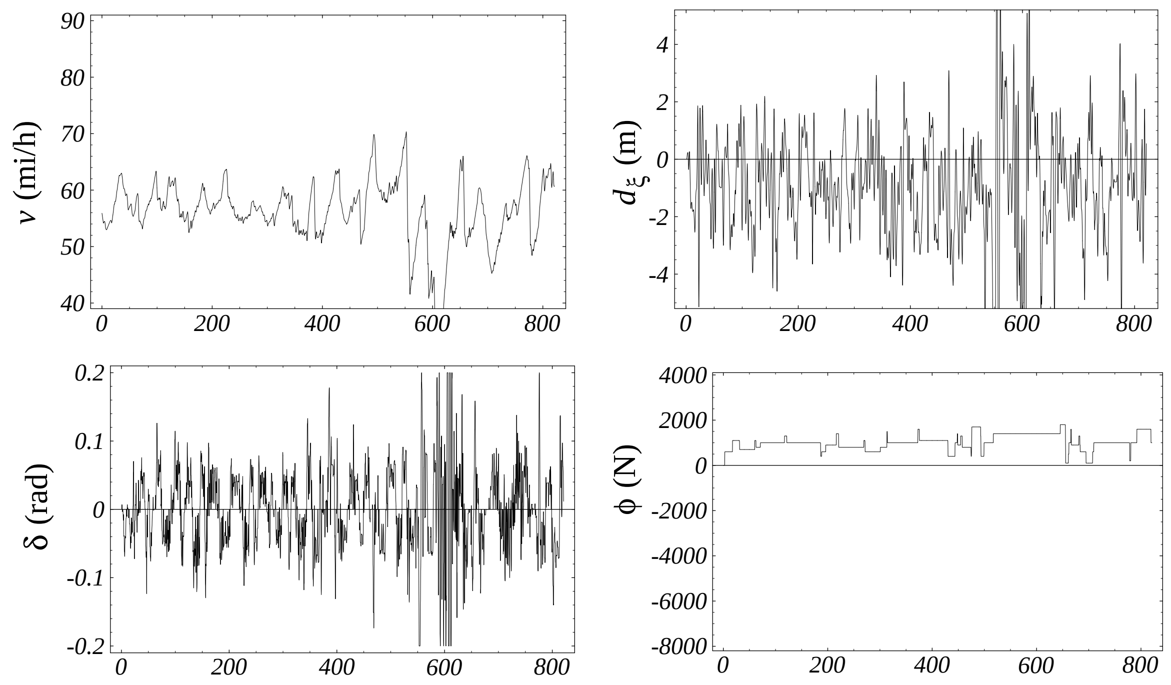


Figure A-18: Zeppo's run over road #3 as a function of time (sec).

Appendix B:
HMM Training

In this appendix, we briefly summarize the forward-backward and Baum-Welch algorithms. For a complete discussion of these algorithms, please see [94].

B.1 Forward-backward algorithm

The forward-backward algorithm is a computationally efficient algorithm for calculating $P(O|\lambda)$, for a discrete-output Hidden Markov Model $\lambda = \{A, B, \pi\}$ with n states, and a discrete observation sequence $O = \{o_1, o_2, \dots, o_T\}$, where,

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ : & : & : & : \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix},\ B = \begin{bmatrix} b_1(1) & b_2(1) & \dots & b_n(1) \\ b_1(2) & b_2(2) & \dots & b_n(2) \\ : & : & : & : \\ b_1(L) & b_2(L) & \dots & b_n(L) \end{bmatrix},\ \pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ : \\ \pi_n \end{bmatrix}. \tag{B-1}$$

It is also the first step of the Baum-Welch algorithm described in Section B.2. The forward algorithm is described below:

$$\alpha_1(i) = \pi_i b_i(o_1),\ i \in \{1, 2, \dots, n\} \tag{B-2}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{n} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}),\ t \in \{1, 2, \dots, T-1\},\ j \in \{1, 2, \dots, n\} \tag{B-3}$$

$$P(O|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \qquad \text{(B-4)}$$

For long observation sequences, the above algorithm is not numerically stable, as it will result in numerical underflow. By appropriate scaling, the forward algorithm can be modified to eliminate this problem:

$$\alpha_1(i) = \pi_i b_i(o_1),\ 1 \le i \le n \qquad \text{(B-5)}$$

$$\hat{\alpha}_1(i) = c_1 \alpha_1(i),\ i \in \{1, 2, \ldots, n\}\ \text{(scaling)} \qquad \text{(B-6)}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{n} \hat{\alpha}_t(i) a_{ij}\right] b_j(o_{t+1}),\ t \in \{1, 2, \ldots, T-1\},\ j \in \{1, 2, \ldots, n\} \qquad \text{(B-7)}$$

$$\hat{\alpha}_{t+1}(j) = c_{t+1} \alpha_{t+1}(j),\ t \in \{1, 2, \ldots, T-1\},\ j \in \{1, 2, \ldots, n\}\ \text{(scaling)} \qquad \text{(B-8)}$$

$$c_t = 1 / \left(\sum_{i=1}^{n} \alpha_t(i)\right),\ t \in \{1, 2, \ldots, T\}\ \text{(scaling)} \qquad \text{(B-9)}$$

$$P(O|\lambda) = 1 / \left(\prod_{t=1}^{T} c_t\right) \qquad \text{(B-10)}$$

Similarly, the backward algorithm (with scaling) is defined as follows:

$$\beta_T(i) = 1,\ 1 \le i \le n \qquad \text{(B-11)}$$

$$\hat{\beta}_T(i) = c_T \beta_T(i),\ 1 \le i \le n\ \text{(scaling)} \qquad \text{(B-12)}$$

$$\beta_t(i) = \sum_{j=1}^{n} a_{ij} b_j(o_{t+1}) \hat{\beta}_{t+1}(j)\ \ t \in \{T-1, T-2, \ldots, 1\},\ 1 \le i \le n \qquad \text{(B-13)}$$

$$\hat{\beta}_t(i) = c_t \beta_t(i),\ t \in \{T-1, T-2, \ldots, 1\},\ 1 \le i \le n\ \text{(scaling)} \qquad \text{(B-14)}$$

Note that the scaling coefficients c_t are chosen to be the same for the forward and backward algorithms.

B.2 Baum-Welch algorithm (with scaling)

Assume that we have multiple observation sequences $O^{(k)}$, of length T_k, $1 \le k \le K$. Furthermore, assume that we have a current estimate of the optimized Hidden Markov Model $\lambda = \{A, B, \pi\}$ with n states and L output observables. We would now like to generate a new estimate $\bar{\lambda} = \{\bar{A}, \bar{B}, \bar{\pi}\}$ which guarantees that,

$$\prod_{k=1}^{K} P(\bar{\lambda} | O^{(k)}) \ge \prod_{k=1}^{K} P(\lambda | O^{(k)}) \tag{B-15}$$

The Baum-Welch algorithm does just that. The state transition matrix A is updated by,

$$\overline{a_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \frac{\hat{\alpha}_t^k(i) \hat{\beta}_t^k(j)}{c_t^k}}, \ 1 \le i, j \le n \tag{B-16}$$

while the output probability distribution matrix B is updated by,

$$\overline{b_j}(l) = \frac{\sum_{k=1}^{K} \sum_{\substack{t = 1 \\ \text{s.t. } o_t = v_l}}^{T_k} \frac{\hat{\alpha}_t^k(i) \hat{\beta}_t^k(j)}{c_t^k}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \frac{\hat{\alpha}_t^k(i) \hat{\beta}_t^k(j)}{c_t^k}}, \ 1 \le l \le L, \ 1 \le j \le n \tag{B-17}$$

Rabiner provides an excellent and practical introduction to the Baum-Welch algorithm [94]. Unfortunately, in [94] the final equations summarizing the Baum-Welch algorithm — namely, equations (110) and (111), corresponding to equations (B-16) and (B-17) — are incorrect.

In equation (110), the upper summation limit for t should be T_k, not $(T_k - 1)$; otherwise, the last symbol for each observation sequence is ignored in the reestimation formula for B. Thus, equation (110) should read,

$$\left(\overline{b_j}(l) = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{\substack{t=1 \\ \text{s.t. } o_t = v_l}}^{T_k - 1} \alpha_t^k(i)\beta_t^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \alpha_t^k(i)\beta_t^k(j)} \right) \Rightarrow \left(\overline{b_j}(l) = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{\substack{t=1 \\ \text{s.t. } o_t = v_l}}^{T_k} \alpha_t^k(i)\beta_t^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k} \alpha_t^k(i)\beta_t^k(j)} \right) \tag{B-18}$$

The problem with equation (111) is two-fold. First note from equation (109),

$$\overline{a_{ij}} = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \alpha_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \alpha_t^k(i)\beta_t^k(j)} \tag{B-19}$$

that for multiple observation sequences, each term in the k-sum is weighted by $1/P_k$. The $1/P_k$ factor comes from the definition of the $\xi_t(i, j)$ and $\gamma_t(i)$ variables in equations (37) and (38), respectively. Now, consider equation (111) (in terms of the scaled forward-backward variables):

$$\overline{a_{ij}} = \frac{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\displaystyle\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i)\hat{\beta}_t^k(j)} \tag{B-20}$$

Rewriting the above equation in terms of unscaled variables, we get,

$$\overline{a_{ij}} = \frac{\sum_{k=1}^{K}\frac{1}{P_k}\sum_{t=1}^{T_k-1} C_t^k \alpha_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) D_{t+1}^k \beta_{t+1}^k(j)}{\sum_{k=1}^{K}\frac{1}{P_k}\sum_{t=1}^{T_k-1} C_t^k \alpha_t^k(i) D_t^k \beta_t^k(j)} \text{ } (\textit{Wrong!}) \qquad \text{(B-21)}$$

$$C_t^k D_{t+1}^k = C_T^k;\ C_t^k D_t^k = c_t^k C_T^k;\ C_T^k = \frac{1}{P_k} \text{ (by definition)} \qquad \text{(B-22)}$$

$$\overline{a_{ij}} = \frac{\sum_{k=1}^{K}\frac{C_T^k}{P_k}\sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^{K}\frac{C_T^k}{P_k}\sum_{t=1}^{T_k-1} c_t^k \alpha_t^k(i) \beta_t^k(j)} = \frac{\sum_{k=1}^{K}\frac{1}{P_k^2}\sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^{K}\frac{1}{P_k^2}\sum_{t=1}^{T_k-1} c_t^k \alpha_t^k(i) \beta_t^k(j)}$$

$$(\textit{Wrong!}) \qquad \text{(B-23)}$$

Thus, equation (111) makes two errors: (1) it inadvertently scales each observation sequence by $1/P_k^2$, and (2) it leaves an extra c_t^k term in the denominator. Equation (B-16) above corrects these errors. Using the incorrect equations leads to oscillating behavior of the Baum-Welch algorithm, which, theoretically, is guaranteed not to happen.

Appendix C: Author's Publications

The following is a complete list of journal and refereed conference publications derived from this work (in reverse chronological order):

[1] M. C. Nechyba and Y. Xu, "Stochastic Similarity for Validating Human Control Strategy Models," to appear in *IEEE Trans. on Robotics and Automation*, June, 1998.

[2] M. C. Nechyba and Y. Xu, "On Discontinuous Human Control Strategies," to appear in *Proc. IEEE Int. Conference on Robotics and Automation*, May, 1998.

[3] J. Song, Y. Xu, M. C. Nechyba and Y. Yam, "Two Performance Measures for Evaluating Human Control Strategy," to appear in *Proc. IEEE Int. Conference on Robotics and Automation*, May, 1998.

[4] M. C. Nechyba and Y. Xu, "Learning and Transfer of Human Real-Time Control Strategies," *Journal of Advanced Computational Intelligence*, vol. 1, no. 2, pp. 137-54, 1997.

[5] M. C. Nechyba and Y. Xu, "Human Control Strategy: Abstraction, Verification and Replication," *IEEE Control Systems Magazine*, vol. 17, no. 5, pp. 48-61, 1997.

[6] M. C. Nechyba and Y. Xu, “Cascade Neural Networks with Node-Decoupled Extended Kalman Filtering,” *Proc. IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 214-9, 1997.

[7] M. C. Nechyba and Y. Xu, “Stochastic Similarity for Validating Human Control Strategy Models,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 278-83, 1997.

[8] M. C. Nechyba and Y. Xu, “On the Fidelity of Human Skill Models,” *Proc. IEEE Int. Conference on Robotics and Automation*, vol. 3, pp. 2688-93, 1996.

[9] M. C. Nechyba and Y. Xu, “Human Skill Transfer: Neural Networks as Learners and Teachers,” *Proc. IEEE Int. Conference on Intelligent Robots and Systems*, vol. 3, pp. 314-9, 1995.

[10] M. C. Nechyba and Y. Xu, “Neural Network Approach to Control System Identification with Variable Activation Functions,” *Proc. IEEE Int. Symp. on Intelligent Control*, vol. 1, pp. 358-63, 1994.

Bibliography

[1] J. Albus, “A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (Cmac),” *Trans. ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 97, pp. 220-7, 1975.

[2] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice-Hall, Englewood Cliffs, 1979.

[3] P. J. Antsaklis, guest ed., Special Issue on Neural Networks in Control Systems, *IEEE Control System Magazine*, vol. 10, no. 3, pp. 3-87, 1990.

[4] P. J. Antsaklis, guest ed., Special Issue on Neural Networks in Control Systems, *IEEE Control System Magazine*, vol. 12, no. 2, pp. 8-57, 1992.

[5] H. Asada and S. Liu, “Transfer of Human Skills to Neural Net Robot Controllers,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 3, pp. 2442-2447, 1991.

[6] H. Asada and B. Yang, “Skill Acquisition from Human Experts Through Pattern Processing of Teaching Data,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 3, pp. 1302-7, 1989.

[7] T. Ash, “Dynamic Node Creation in Backpropagation Networks,” *Connection Science*, vol. 1, no. 4, pp. 365-75, 1989.

[8] K. J. Åström and T. J. McAvoy, “Intelligent Control: An Overview and Evaluation,” *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches,* D. A. White and D. A. Sofge, eds., pp. 3-34, Multiscience Press, New York, 1992.

[9] C. G. Atkeson, A. W. Moore and S. Schaal, “Locally Weighted Learning,” *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 11-73, 1997.

[10] C. G. Atkeson, A. W. Moore and S. Schaal, “Locally Weighted Learning for Control,” *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 75-113, 1997.

[11] S. Baluja and R. Caruana, “Removing Genetics from the Standard Genetic Algorithm,” *Proc. of the 12th Int. Conf. on Machine Learning*, vol. 1, pp. 38-46, 1995.

[12] R. Basri and D. Weinshall, “Distance Metric Between 3D Models and 2D Images for Recognition and Classification,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 43, no. 4, pp. 465-479, 1996.

[13] E. B. Bartlett, “Dynamic Node Architecture Learning: An Information Theoretic Approach,” *Neural Networks*, vol. 7, no. 1, pp. 129-40, 1994.

[14] A. G. Barto, R. S. Sutton and C. J. Watkins, “Learning and Sequential Decision Making,” *Learning and Computational Neuroscience*, ed. M. Gabriel and J. W. Moore, MIT Press, Cambridge, pp. 539-602, 1990.

[15] P. H. Batavia, “Driver Adaptive Warning Systems,” Technical Report, CMU-RI-TR-98-07, Carnegie Mellon University, 1998.

[16] L. E. Baum, T. Petrie, G. Soules and N. Weiss, “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains,” *Ann. Mathematical Statistics*, vol. 41, no. 1, pp. 164-71, 1970.

[17] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 19, no. 9, pp. 509-17, 1975.

[18] N. V. Bhat and T. J. McAvoy, "Determining Model Structure for Neural Models by Network Stripping," *Computers and Chemical Engineering*, vol. 16, no. 4, pp. 271-81, 1992.

[19] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[20] M. Boninsegna and M. Rossi, "Similarity Measures in Computer Vision," *Pattern Recognition Letters*, vol. 15, no. 12, pp. 1255-60, 1994.

[21] P. J. Brockwell and R. A. Davis, *Time Series: Theory and Methods*, 2nd. ed., Springer-Verlag, New York, 1991.

[22] P. Burrascano, "A Pruning Technique Maximizing Generalization," *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 347-50, 1993.

[23] G. Castellano, A. M. Fanelli and M. Pelillo, "An Empirical Comparison of Node Pruning Methods for Layered Feed-forward Neural Networks," *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 321-6, 1993.

[24] J. P. Cater, "Successfully Using Peak Learning Rates of 10 (and Greater) in Back-Propagation Networks with the Heuristic Learning Algorithm," *IEEE First Int. Conf. on Neural Networks*, vol. 2, pp. 645-51, 1987.

[25] K. Chen and R. D. Ervin, "Worldwide IVHS Activities: A Comparative Overview," *Proc. CONVERGENCE'92 — Int. Congress on Transportation Electronics*, pp. 339-49, 1992.

[26] W. C. Collier and R. J. Weiland, “Smart Cars, Smart Highways,” *IEEE Spectrum*, vol. 31, No. 4, pp. 27-33, 1994.

[27] G. Cybenko, “Approximation by Superposition of a Sigmoidal Function,” *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303-14, 1989.

[28] N. Delson and H. West, “Robot Programming by Human Demonstration: Adaptation and Inconsistency in Constrained Motion,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 30-6, 1996.

[29] K. Deng, A. Moore and M. C. Nechyba, “Learning to Recognize Time Series: Combining ARMA Models with Memory-Based Learning,” *Proc. IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 246-50, 1997.

[30] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[31] S. E. Fahlman, “An Empirical Study of Learning Speed in Back-Propagation Networks,” Technical Report, CMU-CS-TR-88-162, Carnegie Mellon University, 1988.

[32] S. E. Fahlman and C. Lebiere, “The Cascade-Correlation Learning Architecture,” Technical Report, CMU-CS-TR-91-100, Carnegie Mellon University, 1991.

[33] S. E. Fahlman, L. D. Baker and J. A. Boyan, “The Cascade 2 Learning Architecture,” Technical Report, CMU-CS-TR-96-184, Carnegie Mellon University, 1996.

[34] E. Fix and H. G. Armstrong, “Modeling Human Performance with Neural Networks,” *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 247-52, 1990.

[35] E. Fix and H. G. Armstrong, “Neural Network Based Human Performance Modeling,” *Proc. IEEE National Aerospace and Electronics Conf.*, vol. 3, pp. 1162-5, 1990.

[36] H. Friedrich, M. Kaiser and R. Dillman, “What Can Robots Learn From Humans?” *Annual Reviews in Control*, vol. 20, pp. 167-72, 1996.

[37] K. Funahashi, “On the Approximate Realization of Continuous Mappings by Neural Networks,” *Neural Net.*, vol. 2, no. 3, pp. 183-92, 1989.

[38] S. Geva and J. Sitte, “A Cartpole Experiment Benchmark for Trainable Controllers,” IEEE Control Systems Magazine, vol. 13, no. 5, pp. 40-51, 1993.

[39] C. G. Gingrich, D. R. Kuespert and T. J. McAvoy, “Modeling Human Operators Using Neural Networks,” *ISA Trans.*, vol. 31, no. 3, pp. 81-90, 1992.

[40] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.

[41] D. Gopher, M. Weil and T. Bareket, “The Transfer of Skill from a Computer Game Trainer to Actual Flight,” *Proc. Human Factors Society 36th Annual Meeting*, vol. 2, pp. 1285-1290, 1992.

[42] A. Guez and J. Selinsky, “A Trainable Neuromorphic Controller,” *Journal of Robotic Systems*, vol. 5, no. 4, pp. 363-88, 1988.

[43] V. Gullapalli, J. A. Franklin and H. Benbrahim, “Acquiring Robot Skills Via Reinforcement Learning,” *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 13-24, 1994.

[44] M. Hagiwara, “Removal of Hidden Units and Weights for Back Propagation Networks,” *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 351-54, 1993.

[45] B. Hannaford and P. Lee, “Hidden Markov Model Analysis of Force/Torque Information in Telemanipulation,” *Int. Journal Robotics Research*, vol. 10, no. 5, pp. 528-39, 1991.

[46] H. Hatwal and E. C. Mikulcik, “Some Inverse Solutions to an Automobile Path-Tracking Problem with Input Control of Steering and Brakes,” *Vehicle System Dynamics*, vol. 15, pp. 61-71, 1986.

[47] J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing, Redwood City, 1991.

[48] R. A. Hess, “Human-in-the-Loop Control,” *The Control Handbook*, ed. W. S. Levine, CRC Press, pp. 1497- 505, 1996.

[49] Y. Hiroshe, K. Yamashita and S. Hijiya, “Backpropagation Algorithm Which Varies the Number of Hidden Units,” *Neural Networks*, vol. 4, no. 1, pp. 61-6, 1991.

[50] G. E. Hovland, P. Sikka and B. J. MacCarragher, “Skill Acquisition from Human Demonstration Using a Hidden Markov Model,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 3, pp. 2706-11, 1997.

[51] X. D. Huang, Y. Ariki and M. A. Jack, *Hidden Markov Models for Speech Recognition*, Edinburgh University Press, Edinburgh, 1990.

[52] K. J. Hunt, *et. al.*, “Neural Networks for Control Systems - A Survey,” *Automatica*, vol. 28, no. 6, pp. 1083-112, 1992.

[53] W. Iba, “Modeling the Acquisition and Improvement of Motor Skills,” *Machine Learning: Proc. Eighth Int. Workshop on Machine Learning*, vol. 1, pp. 60-64, 1991.

[54] R. Jain, S. N. J. Murty, *et. al.*, “Similarity Measures for Image Databases,” in *Proc. IEEE Int. Conf. on Fuzzy System*s, vol. 3, pp. 1247-54, 1995.

[55] B. H. Juang and L. R. Rabiner, “A Probabilistic Distance Measure for Hidden Markov Models,” *AT&T Technical Journal*, vol. 64, no. 2, pp. 391-408, 1985.

[56] M. Kaiser, “Transfer of Elementary Skills via Human-Robot Interaction,” *Adaptive Behavior*, vol. 5, no. 3-4, pp. 249-80, 1997.

[57] S. B. Kang, “Automatic Robot Instruction from Human Demonstration,” Ph.D. Thesis, The Robotics Institute, Carnegie Mellon University, 1994.

[58] S. Kollias and D. Anastassiou, “An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks,” *IEEE Trans. on Circuits and Systems*, vol. 36, no. 8, pp. 1092-101, 1989.

[59] K. Kosuge, T. Fukuda and H. Asada, “Acquisition of Human Skills for Robotic Systems,” *Proc. IEEE Int. Symp. on Intelligent Control*, pp. 469-74, 1991.

[60] U. Kramer, “On the Application of Fuzzy Sets to the Analysis of the System-Driver-Vehicle Environment,” *Automatica*, vol. 21, no. 1, pp. 101-7, 1985.

[61] A. Kundu, G. C. Chen and C. E. Persons, “Transient Sonar Signal Classification Using Hidden Markov Models and Neural Nets,” *IEEE Jour. Oceanic Engineering*, vol. 19, no. 1, pp. 87-99, 1994.

[62] K. Y. Kupeev and H. J. Wolfson, “On Shape Similarity,” *Proc. of 12th IAPR Int. Conf. on Pattern Recognition*, vol. 1, pp. 227-31, 1994.

[63] C. C. Lee, “Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part I,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 404-18, 1990.

[64] C. C. Lee, “Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part II,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 419-35, 1990.

[65] C. Lee, “Transferring Human Skills to Robots via Task Demonstrations in Virtual Environments,” Ph.D. Thesis Proposal, Carnegie Mellon University, 1997.

[66] S. Lee and J. Chen, “Skill Learning from Observations,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 4, pp. 3245-250, 1994.

[67] S. Lee and M. H. Kim, “Cognitive Control of Dynamic Systems,” *Proc. IEEE Int. Symp. on Intelligent Control*, pp. 455-60, 1987.

[68] S. Lee and M. H. Kim, “Learning Expert Systems for Robot Fine Motion Control,” *Proc. IEEE Int. Symp. on Intelligent Control*, pp. 534-44, 1988.

[69] Y. Linde, A. Buzo and R. M. Gray, “An Algorithm for Vector Quantizer Design,” *IEEE Trans. Communication*, vol. COM-28, no. 1, pp. 84-95, 1980.

[70] S. Liu and H. Asada, “Transferring Manipulative Skills to Robots: Representation and Acquisition of Tool Manipulative Skills Using a Process Dynamics Model,” *Trans. ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 114, pp. 220-8, 1992.

[71] M. C. Mackey and L. Glass, “Oscillations and Chaos in Physiological Control Systems,” *Science*, vol. 197, no. 4300, pp. 287-9, 1977.

[72] D. T. McRuer and E. S. Krendel, “Human Dynamics in Man-Machine Systems,” *Automatica*, vol. 16, no. 3, pp. 237-53, 1980.

[73] W. T. Miller, R. S. Sutton and P. I. Werbos, eds., “Neural Networks For Control,” MIT Press, Cambridge, 1990.

[74] A. Modjtahedzadeh and R. A. Hess, “A Model of Driver Steering Control Behavior for Use in Assessing Vehicle Handling Qualities,” *Trans. ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 115, no. 3, pp. 456-64, 1993.

[75] J. Moody and C. Darken, “Fast Learning in Networks of Locally Tuned Processing Units,” *Neural Computation*, vol. 1, no. 2, pp. 281-94, 1989.

[76] A. W. Moore and C. G. Atkeson, “Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time,” *Machine Learning*, vol 13, no. 1, pp. 103-30, 1993.

[77] M. C. Mozer and P. Smolensky, “Skeletonization: A Technique for Trimming the Fat From a Network Via Relevance Assessment,” *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, ed., Morgan Kaufmann Publishers, pp. 107-15, 1989.

[78] T. M. Nabhan and A. Y. Zomaya, “Toward Generating Neural Network Structures for Function Approximation,” *Neural Networks*, vol. 7, no. 1, pp. 89-99, 1994.

[79] K. S. Narendra and K. Parthasarathy, “Identification and Control of Dynamical Systems Using Neural Networks,” *IEEE Trans. on Neural Networks*, vol. 1, no. 1, pp. 4-27, 1990.

[80] M. C. Nechyba and Y. Xu, “Learning and Transfer of Human Real-Time Control Strategies,” *Journal of Advanced Computational Intelligence*, vol. 1, no. 2, pp. 137-54, 1997.

[81] M. C. Nechyba and Y. Xu, “Neural Network Approach to Control System Identification with Variable Activation Functions,” *Proc. IEEE Int. Symp. on Intelligent Control*, vol. 1, pp. 358-63, 1994.

[82] S. Neuser, J. Nijhuis, *et. al.*, “Neurocontrol for Lateral Vehicle Guidance,” *IEEE Micro*, vol. 13, no. 1, pp. 57-66, 1993.

[83] D. O’Hare and S. Roscoe, *Flight Deck Performance: The Human Factor*, Iowa State University Press, Ames, 1990.

[84] S. Omohundro, “Bumptrees for Efficient Function, Constraint, and Classification Learning,” *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody and D. S. Touretzky, eds, Morgan Kaufmann Publishers, pp. 693-9, 1991.

[85] E. H. Park, et. al., “Adaptive Learning of Human Motion by a Telerobot Using a Neural Network Model as a Teacher,” *Computer and Industrial Engineering*, vol. 27, pp. 453-6, 1994.

[86] A. Pentland and A. Liu, “Toward Augmented Control Systems,” *Proc. Intelligent Vehicles*, vol. 1, pp. 350-5, 1995.

[87] D. Plaut, S. Nowlan and G. Hinton, “Experiment on Learning by Backpropagation,” Technical Report, CMU-CS-86-126, Carnegie Mellon University, 1986.

[88] D. A. Pomerleau and T. Jochem, “Rapidly Adapting Machine Vision for Automated Vehicle Steering,” *IEEE Expert*, vol. 11, no. 2, pp. 19-27, 1996.

[89] D. A. Pomerleau, “Neural Network Perception for Mobile Robot Guidance,” Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1992.

[90] D. A. Pomerleau, “Reliability Estimation for Neural Network Based Autonomous Driving,” *Robotics and Autonomous Systems*, vol. 12, no. 3-4, pp. 113-9, 1994.

[91] W. H. Press, et. al., *Numerical Recipes in C: The Art of Scientific Computing, 2nd ed.*, Cambrige University Press, Cambridge, 1992.

[92] G. V. Puskorius and L. A. Feldkamp, “Decoupled Extended Kalman Filter Training of Feedforward Layered Networks,” *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 771-7, 1991.

[93] S. Qin, H. Su and T. J. McAvoy, “Comparison of Four Neural Net Learning Methods for Dynamic System Identification,” *IEEE Trans. on Neural Networks*, vol. 3, no. 1, pp. 122-30, 1992.

[94] L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proc. IEEE*, vol. 77, no. 2, pp. 257-86, 1989.

[95] L. R. Rabiner, B. H. Juang, S. E. Levinson and M. M. Sondhi, "Some Properties of Continuous Hidden Markov Model Representations," *AT&T Technical Journal*, vol. 64, no. 6, pp. 1211-1222, 1986.

[96] G. Radons, J. D. Becker, B. Dulfer and J. Kruger, "Analysis, Classification and Coding of Multielectrode Spike Trains with Hidden Markov Models," *Biological Cybernetics*, vol. 71, no. 4, pp. 359-73, 1994.

[97] K. R. Rao and D. F. Elliott, *Fast Transforms: Algorithms, Analyses and Applications*, Academic Press, New York, 1982.

[98] D. E. Rumelhart, J. L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, MIT Press, Cambridge, 1986.

[99] T. Samad, "Neurocontrol: Concepts and Applications," *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, vol. 1, pp. 369-74, 1992.

[100] S. Schaal and C. G. Atkeson, "Memory-Based Robot Learning," *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 4, pp. 2928-33, 1994.

[101] S. Schaal and C. G. Atkeson, "Robot Juggling: Implementation of Memory-Based Learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 57-71, 1994.

[102] J. G. Schneider, "Robot Skill Learning Through Intelligent Experimentation," Ph.D. Thesis, School of Computer Science, University of Rochester, 1995.

[103] W. L. Shebilske and J. W. Regian, "Video Games, Training, and Investigating Complex Skills," *Proc. Human Factors Society 36th Annual Meeting*, vol. 2, pp. 1296-1300, 1992.

[104] T. B. Sheridan, “Space Teleoperation Through Time Delay: Review and Prognosis,” *IEEE Trans. on Robotics and Automation*, vol. 9, no. 5, pp. 592-606, 1993.

[105] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*, Cambridge Press, Cambrdige, 1992.

[106] K. Shimokura and S. Liu, “Programming Deburring Robots Based on Human Demonstration with Direct Burr Size Measurement,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 572-7, 1994.

[107] H. Y. Shum, M. Hebert and K. Ikeuchi, “On 3D Shape Similarity,” Technical Report, CMU-CS-95-212, Carnegie Mellon University, 1995.

[108] S. Singhal and L. Wu, “Training Multilayer Perceptrons with the Extended Kalman Algorithm,” *Advances in Neural Information Processing Systems 1*, ed. Touretzky, D. S., Morgan Kaufmann Publishers, pp. 133-40, 1989.

[109] M. Skubic and R. A. Volz, “Learning Force Sensory Patterns and Skills From Human Demonstration,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 284-90, 1997.

[110] J. Song, Y. Xu, M. C. Nechyba and Y. Yam, “Two Performance Measures for Evaluating Human Control Strategy,” to appear in *Proc. IEEE Int. Conference on Robotics and Automation*, May, 1998.

[111] J. Song, Y. Xu, Y. Yam and M. C. Nechyba, “Optimization of Human Control Strategies with Simultaneously Perturbed Stochastic Approximation,” *submitted to Proc. IEEE Int. Conference on Intelligent Robots and Systems*, October, 1998.

[112] L. G. Sotelino, M. Saerens and H. Bersini, “Classification of Temporal Trajectories by Continuous-Time Recurrent Nets,” *Neural Networks*, vol. 7, no. 5, pp. 767-76, 1994.

[113] J. C. Spall, “Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation,” *IEEE Trans. on Automation Control*, vol. 37, no. 3, pp. 332-41, 1992.

[114] M. Sun, G. Burk and R. J. Sclabassi, “Measurement of Signal Similarity Using the Maxima of the Wavelet Transform,” *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 583-586, 1993.

[115] R. S. Sutton, “Learning to Predict by the Methods of Temporal Differences,” *Machine Learning*, vol. 3, no. 1, pp. 9-44, 1988.

[116] R. Sutton and D. R. Towill, “Modeling the Helmsan in a Ship Steering System Using Fuzzy Sets,” *Analysis, Design and Evaluation of Man-Machine Systems: Selected Papers from the Third IFAC Conference*, vol. 1, pp. 157-62, 1988.

[117] H. H. Thodberg, “Improving Generalization of Neural Networks Through Pruning,” *Int. Journal of Neural Systems*, vol. 1, no. 4, pp. 317-26, 1991.

[118] A. Ude, “Trajectory Generation from Noisy Positions of Object Features for Teaching Robot Paths,” *Robotics and Autonomous Systems*, vol. 11, no. 2, pp. 113-27, 1993.

[119] R. M. Voyles, J. D. Morrow and P. K. Khosla, “Towards Gesture-Based Programming: Shape from Motion Primordial Learning of Sensorimotor Primitives,” *Robotics and Autonomous Systems*, vol. 22, no. 3-4, pp. 361-75, 1997.

[120] C. J. Watkins, “Learning from Delayed Rewards,” Ph.D. Thesis, King’s College, University of Cambridge, 1989.

[121] M. Werman and D. Weinshall, “Similarity and Affine Invariant Distances Between 2D Point Sets,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 810-14, 1995.

[122] D. A. White and D. A. Sofge, eds., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, Multiscience Press, New York, 1992.

[123] J. Yamato, S. Kurakake, A. Tomono and K. Ishii, “Human Action Recognition Using HMM with Category Separated Vector Quantization,” *Trans. Institute of Electronics, Information and Communication Engineers D-II*, vol. J77D-II, no. 7, pp. 1311-18, 1994.

[124] J. Yamato, J. Ohya and K. Ishii, “REcognizing Human Action in Time-sequential Images Using Hidden Markov Models,” *Trans. Institute of Electronics, Information, and Communication Engineers D-II*, vol. J76D-II, no. 12, pp. 2556-2563, 1993.

[125] B. Yang and H. Asada, “Hybrid Linguistic/Numeric Control of Deburring Robots Based on Human Skills,” *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 1467-74, 1992.

[126] J. Yang, Y. Xu and C. S. Chen, “Hidden Markov Model Approach to Skill Learning and its Application to Telerobotics,” *IEEE Trans. on Robotics and Automation*, vol. 10, no. 5, pp. 621-31, 1994.

[127] J. Yang, Y. Xu and C. S. Chen, “Human Action Learning via Hidden Markov Models,” *IEEE Trans. on Systems, Man, and Cybernetics — Part A: Systems and Humans*, vol. 27, no. 1, pp. 34-44, 1997.