Stabilizing Human Control Strategies through Reinforcement Learning

Michael C. Nechyba
nechyba@mil.ufl.edu

J. Andrew Bagnell
dbagnell@mil.ufl.edu

Machine Intelligence Laboratory
Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611-6200

Abstract

In attempting to build advanced robots with sophisticated intelligent behaviors, the modern roboticist does not have to look far to find examples of such behavior. Humans are, and for the foreseeable future remain our best and only example of true intelligence. In comparison, even advanced robots are still embarrassingly stupid. Consequently, one popular approach for imparting intelligent behaviors to robots and other machines abstracts models of human control strategy (HCS), learned directly from human control data. This type of approach can be broadly classified as "learning through observation." A competing approach, which builds up complex behaviors through exploration and optimization over time, is reinforcement learning. We seek to unite these two approaches, previously considered disparate, and show that each approach, in fact, complements the other. Specifically, we propose a new algorithm, rooted in reinforcement learning, for stabilizing learned models of human control strategy. In this paper, we first describe the real-time driving simulator which we have developed for investigating human control strategies. Next, we motivate and describe our framework for modeling human control strategies. We then illustrate how the resulting HCS models can be stabilized through reinforcement learning and finally report some positive experimental results with the proposed algorithm.

1. Introduction

Models of human skill, which accurately emulate dynamic human behavior, have far reaching potential in areas ranging from robotics to virtual reality to the intelligent vehicle highway system. In robotics, such models could encapsulate necessary intelligent control behaviors. For some robotic applications including highway driving, efforts are underway to automate tasks that have been traditionally carried out by humans. Also, in simulation, dynamic models of human behavior would not only enhance the realism of virtual reality games, but also aid in the analysis of large-scale human-in-the-loop systems.

Thus, a number of different researchers have endeavored in recent years to abstract models of human skill directly from observed human input-output data (see [1,2] for overviews of the literature). In our work, we focus on a particular class of human skill, which we refer to as *human control strategy (HCS)*. In terms of complexity, human control strategy lies between low-level feedback control and high-level reasoning; as such, the term encompasses a wide range of physical tasks with a reasonably well-defined input-output representation. Two classic examples of human control strategy are teleoperation of robots in remote environment and automobile driving.

Unfortunately, capturing intelligent behaviors through human modeling suffers from some potential weaknesses. Because human control strategies are dynamic, nonlinear, stochastic processes, *analytic* models of human actions tends to be quite difficult, if not impossible, to abstract. Therefore, HCS models are usually derived *empirically*, rather than analytically from real-time human input-output data. As such, traditional performance or stability guarantees, like those in linear control for example, are typically not available. Moreover, the performance of the learned HCS models is necessarily limited by the expertise of the human trainer. It would be naive to expect that learning algorithms, which rely exclusively on the human's training data, will generate models superior to the human trainer.

In previous work, we have sought to address these issues through task-specific performance measures and post-training performance optimization [3]. Here, however, we propose a new algorithm for improving the performance of learned HCS models through *reinforcement learning*. Reinforcement learning denotes a class of adaptive techniques that seek to learn to predict and control the behavior of an autonomous agent through that agent's interaction with his/her environment. Modern reinforcement learning borrows heavily from the fields of operations research and optimal control; in particular, the agent's environment is approximated as a Markov Decision Process (MDP) [4], so that the agent is asked to maximize rewards emitted by the MDP.

Reinforcement learning offers powerful techniques with proven success in solving problems that can be modeled within the MDP framework [5, 6]. It too, however, suffers from some significant weaknesses. First, reinforcement learning techniques often do not scale well to problems with high-dimensional input spaces. The universe of admissible policies in high-dimensional input spaces often becomes so large that the learning agent cannot even begin to explore all possible options in a reasonable amount of time. Furthermore, much of the literature in reinforcement learning deals only with problems where the agent has perfect knowledge about the state of his/her environment. This condition is rarely met in real life (e.g. noisy sensors, finite precision, etc.) and techniques tailored for a perfect-knowledge environment can degenerate to give arbitrarily poor results when uncertainty about the agent's state exists [7].

In this paper, we propose to combine reinforcement learning with the modeling of human control strategies in order to address the weaknesses inherent in each approach by itself. The HCS model aids reinforcement learning by intelligently partitioning a

high-dimensional input space into regions that are meaningfully different to the reinforcement learner. Even more significantly, the HCS model can serve to "jump-start" the policy of the reinforcement learner. At the same time, reinforcement learning can take an initially imperfect HCS model and — as we will show — improve it's performance substantially.

This paper is organized as follows. We first describe our experimental setup for recording human control data, a real-time, dynamic graphic driving simulator. We then discuss our approach for modeling human control strategies, and describe some corresponding modeling experiments. Next we incorporate reinforcement learning in our overall learning framework, and demonstrate how reinforcement learning substantially improves the HCS models. We conclude the paper with some observations about the proposed integration of modeling and reinforcement learning, and propose some avenues for future research.

2. Real-time driving simulator

Driving is a prototypical example of human control strategy that offers a rich environment for studying HCS modeling. The task is inherently multi-input, multi-output (MIMO), and includes control outputs which vary both continuously and discontinuously with sensor inputs. Below we describe a driving simulator that we have developed for investigating human control strategies. We choose virtual driving over real driving for a number of reasons: (1) it is safer for the human operator, (2) it allows us better control of our experimental environment, and (3) it allows us to vary the control difficulty of the task without fear of accident or injury.

Figure 1 shows the real-time graphic driving simulator which we have developed as an experimental platform. In the simulator, the human operator has independent control over the steering of the car, the brake and the accelerator, although the simulator does not allow both the gas and brake pedals to be pushed at the same time. The state of the car is described by $\{v_\xi, v_\eta, \omega\}$ [10,11], where 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; the controls are given by,

$$-8000\text{N} \le \alpha \le 4000\text{N}, \tag{1}$$

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

where α is the user-applied longitudinal force on the front tires and δ is the user-applied steering angle.

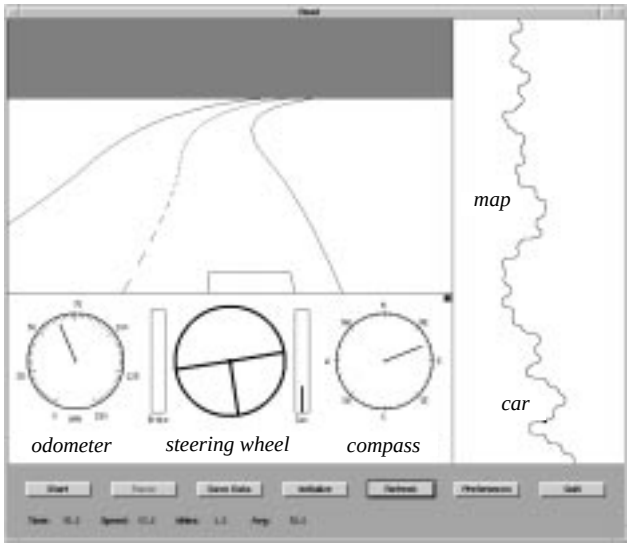

Fig. 1: The driving simulator gives the user a perspective preview of the road ahead. The user has independent controls of the steering, brake, and accelerator (gas).

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

$$\begin{aligned}\alpha(k+1) \in \{&\alpha(k), \min(\alpha(k) + \Delta\alpha_g, 4000),\\ &\max(\alpha(k) - \Delta\alpha_g, 0), -\Delta\alpha_b\}\end{aligned} \tag{3}$$

and for $\alpha(k) < 0$,

$$\begin{aligned}\alpha(k+1) \in \{&\alpha(k), \max(\alpha(k) - \Delta\alpha_b, -8000),\\ &\min(\alpha(k) + \Delta\alpha_b, 0), \Delta\alpha_g\}\end{aligned} \tag{4}$$

For the experiments in this paper, we collect human driving data across randomly generated roads like the 20km one shown in the map of Figure 1. The roads are described by a sequence of (1) straight-line segments and (2) circular arcs. The length of each straight-line segment, as well as the radius of curvature of each arc, lies between 100 and 200 meters. Finally, the visible horizon is set at 100m.

3. Human control strategy modeling

In modeling human control strategies, we want to map sensory inputs to control action outputs. Viewed as a mapping from inputs to outputs, note that the two controls — steering and acceleration — are fundamentally quite different. For given human driving data, steering will tend to vary *continuously* with sensory inputs, while acceleration will tend to vary *discontinuously* with sensory inputs. This is *not* merely an artifact of the constraints in equations (3) and (4), but is caused primarily by the necessary switching between the brake and gas pedals, as is also the case for real driving.

As was demonstrated in [8], continuous learning architectures — whether they be fuzzy logic-based, neural network-based, or memory-based — cannot faithfully reproduce control strategies where discrete events or decisions introduce discontinuities in the input-output mapping. Therefore, we have developed a hybrid continuous/discontinuous modeling framework for handling the two different control types [2,8]. The resulting architecture is illustrated in Figure 2.

3.1 Input-output representation

A necessary condition for successful learning is, of course, that the model be presented with those state and environmental variables upon which the human operator relies. Thus, the inputs

to the HCS model should include (1) current and previous state information $\{v_\xi, v_\eta, \omega\}$, (2) previous output (control) information $\{\delta, \alpha\}$, and (3) a description of the road visible from the current car position. More precisely, the network input vector $\zeta(k)$ at time step k is given by,

$$\begin{aligned}&\{v_\xi(k-n_s), \ldots, v_\xi(k-1), v_\xi(k)\\ &v_\eta(k-n_s), \ldots, v_\eta(k-1), v_\eta(k)\\ &\omega(k-n_s), \ldots, \omega(k-1), \omega(k)\}\end{aligned}, \tag{5}$$

$$\begin{aligned}&\{\delta(k-n_c), \ldots, \delta(k-1), \delta(k)\\ &\alpha(k-n_c), \ldots, \alpha(k-1), \alpha(k)\}\end{aligned}, \tag{6}$$

$$\{x_1(k), x_2(k), \ldots, x_{n_r}(k), y_1(k), y_2(k), \ldots, y_{n_r}(k)\}. \tag{7}$$

where n_s is the length of the state histories and n_c is the length of the previous command histories presented to the model as input. For the road description, we partition the visible view of the road ahead into n_r equivalently spaced, body-relative (x, y) coordinates of the road median, and provide that sequence of coordinates as input to the network. Thus, the total number of inputs to the model at any given time k is $3n_s + 2n_c + 2n_r$. The outputs of the model are $\{\delta(k+1), \alpha(k+1)\}$, the steering and acceleration commands at the next time step, respectively.

3.2 Continuous control

Neural networks are powerful, nonlinear function approximators that have found a number of successful applications in nonlinear control [9]. We believe that neural networks are well suited for learning the complex internal mappings from sensory inputs to continuous control action outputs that are part of a given human's control strategy. Thus, in our modeling framework, the continuous steering control is modeled by a *cascade neural network*, which offers several advantages over more traditional neural network architectures: (1) the network architecture is not fixed prior to learning, but rather adapts as a function of learning [12]; (2) hidden units in the neural network can assume variable activation functions [10]; and (3) the weights in the neural network are trained through the fast-converging node-decoupled extended Kalman filter [13]. The flexibility of these cascade networks is ideal for HCS modeling, since few *a priori* assumptions are made about the underlying structure of the human controller.

3.3 Discontinuous control

Below, we derive a statistical framework for modeling discontinuous control strategies, which model the control strategy not as a deterministic functional mapping, but rather as a probabilistic relationship between inputs and discontinuous outputs. This framework will subsequently be applied to modeling the discontinuous acceleration command α.

3.3.1 General statistical framework

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, $i \in \{1, \ldots, N\}$. Second, assume that we have sets of input vector training examples $\{\zeta_i^j\}$, $j \in \{1, 2, \ldots, n_i\}$, where each ζ_i^j leads to control action A_i at the next time step. Finally, assume that we can train statistical models λ_i, so that

$$\prod_{j=1}^{n_i} P(\lambda_i | \zeta_i^j), \; i \in \{1, \ldots, N\}. \tag{8}$$

is maximized, where $P(\lambda_i | \zeta_i^j)$ denotes the probability of model λ_i given ζ_i^j.

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,

$$p(\zeta^* | A_i) \equiv p(\zeta^* | \lambda_i), \tag{9}$$

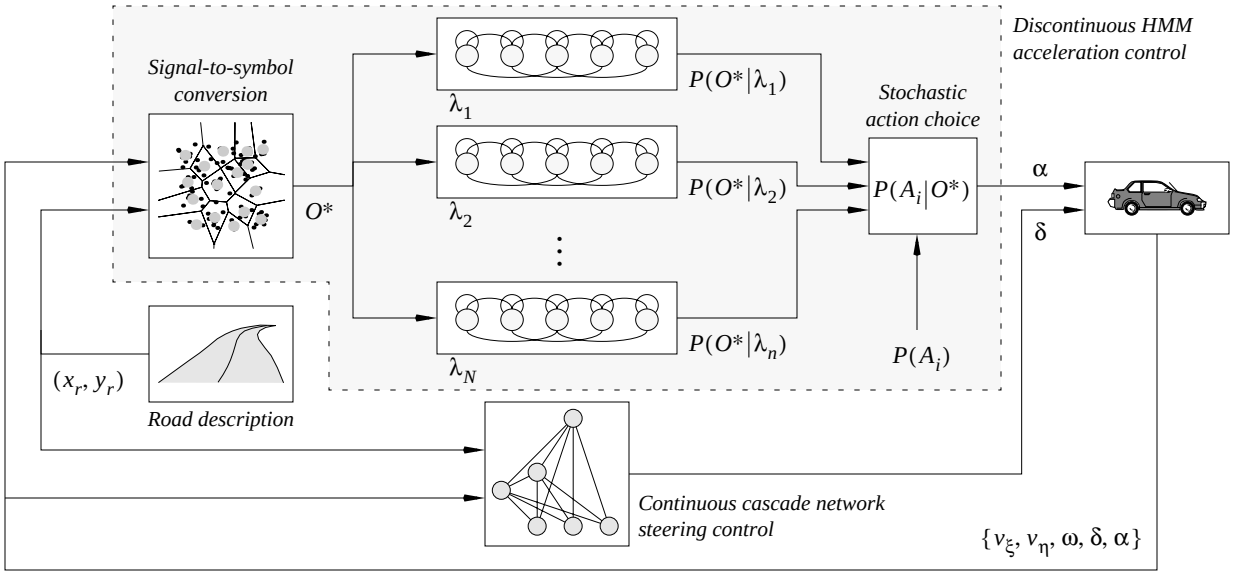

Fig. 2: Overall control structure. Steering is controlled by a cascade neural network, while the discontinuous acceleration command is controlled by the HMM-based controller (shaded box).

where $p(\zeta^*|A_i)$ denotes the likelihood of ζ^* given A_i. By Bayes Rule,

$$P(A_i|\zeta^*) = \frac{p(\zeta^*|A_i)P(A_i)}{p(\zeta^*)}, \quad (10)$$

where,

$$p(\zeta^*) \equiv \sum_{i=1}^{N} p(\zeta^*|A_i)P(A_i), \quad (11)$$

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 define the following stochastic policy for A^*. Let,

$$A^* = A_i \text{ with probability } P(A_i|\zeta^*), \quad (12)$$

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

3.3.2 Statistical model

Hidden Markov Models [14] are powerful, trainable statistical models which have previously been applied in a number areas, including speech recognition [14, 15], modeling open-loop human actions [16], and analyzing similarity between human control strategies [17]. Because of their capacity to model arbitrary statistical distributions, we choose HMMs to be the trainable statistical models λ_i of the previous section.

A discrete Hidden Markov Model[1] consists of a set of n states, interconnected through probabilistic transitions, and is completely defined by $\lambda = \{A, B, \pi\}$, where A is the probabilistic $n \times n$ state transition matrix, B is the $L \times n$ output probability matrix with L discrete output symbols, and π is the n-length initial state probability distribution vector. For an observation sequence O of discrete symbols, we can locally maximize $P(\lambda|O)$ (i.e.probability of model λ given observation sequence O) using the Baum-Welch Expectation-Maximization (EM) algorithm. We can also evaluate $P(O|\lambda)$ through the efficient Forward-Backward algorithm.

Using discrete HMMs, note from Figure 2 that the discontinuous part of the HCS model consists of three distinct steps:

1. Input-space signals ζ^* are first converted to an observation sequence of discrete symbols O^*, in preparation for Hidden Markov Model (HMM) evaluation.
2. The resulting observation sequence O^* is then evaluated on a bank of discrete-output HMMs, each of which represents a possible control action A_i and each of which has previously been trained on corresponding human control data $\{\zeta_i^j\}$.
3. Finally, the HMM evaluation probabilities are combined with prior probabilities for each action A_i according to equations (10) and (12) to stochastically select and execute action A^* corresponding to input observation sequence O^*.

1. Although continuous and semi-continuous HMMs have been developed, discrete HMMs are often preferred in practice because of their relative computational simplicity and reduced sensitivity to initial parameter settings during training [14].

3.3.3 Signal-to-symbol conversion

In order to use discrete-output HMMs, we must first convert the multi-dimensional real-valued input space, to a sequence of discrete symbols. At a minimum, this process involves vector quantizing the input-space vectors $\zeta(k)$ to discrete symbols. We choose the well-known LBG VQ algorithm [18], which iteratively generates vector codebooks of size $L = 2^m$, $m \in \{0, 1, \ldots\}$, and can be stopped at an appropriate level of discretization, as determined by the amount of available data. By optimizing the vector codebook on the human training data, we seek to minimize the amount of distortion introduced by the vector quantization process.

Now, suppose that we want to provide the 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. First, we could set,

$$n_s = n_c = 1 \quad (13)$$

in equations (5) and (6) and then train the models λ_i on observable sequences of length $n_O = m$. Alternatively, we could set,

$$n_s = n_c = m \text{ and } n_O = 1. \quad (14)$$

In the first case, we vector quantize shorter input vectors but provide a longer sequence of observables $n_O > 1$ for HMM training and evaluation. In the second case, we vector quantize the entire input vector into a single observable, and base our action choice solely 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.

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 $\zeta(k)$ minimally includes $2n_r$ road inputs. If we let $n_r = 10$, then for $n_s = n_c = 1$, 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_s, and n_c, 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. For (14) above,

$$P(A_i|\zeta^*) \propto P(O^*|\lambda_i)P(A_i) = b(j)_i P(A_i) \quad (15)$$

where $b(j)_i$ denotes the j th element in the λ_i model's output probability vector B_i. Equation (15) defines a learned stochastic policy,

$$\pi(o, a) = P(a = A_i | o = O_j), \forall i, j, \quad (16)$$

for each possible input observable O_j and action A_i.

3.3.4 Action definitions

As we point out in equations (3) and (4), 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),

$$A_1 : \phi(k+1) = \phi(k) \quad (17)$$

$$A_2 : \phi(k+1) = \min(\phi(k) + \Delta\phi_g, 4000), \quad (18)$$

$$A_3 : \phi(k+1) = \max(\phi(k) - \Delta\phi_g, 0), \quad (19)$$

$$A_4 : \phi(k+1) = -\Delta\phi_b, \quad (20)$$

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

$$A_5 : \phi(k+1) = \phi(k) \quad (21)$$

$$A_6 : \phi(k+1) = \max(\phi(k) - \Delta\phi_b, -8000), \quad (22)$$

$$A_7 : \phi(k+1) = \min(\phi(k) + \Delta\phi_b, 0), \quad (23)$$

$$A_8 : \phi(k+1) = \Delta\phi_g, \quad (24)$$

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. We estimate the priors $P(A_i)$ by the frequency of occurrence of each action A_i in the human control training data. For $\phi(k) \geq 0$,

$$P(A_i) = \begin{cases} n_i / \sum_{k=1}^{4} n_k & i \in \{1, 2, 3, 4\} \\ 0 & i \in \{5, 6, 7, 8\} \end{cases}, \quad (25)$$

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

$$P(A_i) = \begin{cases} 0 & i \in \{1, 2, 3, 4\} \\ n_i / \sum_{k=5}^{8} n_k & i \in \{5, 6, 7, 8\} \end{cases} \quad (26)$$

3.4 Experiment

We ask Larry to drive over two different randomly generated 20km roads ρ_1 and ρ_2, where each run lasts approximately 10 minutes. A part of Larry's second run, for example, is shown in Figure 3(a) below. Larry's driving behavior is representative of other runs recorded by him, as well as other individuals, in that (1) the steering control is reasonably continuous; (2) the acceleration control has significant discontinuities due to rapid switching between the brake and gas pedals; and (3) Larry manages to stay on the road (±5m deviation from the road median) for most of the run, with only a few brief off-road episodes in especially tight turns.

Now, we use Larry's first run (ρ_1) to train a hybrid continuous/discontinuous HCS model (Figure 2), and reserve the run on road ρ_2 for testing the learned model. We use the following suitable input space representation for the model,

$$n_s = n_c = 6, n_r = 10, n_O = 1 \quad (27)$$

as discussed previously, and quantize the input vectors $\zeta(k)$ to $L = 512$ observables. Figure 3(b) plots a typical control trajectory over road ρ_2 for the resulting HCS model. In order to benchmark the performance of the hybrid discontinuous/continuous HCS model, we also train a second HCS model which maps both the steering δ *and* the acceleration α with continuous cascade neural networks, using the input representation of equation (27). Figure 3(c) plots part of the control trajectory over road ρ_2 for this strictly continuous HCS model. Table 1, compares some aggregate statistics for Larry's original data and the two different HCS models.

3.5 Discussion

From Figure 3 and Table 1 we make several observations. First, we note that the continuous HCS model [Figure 3(c)], despite the discontinuous acceleration command, is able to learn *something*; that is, the model keeps the vehicle on the road (except for one high-curvature turn that Larry himself was not able to handle properly). Not only that, but it does so at approximately the same average speed and lateral distance from the road median using a similar steering control strategy as Larry. In some respects, the model's control can even be considered superior to Larry's control. The model only rarely engages the brake, and maintains tighter lateral road position.

If we judge the continuous model on how faithfully it reproduces Larry's acceleration control strategy, however, it rates significantly worse; that is, the model's acceleration control looks nothing like Larry's. It was shown in [8] that Larry's acceleration control is not easily expressed in a continuous functional form, such as a neural network, since the switching discontinuities in the acceleration control essentially require very similar input vectors to be mapped to radically different output vectors.

The hybrid continuous/discontinuous controller [Figure 3(b)] appears to do a much better job in modeling Larry's driving control strategy. In fact, we can quantify the degree of similarity between Larry's control strategy and each of the two models, by computing a stochastic similarity measure σ which we have developed previously [17] for comparing human control strategies. The similarity measure is capable of comparing stochastic, multidimensional trajectories and yields a value between 0 and 1, with

Table 1: Statistical comparison

Road ρ_2	*Larry*	*hybrid model*	*continuous model*
v (mph)	71.9 ± 9.0	70.7 ± 8.1	73.6 ± 2.5
d (m)	-0.72 ± 1.46	-1.38 ± 1.70	-1.00 ± 0.60
δ (rad)	± 0.094	± 0.081	± 0.068
α (N)	2240 ± 2620	1970 ± 2340	1780 ± 760

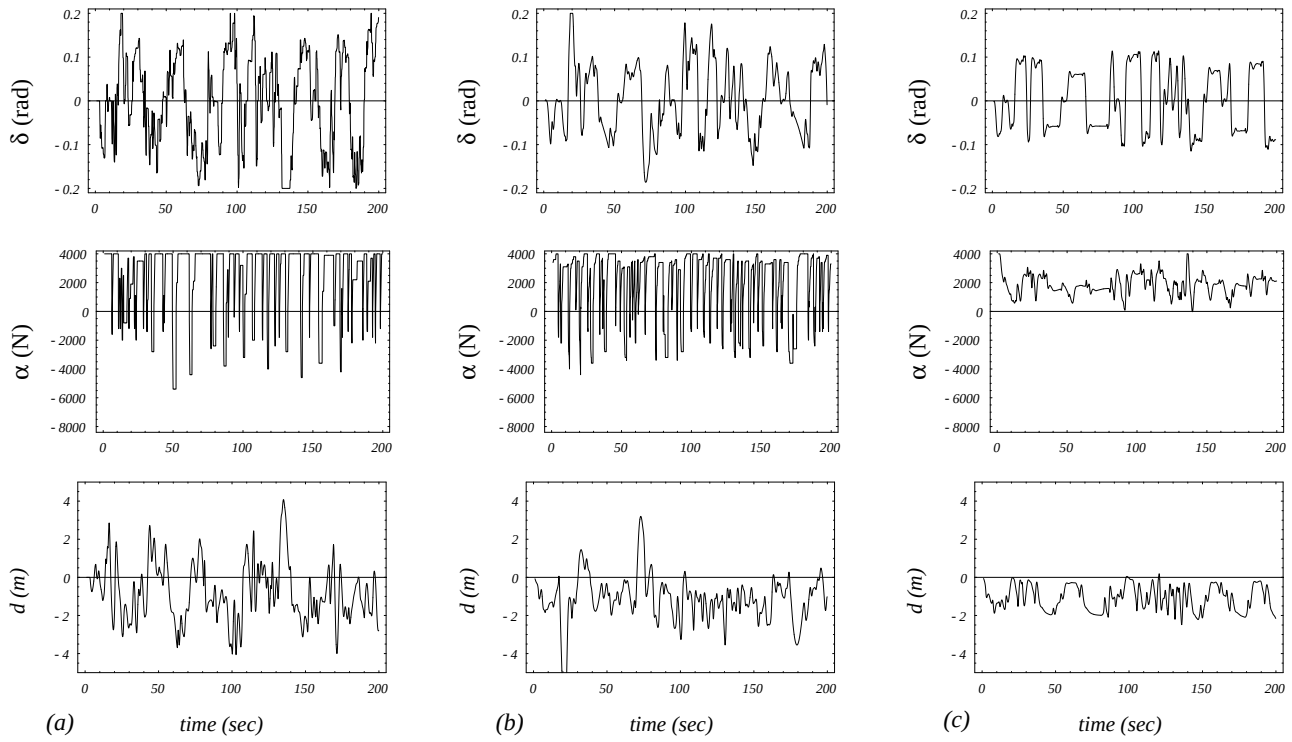

Fig. 3: (a) Part of Larry's steering, acceleration and lateral offset trajectories over time; (b) part of the hybrid continuous/ discontinuous HCS model's steering, acceleration and lateral offset trajectories; and (c) part of the purely continuous HCS model's steering, acceleration and lateral offset trajectories.

larger values indicating greater similarity. For the similarity comparison here, we include all relevant state and control variables $\{v_\xi, v_\eta, \omega, \delta, \alpha\}$, and arrive at the following similarity values:

$$\sigma(\text{Larry, hybrid model}) = 0.57 \tag{28}$$

$$\sigma(\text{Larry, continuous model}) = 0.08 \tag{29}$$

Hence, the hybrid controller is significantly more faithful to Larry's control strategy than the strictly continuous controller.

Unfortunately though, the hybrid controller also tends to be significantly less stable than its continuous counterpart. Note, for example, from Figure 3(c) that the hybrid controller veers off the road at $t = 20$ sec. While the assignment of priors in equations (25) and (26) are the best estimates for $P(A_i)$ given the human control data, they are sometimes problematic when dealing with marginally stable training data. Consider, for example, Figure 4, where we plot a small part of Larry's first run. We observe that Larry'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 (12) is stochastic, it is possible that the stochastic controller will only brake at time $t_s + \tau$, even if the time t_s is modeled as the most likely time for a control switch. Braking at time $t_s + \tau$, however, may be too late for the car to stay in contact with the road.

Ideally, we want to improve the stability of the hybrid HCS model while still maintaining its relatively high fidelity to the human driving data. In the next section, we propose a reinforcement-learning algorithm which does just that.

4. Reinforcement learning stabilization

In previous work [2], we attempted to address the stability problem of the hybrid HCS models by reasoning that the stability of the system (i.e. the simulated car) is directly related to the kinetic energy T,

$$T \propto \sum_k \phi(k), \tag{30}$$

that is pumped into the system, where the expected value of T, $E[T]$, is given by,

$$E[T] \propto \sum_k E[\phi(k)]. \tag{31}$$

Thus, in an attempt to improve the stability margin of the system, we adjusted the model to generate $\phi'(k)$ so that,

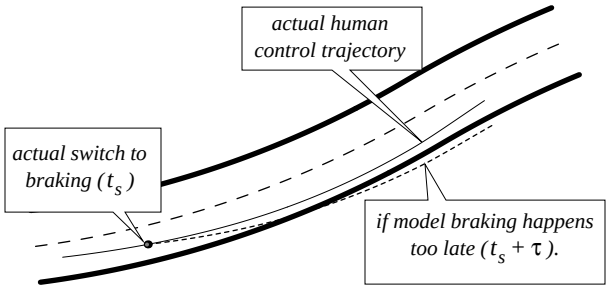

Fig. 4: Instability can result if the hybrid controller switches to braking too late.

$$E[\phi'(k)] < E[\phi(k)] \tag{32}$$

Condition (32) can be realized by increasing the priors for those actions that decrease $E[\phi(k)]$ — namely, A_3 or A_4, by some small amount ε_s, and, to stay within probabilistic constraints, by decreasing the priors A_2 or A_1, respectively, so that,

$$P'(A_3) = P(A_3) + \varepsilon_s \text{ and } P'(A_2) = P(A_2) - \varepsilon_s, \text{ or} \tag{33}$$

$$P'(A_4) = P(A_4) + \varepsilon_s \text{ and } P'(A_1) = P(A_1) - \varepsilon_s, \tag{34}$$

where $\varepsilon_s > 0$ determines the degree to which $E[\phi'(k)]$ is reduced.

The problem with modifications (33) and (34) is two-fold. First, specific values of ε_s need to be experimentally determined for every new HCS model, and second, we cannot be sure that (33) and (34) address all sources of instability. Therefore, we look towards reinforcement learning for a more principled and less *ad hoc* approach to improving model stability.

4.1 POMDPs

Our overall approach for stability improvement proposed below seeks to modify the initial hybrid HCS model through reinforcement learning. We begin by introducing Partially Observable Markov Decision Processes (POMDP) (see [19] for an excellent discussion) as an appropriate modeling framework for our driving domain. In other words, we propose that in the driving domain (and our driving simulator, specifically), there exist meaningful underlying states that are sufficient statistics to determine the next state of the driving agent, and therefore to make optimal decisions, but that the agent is limited to observing a small number of messages from the environment that encode and compress this information. Formally, a POMDP is defined by a tuple,

$$\langle S, A, T, R, O, \Omega \rangle \tag{35}$$

where S is a set of underlying Markovian states (possibly very large or even infinite); A is a finite set of actions an agent can perform; $T(S \times A) \to S$ is a probabilistic transition function that maps the current state and action to the next state; $R(S \times A) \to \Re$ is a function that maps states and actions into real-valued scalar rewards; O is a set of messages or observations; and Ω is a function that maps previous states and actions into probability distributions over observations.

In order to apply the theory of POMDPs to our problem, we must define a goal for the driving agent, as defined by the reward function R. We choose the following simple assignment of rewards. All states that are off the road are assigned a reward of -1, a form of punishment. Further, if the driving agent loses sight of the road (deemed a "catastrophic failure"), we reset the agent to its starting position after applying the -1 penalty for the previous 50 steps. This extended penalty prevents the agent from learning that if it veers off the road, it should aim to reach a catastrophic state so as to quickly return to the good initial position.

Notice that this assignment of rewards does not explicitly impose any conditions on the speed of the vehicle, but rather addresses only stability concerns. This will allow us to monitor what changes the agent imposes first to improve its stability.

Given the reward function, we expect our agent to maximize the average reward it receives over time. To make this criterion well defined, certain technical assumptions on the underlying MDP are required — namely, that under any stationary policy the MDP forms a Markov chain with one irreducible class [4]. We also allow our reinforcement agent to implement stochastic — rather than deterministic — policies for two important reasons: first, to admit the strategies that the HCS modeling architecture requires, and second, to allow strategies that can be shown [7] to be arbitrarily better than deterministic ones. That stochastic policies can achieve better results in the POMDP setting should not be surprising, and may account for the success of stochastic modeling techniques for human control strategies.

4.2 RL algorithm for stabilization

Well-studied algorithms in reinforcement learning (RL), like *Q*-learning [20], suffer from significant weaknesses in our current setting. First, little work has been done with stochastic policies in reinforcement learning. Second, traditional RL techniques are not capable of maintaining fidelity to prior human control data, since they typically modify a policy at every step of policy evaluation. Consequently, a learned policy would very quickly bear little resemblance to the human training data, defeating the main goal of this paper.

Third, many of the current techniques in reinforcement learning have problems dealing with partial observability. Defining an optimal policy can be difficult, as an agent cannot, in general, maximize the value of all observations simultaneously. Moreover, the one-state updates that most algorithms employ do not accurately estimate the value of each observation [7]. Finally, in this domain, there are often many steps between punishments; that is, policies have long mixing times, a further difficulty for reinforcement learning algorithms.

To deal with these difficulties, we propose an algorithm very similar to the one described in [21]. Let (o, a) denote an observation/action pair; let $\pi(o, a)$ denote the current stochastic policy for all (o, a); let $h(o, a)$ denote the current relative value for all (o, a); let $e(o, a)$ denote the current eligibility for all (o, a); and let $v(o, a)$ denote the current number of times that the pair (o, a) has been visited. Also, let $\gamma \in [0, 1)$ denote the temporal-difference learning discount factor; let tot denote the total number of actions performed; and let ρ denote the current average reward. Then the algorithm proceeds as follows:

---

1. **Initialization**:

   At the start of the algorithm, we initialize the following values:

   $$h(o, a) = 0,\ e(o, a) = 0,\ v(o, a) = 0,\ \forall o, a \tag{36}$$

   $$\rho = 0,\ tot = 0 \tag{37}$$

   $$\pi(o, a) = \text{learned HCS model [equation (16)]} \tag{38}$$

   $$o = \text{initial observation} \tag{39}$$

---

2. **Policy evaluation**:

   In evaluating the current policy, we iterate the steps below for max steps. First, we choose an action a stochastically according to the current policy $\pi(o, a)$ and the current observation o, and then execute a. Consequently $v(o, a)$ and tot are incremented,

   $$v(o, a) = v(o, a) + 1, \tag{40}$$

$$tot = tot + 1 \quad (41)$$

Defining,

$$r = \text{reward after execution of } a, \quad (42)$$

$$o' = o \text{ (previous observation), and} \quad (43)$$

$$o = \text{current observation after execution of } a, \quad (44)$$

we compute the differential reward Δ for action a versus the reward we would otherwise have predicted,

$$\Delta = r - \rho + h(o) - h(o', a), \quad (45)$$

where $h(o)$ is the value of the observation (state) o defined as,

$$h(o) \equiv \sum_b h(o, b)\pi(o, b) \quad (46)$$

We then update the value function $h(o, a)$,

$$h(o', a) = h(o', a) + \Delta / v(o', a), \quad (47)$$

$$h(o'', a'') = h(o'', a'') + \frac{\Delta \cdot e(o'', a'')}{v(o', a)},$$

$$\forall (o'', a'') \neq (o', a), \quad (48)$$

and the eligibility function $e(o, a)$,

$$e(o', a) = 1, \quad (49)$$

$$e(o'', a'') = \gamma \cdot e(o'', a''), \ \forall (o'', a'') \neq (o', a). \quad (50)$$

Finally, we update the average reward ρ,

$$\rho = \rho + \Delta / tot \quad (51)$$

---

3. **Policy improvement**:
We are now interested in updating the policy $\pi(o, a)$ for those observations (states) o'' with low values $h(o'')$ (i.e. poor stability). Hence, if,

$$h(o'') < 0, \quad (52)$$

and,

$$\max_{a''} [h(o'', a'')] > h(o'') + \Delta_{min}, \quad (53)$$

then we modify our current policy by,

$$\pi(o'', a'') = (1 - \varepsilon)\pi(o'', a'') + \varepsilon\pi' \quad (54)$$

for some constant Δ_{min} and,

$$\pi' = \begin{cases} 1 & \operatorname{argmax}_{a''} [h(o'', a'')] \\ 0 & \text{otherwise} \end{cases} \quad (55)$$

In other words, the policy for states with poor stability is modified if the value of the best action for those states is significantly larger than the value of those states themselves [equation (53)].

Our algorithm attempts to estimate the average reward ρ of a policy and the relative values $h(o, a)$ [22] of each observation/action pair given the current policy $\pi(o, a)$. The algorithm does this through temporal difference learning [23] to provide multiple-step updates for the relative values $h(o, a)$. The update rule can be thought of as an on-policy version of R-learning [24] with replacing eligibility traces, except that we use stochastic policies, rather than deterministic ones.

After estimating the values of observations, we apply the Policy Improvement Theorem in [21], which states that perturbing our policy toward actions that maximize $h(o, a)$ will improve the average reward as long as the perturbations ε are small. Thus, we carefully choose to perturb the policy only for those observations o'' that stand to have a significant improvement, as indicated by equation (53). We expect that this method of policy improvement will achieve large gains in stability, while incurring only a small loss of fidelity with respect to the original HCS model.

4.3 Experiment

We now apply the algorithm described in the previous section to the hybrid HCS model learned from Larry's control data. The stabilization algorithm is executed on roads that are statistically similar to road ρ_1 with,

$$\varepsilon = 0.05, \ \gamma = 0.96, \ max = 20,000, \ \Delta_{min} = 0.12 \quad (56)$$

The resulting modified model exhibits a dramatic improvement in stability. Where before Larry's model rarely could travel for more than 5km without a catastrophic failure, the modified HCS model completes100km courses without any catastrophic failures. Furthermore, it does so with little drop in average speed and fidelity, as illustrated in Table 2 below.

Table 2: Comparisons

Data set	v_{avg} (mph)	% off-road[a]	similarity σ
Larry	73.1	1.63	0.75[b]
Original model	72.4	**9.02**	0.57
Perturbed model	70.1	**1.23**	0.45

a. Distance from median larger than 5 meters.
b. Self-similarity between Larry's two runs.

4.4 Discussion and future work

Table 2 clearly demonstrates that the RL optimization of the initial HCS model improves the model's stability. It was not possible to achieve similar such improvements using the *ad hoc* stabilization in equations (33) and (34). Let us illustrate schematically why this might be the case. In Figure 5, the policy $\pi(o, a)$ is drawn as a grid, where each box in the grid represents one observation/action pair. (For readability, we drew a grid with only 20 possible observations o.) Boxes in the grid that are shaded are modified after training the HCS model, while boxes that are white represent unmodified states. As part (a) of Figure 5 indicates, the *ad hoc* stabilization procedure of, for example, equation (33) modifies a set number of (o, a) pairs by a fixed amount. In the RL stabilization procedure [Figure 5(b)], however, any (o, a) pair with initial probability greater than zero can potentially be modified by some not predetermined amount. It therefore has significantly greater flexibility in stabilizing the initial HCS model, while still retaining its fidelity to the human control data. Moreover, the RL stabilization procedure is not based on some unprov-

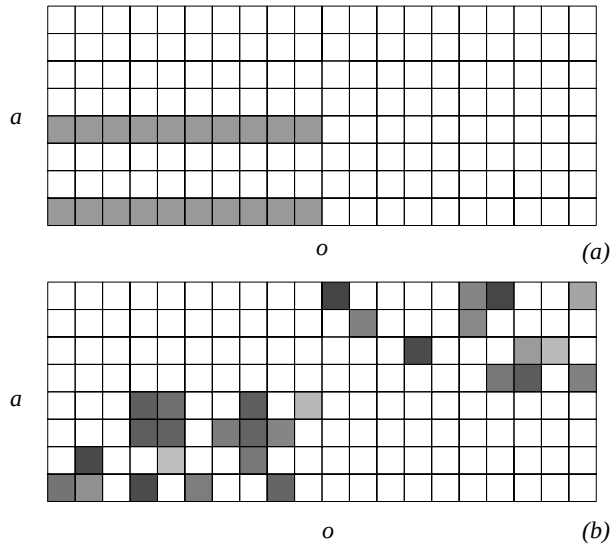

Fig. 5: (a) The *ad hoc* stabilization procedure modifies a fixed set of (o, a) pairs by a fixed amount, while (b) the RL stabilization procedure adjusts (by a variable amount) only those (o, a) that most affect stabilization.

en hypothesis, but rather on active exploration and consequent optimization of the actual system.

We believe that this work opens up a promising direction for future research by combining learning through observation (from humans) with subsequent reinforcement-learning-based optimization. Little work has been done previously to incorporate prior knowledge into reinforcement learning, and this paper offers one potentially successful approach for tackling this problem. Currently, we are exploring the modeling framework of this paper with more varied data (from different humans), varied learning parameters, algorithmic variations and applications in other learning domains. This future work will go a long way towards validating the usefulness and generality of the proposed modeling framework.

References

[1] 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.

[2] M. C. Nechyba, *Learning and validation of human control strategies*, Ph.D. thesis, Carnegie Mellon University, 1998.

[3] J. Song, Y. Xu, Y. Yam and M. C. Nechyba, "Optimization of Human Control Strategies with Simultaneously Perturbed Stochastic Approximation," *Proc. IEEE Int. Conference on Intelligent Robots and Systems*, vol. 2, pp. 983-8, 1998.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New York, 1994.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, 1998.

[6] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-85, 1996.

[7] S. P. Singh, T. Jaakkola and M. Jordan, "Learning Without State Estimation in Partially Observable Markovian Decision Processes," *Proc. Eleventh Int. Conf. on Machine Learning*, 1994.

[8] M. C. Nechyba and Y. Xu, "On Discontinuous Human Control Strategies," *Proc. IEEE Int. Conference on Robotics and Automation*, vol. 3, pp. 2237-43, 1998.

[9] W. T. Miller, R. S. Sutton and P. I. Werbos, eds., *Neural Networks for Control*, MIT Press, Cambridge, 1990.

[10] M. C. Nechyba and Y. Xu, "Learning and Transfer of Real-Time Human Control Strategies," *Journal of Advanced Computational Intelligence*, vol. 1, no. 2, pp. 137-54, 1997.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] X. D. Huang, Y. Ariki and M. A. Jack, *Hidden Markov Models for Speech Recognition*, Edinburgh Univ. Press, 1990.

[16] J. Yang, Y. Xu and C. S. Chen, "Human Action Learning Via Hidden Markov Model," *IEEE Trans. Systems, Man and Cybernetics, Part A*, vol. 27, no. 1, pp. 34-44, 1997.

[17] M. C. Nechyba and Y. Xu, "Stochastic Similarity for Validating Human Control Strategy Models," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 3, pp. 437-51, 1998.

[18] 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.

[19] L. P. Kaelbling, M. L. Littman and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99-134, 1998.

[20] C. J. Watkins, Learning from Delayed Rewards, Ph.D. thesis, King's College, Cambridge, UK, 1989.

[21] T. Jaakkola, S. P. Singh and M. I. Jordan, "Monte-Carlo Reinforcement Learning in Non-Markovian Decision Problems," *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky and T. K. Lee, eds., MIT Press, Cambridge, 1995.

[22] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[23] R. S. Sutton, "Learning to Predict by the Method of Temporal Difference," *Machine Learning*, vol. 3, no. 1, pp. 9-44, 1988.

[24] A. Schwartz, "A Reinforcement Learning Method for Maximizing Undiscounted Rewards," *Proc. Tenth Int. Conf. on Machine Learning*, pp. 298-305, 1993.