

# Simulation Study of Multiple Intelligent Vehicle Control Using Stochastic Learning Automata

Cem Ünsal

Postdoctoral Fellow  
Robotics Institute  
Carnegie Mellon University  
5000 Forbes Avenue  
Pittsburgh, PA 15213-3890  
☎ (412) 268-5594  
📧 (412) 268-5571  
💻 unsal@ri.cmu.edu

Pushkin Kachroo

Assistant Professor  
The Bradley Department of Electrical Engineering  
Virginia Polytechnic Institute and State University  
1700 Kraft Dr., Suite 2000  
Blacksburg, VA 24061-0536  
☎ (540) 231-8340  
💻 pushkin@ctr.vt.edu

John S. Bay

Associate Professor  
The Bradley Department of Electrical Engineering  
Virginia Polytechnic Institute and State University  
Blacksburg, VA 24061-0111  
☎ (540) 231-5114  
💻 bay@vt.edu

**Keywords:** Reinforcement Schemes, AHS/IVHS, Vehicle Path Control.

*Published in **TRANSACTIONS**, the Quarterly Archival Journal of the Society for Computer Simulation International, volume 14, number 4, December 1997.*

## Abstract

An intelligent controller is described for an automated vehicle planning its trajectory based on sensor and communication data received. The intelligent controller is designed using a *stochastic learning automaton*. Using the data received from on-board sensors, two automata (for lateral and longitudinal actions) are capable of learning the best possible actions to avoid collisions. The system has the advantage of being able to work in unmodeled stochastic environments. Computer simulation is a way to test the effectiveness of the learning automata method because the system becomes highly complex because of the presence of a large number of vehicles. Simulations for simultaneous lateral and longitudinal control of a vehicle using this method provide encouraging results. Multiple vehicle simulations are also given, and the resulting complexity is discussed. The analysis of the situations is made possible by the study of the interacting reward-penalty mechanisms in individual vehicles. Simple scenarios consisting of multiple vehicles are defined as collections of discrete states, and each state is treated as a *game* of automata. The definition of the physical environment as a series of discrete state transitions associated with a “stationary automata environment” is the key to this analysis and to the design of the intelligent controller. The aim is to obtain the necessary and sufficient rules for state transitions to reach the goal state.

## 1. Introduction

Growing traffic congestion and the number of traffic casualties are two of the most significant surface transportation problems today. Furthermore, building additional highways is becoming increasingly difficult for both monetary and environmental reasons. One of the possible solutions to the problem is the Automated Highway System (AHS). AHS will *evolve* from today’s roads, and provide a fully automated “hands-off” operation at better levels of performance than today’s

roadways in terms of safety, efficiency, and operator comfort. The 1997 AHS demonstration requested by the Congress will include lateral and longitudinal control, maintenance of position in the roadway traffic flow, and lane changing, all in a collision-free automated driving environment<sup>1</sup>.

Vehicle control is one of the most vital parts of the AHS research. Considering the complexity of an AHS system, it is becoming apparent that the classical control methods are not sufficient to provide a fully automated, collision-free environment. Precursor System Analyses of the AHS indicated that artificial intelligence, especially knowledge-based and learning systems, could make a strong contribution to AHS<sup>2</sup>. Intelligent control is one of the important issues for AHS research, although it is obvious that we might not solve the “whole problem” using a single method. An automated vehicle must be able to deal effectively with a large number of possible traffic situations. What is needed is a system that can handle situations unforeseen by the designers. For this type of capability, several artificial intelligence (AI) paradigms are emerging as candidate solutions. While expert systems and knowledge-based planning<sup>3</sup> are effective, learning methods are capable of discovering new situations and optimal responses using simulated environments<sup>4</sup>.

Automatic vehicle control (AVC), as defined in AHS, will remove the driver as the source of control in the vehicle. This step will be an evolving consequence of the previous technological progress. Therefore, in the early stages of AHS, not all vehicles will be equipped with this technology. “Intelligent” and “non-intelligent” vehicles might have to coexist for some time. The “intelligent” ones will have to plan their actions with respect to the other vehicles in the highway. In this paper, we suggest an intelligent controller for an automated vehicle that plans its trajectory based on sensor and possibly communication data received from other vehicles.

The learning controller can be visualized as a part of the five-layer hierarchical control architecture described by Varaiya<sup>5</sup>. The layers of this architecture, starting at the top, are *network*, *link*, *planning & coordination*, *regulation* and *physical* layers. In this architecture, the planning layer creates a plan that approximates the desired path. The regulation layer controls the vehicle trajectory so that it conforms to this plan. Below the regulation layer, the physical layer provides sensor data and responds to actuator signals.

Our intelligent controller is part of the planning layer, and it is based on an artificial intelligence technique called *learning stochastic automata*. The aim is to design a system that can learn the best possible action(s) based on the data received from on-board sensors and/or roadside-to-vehicle communications. We visualize the intelligent controller of a vehicle as two stochastic automata (one for lateral, the other for longitudinal control) in a nonstationary environment. The system will control the path of the vehicle in the automated highway in the case of communication loss with the higher layer in the hierarchy and/or during the transition from automatic to manual control. It may also be used as the sole system for path planning. A learning automaton system for vehicle control has the advantage of being able to work in unmodeled dynamic environments, unlike adaptive control methods or expert systems that need “detailed” information about the system. It is also possible to model driver and vehicle characteristics as a part of the system.

The first attempt to solve the problem of real-time decision making in an automated highway environment was made by Mourou and Fade<sup>6</sup>. They described a “planning method applicable to agents with perception and decision-making capabilities and the ability to communicate with each other.” This work emphasizes the possibility of achieving a more flexible, fast, and reliable reactive system by multi-agent planning methods and execution monitoring.

Recent research on intelligent vehicles includes an adaptive vehicle module for tactical driving algorithms<sup>4</sup>. The system described by Sukthankar *et al* uses a number of modules whose outputs are combined using a voting scheme. The population-based incremental learning (PBIL) method is used to adjust the large number of parameters defining the behavior of the modules. The learning algorithm for the parameters is a combination of evolutionary optimization and hill climbing, and it is very similar to learning automata reinforcement schemes except for the use of a mutation factor for the action probabilities. The controller we describe here can be visualized as a combination of intelligent modules at the tactical level. However, instead of learning the parameters affecting the firing of an action in repeated runs, learning automata learn which action to fire based on the local sensor information. In other words, the learning is not at the design phase, but at the run phase. Another approach to intelligent vehicle navigation uses a decision-theoretic architecture with probabilistic networks<sup>3</sup>. This work considers the combination of dynamic decision networks with a decision tree, *i.e.*, if-then rules where each predicate is actually a complex set of probability thresholds on specific variables. It is similar to the previous work mentioned above in the sense of firing actions. Similarly, Niehaus and Stengel defined a rule-based navigation system that uses worst-case decision making (WCDM) approach. Again, a stochastic model of the traffic situation based on sensor measurements is assumed<sup>7</sup>.

Here, instead of learning the parameter/behaviors/firing rules for the best actions for achieving a safe path, the controller *learns the action* in real-time. In that sense, the decision to fire an action is never taken at exactly the same time for similar conditions. Furthermore, there are no “prescribed conditions” for actions. The parameters that define the learning process of the stochastic automaton, as well as the sensor parameters defining decision ranges (see Section 3.2), can be learned too using methods similar to ones described above. The idea of defining a “fixed”

structure to be utilized to find the optimal action has its own appeal, since the performance of the system is deterministic in the sense that the best action to be taken for a specific situation is known. However, even a good driver does not follow rules deterministically. In this sense, the learning automata approach is able to capture the dynamics of the driver behavior. A rule-based system, although known to perform well in many situations, has the disadvantage of requiring extensive modifications, even for a minor adjustment. Furthermore, such systems cannot handle unanticipated situations<sup>8</sup>.

Simulation is a way to test the effectiveness of the learning automata method because the system becomes highly complex because of the presence of a large number of vehicles. The overall system is hybrid because the vehicle dynamics are continuous, and the control/decision mechanism is usually a discrete system. The design and analysis of such systems are very difficult, and a rapidly growing research area<sup>9,10,11</sup>. In this paper, however, we have simplified the vehicle dynamics to simple Newtonian differential equations, and assumed discrete speed changes. Our primary aim is to study the feasibility of the learning techniques for intelligent control.

When the single vehicle simulations are extended to multiple vehicles interacting on a highway, complications arise. We will discuss the interaction of multiple automata and the need for more complex behavior rules for the vehicles. Furthermore, we attempt to model simple highway scenarios with discrete state transitions, and with games of learning automata at every discrete state. The learning schemes and the reinforcement rules for the automata guarantee that at every given state, lateral and longitudinal automata will converge to the optimal action(s). Therefore, we may analyze the behavior of the vehicles by studying the state transitions, and may find rules to ‘force’ the vehicles/automata to perform the necessary state transitions to reach their goal state.

In the next section, we will introduce the learning automata and related definitions. Section 3 describes our application of learning automata to intelligent vehicle control. Simulation results for a single automated vehicle are given in Section 4. The treatment of multiple intelligent vehicles as games of interacting automata and the resulting simulations follow in Section 5. A discussion of results, improvements and further research concludes the paper.

## **2. Learning Automata**

Classical control theory requires a fair amount of knowledge of the system to be controlled. The mathematical model is often assumed to be exact, and the inputs are deterministic functions of time. Stochastic control theory, on the other hand, explicitly considers the uncertainties present in the system, but assumes that the statistical characteristics of the uncertainties are known. However, all those assumptions on uncertainties and/or input functions may not be valid or accurate. It is therefore necessary to obtain further knowledge of the system by observing it during operation, since *a priori* assumptions may not be sufficient.

It is possible to view the problem as a problem in learning. Learning is defined as a change in behavior as a result of past experience. A learning system should therefore have the ability to improve its behavior with time. As defined by Narendra and Thathachar, “in a purely mathematical context, the goal of a learning system is the optimization of a functional not known explicitly<sup>12</sup>.” The idea behind designing a learning system is to guarantee robust behavior without the complete knowledge, if any, of the system/environment to be controlled. A crucial advantage of reinforcement learning compared to other learning approaches is that it requires no information about the environment except for the reinforcement signal<sup>13,14</sup>. A reinforcement learning system is slower than other approaches for most applications because every action needs to be tested a

number of times. For a satisfactory performance, either the learning process must be much faster than the environment changes, or the reinforcement learning must be combined with adaptive forward model that anticipates the changes in the environment.

The stochastic automaton attempts a solution of the problem without any *a priori* information on the optimal action. One action is selected at random, the response from the environment is observed, action probabilities are updated based on that response, and the procedure is repeated. A stochastic automaton acting as described to improve its performance is called a *learning automaton*.

This learning process can be examined from two different points of view<sup>15</sup>. It can be thought of simply as an attempt to find the best solution. In case of failure, the attempt is not rewarded (or is penalized). This view of learning is not very attractive. On the other hand, the trial and error method is important in gathering information necessary to achieve optimal action. Choosing non-optimal actions to gain information improves the long-term performance while resulting in temporary penalty responses from the environment. Learning automata is one of the successful examples of such methods.

The first learning automata models were developed in mathematical psychology. Early research in this area is surveyed by Bush and Mosteller<sup>16</sup> and Atkinson *et al.*<sup>17</sup>. Tsetlin<sup>18</sup> introduced deterministic automata operating in random environments as a model of learning. Fu and colleagues were the first researchers to introduce stochastic automata into the control literature<sup>19</sup>. Recent applications of learning automata to real life problems include control of absorption columns<sup>20</sup> and bioreactors<sup>21</sup>. Recent theoretical results on learning algorithms and techniques can be found in recent IEEE Transactions<sup>22, 23</sup> and in Najim-Poznyak<sup>24</sup>. Here, we will

describe the use of learning automata as an intelligent controller for vehicle path planning in unmodeled traffic.

## 2.1. Learning Paradigm

The automaton can perform a finite number of actions in a random environment. When a specific action  $\alpha$  is performed, the environment responds by producing an environment output  $\beta$  that is stochastically related to the action (Figure 1). This response may be favorable or unfavorable (or may define the degree of “acceptability” for the action). The aim is to design an automaton that can determine the best action guided by past actions and responses. An important point is that the knowledge of the nature of the environment is minimal. The environment may be time varying, the automaton may be a part of a hierarchical decision structure but unaware of its role, or the stochastic characteristics of the output of the environment may be caused by the actions of other agents unknown to the automaton.

The input action  $a(n)$  is applied to the environment at time  $n$ . The output  $b(n)$  of the environment is an element of the set  $\underline{b} = \{0,1\}$  in our application. There are several *models* defined by the output set of the environment. Models in which the output can take only one of two values, 0 or 1, are referred to as *P-models*. The output value of 1 corresponds to an “unfavorable” (failure, penalty) response, while output of 0 means the action is “favorable.”

The environment where the automaton “lives” is defined by a triple  $\{\underline{a}, \underline{c}, \underline{b}\}$  where  $\underline{a}$  is the action set,  $\underline{b}$  represents a (binary) output set, and  $\underline{c}$  is a set of penalty probabilities (or probabilities of receiving a penalty from the environment for an action). Each element  $c_i$  corresponds to one action  $a_i$  of the action set  $\underline{a}$ . The response of the environment is considered to be a random variable. If the probability of receiving a penalty for a given action is constant, the environment is called a *stationary environment*; otherwise, it is *nonstationary*. The need for

learning and adaptation in systems is mainly due to the fact that the environment changes with time. Performance improvement can only be a result of a learning scheme that has sufficient flexibility to track the better actions. The aim in these cases is not to evolve to a single action that is optimal, but to choose the actions to minimize the expected penalty. For our application to intelligent vehicle control, the (automata) environment is nonstationary since the physical environment around the vehicle changes.

The main concept behind the learning automaton model is the concept of a probability vector defined (for  $P$ -model environment) as  $p(n) = \{p_i(n) \in \{0,1\} \mid p_i(n) = \Pr[a(n) = a_i]\}$  where  $a_i$  is one of the possible actions. We consider a stochastic system in which the action probabilities are updated at every stage  $n$  using a *reinforcement scheme*. The updating of the probability vector with this reinforcement scheme provides the learning behavior of the automata.

## 2.2. Reinforcement Schemes

A learning automaton generates a sequence of actions on the basis of its interaction with the environment. If the automaton is “learning” in the process, its performance must be superior to an automaton for which the action probabilities are equal. “The quantitative basis for assessing the learning behavior is quite complex, even in the simplest  $P$ -model and stationary random environments<sup>14</sup>.” Based on the average penalty to the automaton, several definitions of behavior, such as *expediency*, *optimality*, and *absolute expediency*, are given in the literature. Reinforcement schemes are categorized based on the behavior type they provide, and the linearity of the reinforcement algorithm. In general terms, a reinforcement scheme can be represented as:

$$p(n+1) = T[p(n), a(n), b(n)]$$

where  $T$  is a mapping,  $a$  is the action, and  $b$  is the input from the environment. If  $p(n+1)$  is a linear function of  $p(n)$ , the reinforcement scheme is said to be linear; otherwise it is termed

nonlinear. Early studies of reinforcement schemes were centered mostly on linear schemes for reasons of analytical simplicity. In spite of efforts of many researchers, the general algorithm that ensures optimality has not been found<sup>25</sup>. Optimality implies that action  $\alpha_m$  associated with the minimum penalty probability  $c_m$  is chosen asymptotically with probability one. Since it is not possible to achieve optimality in every given situation, a suboptimal behavior is defined, where the asymptotic behavior of the automata is sufficiently close to optimal case.

A few attempts were made to study nonlinear schemes<sup>26,27</sup>. Generalization of such schemes to the multi-action case was not straightforward. Later, researchers started looking for the conditions on the updating functions that ensure a desired behavior. This approach led to the concept of absolute expediency. An automaton is said to be absolutely expedient if the expected value of the average penalty at one iteration step is less than the previous step *for all steps*. Absolutely expedient learning schemes are presently the only class of schemes for which necessary and sufficient conditions of design are available<sup>26, 27</sup>. The algorithms we generally use in our simulations are (a) linear reward-penalty scheme with unequal learning parameters  $L_{R-P}^{\neq}$  and (b) a nonlinear absolutely expedient reinforcement scheme NL<sub>H</sub>. Detailed descriptions of these schemes will be given in Section 3.

### **2.3. Automata and Environment**

The learning automaton may also send its action to multiple environments at the same time. In that case, the actions of an automaton result in a vector of feedback values from the environment. Then, the automaton has to “find” an optimal action that “satisfies” all the environments (in other words, all the “teachers”). In a multi-teacher environment, the automaton is connected to  $N$  separate teachers. The action set of the automaton is of course the same for all teacher/environments. Baba discussed the problem of a variable-structure automaton operating in

multi-teacher (stationary and nonstationary) environments<sup>27</sup>. Conditions for absolute expediency are given in his work. (Our initial nonlinear reinforcement scheme was an adapted version of the algorithms described by Baba<sup>27</sup>.)

Some difficulties arise while formulating a mathematical model of the learning automaton in a multi-teacher environment. Since there are multiple responses from the environment, the question of how to interpret the output vector  $b(n)$  is important: Are the outputs of different teachers to be summed after normalization? Can we introduce weight factors associated with specific teachers? If so, how? The elements of the output vector must be combined in some fashion to form the input to the automaton. A straightforward method is to define the input to the automaton as a weighted sum of all outputs where weight factors attached to each teacher output must be chosen in order to guarantee that the combined response is in the defined range (for the S-model where the environment response  $\beta(n) \in [0,1]$ ). In this application, since the environment is P-model, teacher responses are combined using an OR gate, which forces the system to satisfy all the teachers simultaneously. However, due to safety and other reasons described later, the finalized function for the combined response includes conditions in which one teacher response inhibits the other as we discuss later in Section 3.2.

### **3. Learning Automata as an Intelligent Controller**

Our approach to the problem of vehicle control makes use of learning automata techniques described in the previous section. We model the path controller of an intelligent car as a pair of automata in a nonstationary environment. The aim is to design an automata system that can learn the best possible action(s) based on the data received from on-board sensors. Vehicle-to-vehicle and roadside-to-vehicle communications will be future extensions. The significance of this system

is that the learning automata method we define will be useful as a backup system (or the only system in a homogeneous traffic network in the near future) in controlling the path of a vehicle in the case of communication loss with the higher layer in the hierarchy, as well as during the transition from automated control to manual vehicle operation.

### 3.1. The model

Our basic model of the planning/coordination layer for lane changing and speed control of a vehicle is shown in Figure 2. The automata that constitute the decision structure have three actions each: *stay-in-lane* (the “idle” action/state), *shift-to-right* (lane), *shift-to-left* for lateral control (SiL, SR and SL), and *accelerate*, *decelerate* and *same-speed* for longitudinal control (ACC, DEC, SM). Initially, we assume that there are four types of sensors (*front sensor*, *right sensor*, *left sensor* and *speed sensor*) that provide data. Each sensor block-decision module pair is actually a “teacher” in a nonstationary environment (vehicle-to-vehicle and roadside-to-vehicle communications, as well as a feedback connection from the regulation layer, are possible additions as additional “teachers.” In Section 5, we will give examples of such information sources). The response of the environment is then a combination of the outputs of all four teachers, as discussed in section 3.2. The mapping  $F$  from sensor block outputs to the input  $b$  of the automaton can be a binary function (for a P-model environment), a linear combination of five teacher outputs, or a more complex function — as is the case in this application. An “ideal” automaton model would use a linear combination of teacher outputs with adjustable weight factors (*e.g.*, S-model environment). The function  $F$  is explained in detail in the next section where we discuss the simple sensor models.

Using the method and the reinforcement scheme described in Section 2.1 and 2.2, the learning process is given as follows:

**Step 1.** Choose an action,  $a(n) = a_i$  based on the action probability vector  $p(n)$ .

**Step 2.** Read environment responses  $b_i(n)$  from the sensor modules.

**Step 3.** Compute the combined environment responses  $f_{LAT}$  and  $f_{LNG}$  using the mapping function  $F$ .

**Step 4.** Update action probabilities  $p(n+1) = r(p(n))$  where  $r(\cdot)$  is the predefined reinforcement scheme.

**Step 5.** Go to step 1.

It is important to differentiate between the “automaton environment” and the “physical environment.” The action  $a$  of the automaton is a signal to the regulation layer that defines the current choice of action. It is the regulation layer’s responsibility to interpret this signal. When an action is carried out, it affects the physical environment. The teachers/sensors in turn sense the changes in the environment, and the feedback loop is closed with the signal  $b$ .

The discussion of nonstationary environments is based on the changing penalty probabilities of actions. In this application, the action probabilities in the learning automaton environment are functions of the status of the physical environment (*e.g.*, a vehicle in front will result in a penalty response from front sensor/teacher if the chosen action is *stay in lane* or *accelerate*). The realization of a deterministic mathematical model of this physical environment would be extremely difficult.

### **3.2 Sensors and Teacher Blocks**

The four teacher blocks listed above are actually simple decision modules that calculate the penalty response associated with the corresponding sensor, based on the chosen action. Tables 1 and 2 describe the output of the decision blocks for front and side sensors. As seen in Table 1, a side sensor block returns a penalty response (‘1’) to the lateral automaton if there is a vehicle in

the right/left lane (the range of this side sensor is defined by two parameters  $sr_l$  and  $sr_r$ ; see Figure 3) and the current action is “shifting to the right/left lane”. A reward is returned for other lateral and all longitudinal actions.

We assume that the front sensor is capable of providing the headway distance, and that we can calculate the rate of change of the headway distance to the vehicle in front by comparing two consecutive sampling values. If the sensor ‘sees’ a vehicle at a very close distance ( $< d_1$ ), or the vehicle in front is close and approaching, a penalty response is sent to actions *stay-in-lane* (SiL), *accelerate* (ACC), and *same-speed* (SM). All other actions (shifting lane or decelerating) may serve to avoid a collision, and therefore, would be “encouraged.” If the vehicle in front is not too close (*i.e.*, the distance to the vehicle in front is greater than  $d_1$ , but less than  $d_2$ ) and is not approaching, the response from the front sensor is favorable, except for the action ACC. The same response is valid for headway distances greater than  $d_2$ . All the sensor limits as well as the sensor range  $fsr$  are predefined.

For example, consider the situation shown in Figure 3 where the controlled vehicle’s left and front sensors detect vehicles, and the vehicle in front is slower than the controlled vehicle. If the current automata actions are *shift-to-left* (SL) and *accelerate* (ACC), the lateral action SL will receive a penalty from left sensor (see Table 1), and the longitudinal action ACC will receive a penalty from the front sensor because of the approaching vehicle (See 2<sup>nd</sup> column of Table 2).

Table 3 gives the penalty definitions for the speed decision module. The value  $dev$  is defined as the difference between current speed and the desired speed. A penalty response is received only by longitudinal actions if the absolute difference between the actual and desired speeds is larger than a predefined value  $ds$ . Lateral actions are not affected by the speed sensor block.

The values of the limits  $d_1$ ,  $d_2$  and  $d_s$  define the capability of the sensors as well as the “behavior” of the vehicle, *i.e.*, the sensitivity to the headway distance and to the speed fluctuations. The fifth block in Figure 2 that represents the evaluation of vehicle-to-vehicle and/or roadside-to-vehicle data, is not fully defined. The importance of this block is that the global characteristic of the data received from the roadside and other vehicles should be used to solve some of the problems in the lane changing maneuvers in order to obtain a more optimal path. For example, in the case shown in Figure 3, the action SL will receive a penalty response, although it is one of the two actions to avoid a collision (the other one is *Decelerate*). Using the present algorithm, the vehicle decelerates in a situation like this, and the speed of the vehicle may drop below the desired lower speed limit. The mapping  $F$  forces it to avoid collision. As seen in Figure 4, a reward for action DEC inhibits penalty for action SiL (indicated by ‘\*’ in Table 3) .The data received from a higher level in the control structure can be used to suppress the penalty response of the left sensor block (provided that the result is not dangerous). We expect our designed automata and multi-teacher environment to guide the vehicle without collision using the learning algorithm described in the next section.

The mapping  $F$  shown in Figure 2 is described in detail in Figure 4. The outputs of this mapping function are the feedback signals to the lateral and longitudinal automata ( $f_{LAT}$ , and  $f_{LNG}$ ). There are two conditional rules added to the OR-ed feedback signals for lateral and longitudinal automata. The first rule checks if the output from the headway module conflicts with the sensor module for the longitudinal action *decelerate*. When the headway is relatively small and decreasing, the action DEC is the optimal action no matter what the speed module returns. Therefore, a reward response from the headway module must inhibit a penalty response from the speed module for deceleration. The second rule is designed to discourage unnecessary lane

changes. Even if the front sensor indicates an approaching vehicle, the penalty response is inhibited by a reward response for longitudinal actions. For example, if the vehicle is decreasing its speed to avoid a collision with the vehicle in front, lateral action *stay-in-lane* does not receive a penalty from the headway module provided that the headway distance is not less than  $d_l$  (See Table 2).

### 3.3. Learning algorithms and the Environment Feedback

Besides the standard linear reinforcement schemes<sup>14</sup>, we have used two new reinforcement schemes for this application. The first one is a linear reward-penalty scheme with unequal learning parameters<sup>28</sup>. The second scheme is a nonlinear learning algorithm and is similar to the one given by Baba<sup>27</sup> except an additional term that is needed for fast convergence as we explain below<sup>29</sup>. Here, we will introduce these schemes, and describe their behaviors under certain conditions.

Suppose the action of the automaton chooses the  $i^{th}$  action  $a(n)$  at time step  $n$  with probability  $p_i(n)$ , and the environment response is  $b(n) = f_{LNG}(n)$  or  $b(n) = f_{LAT}(n)$ . Then, the following general linear updating scheme is used to calculate the new values of the action probabilities:

$$\begin{cases} p_i(n+1) = p_i(n) + a \cdot [1 - p_i(n)] \\ p_j(n+1) = (1-a) \cdot p_j(n) \quad \text{for all } j \neq i \end{cases} \quad \text{if } b(n) = 0$$

$$\begin{cases} p_i(n+1) = (1-b) \cdot p_i(n) \\ p_j(n+1) = \frac{b}{r-1} + (1-b) \cdot p_j(n) \quad \text{for all } j \neq i \end{cases} \quad \text{if } b(n) = 1$$
(1)

where  $a$  and  $b$  are real-valued learning parameters associated with reward and penalty respectively. These parameters are defined as positive to keep their reward/penalty characteristics,

and less than 1 to guarantee the condition  $\sum_{i=1}^r p_i(n) = 1$  for all  $n$ .

The algorithm we use is called the linear reward-penalty scheme with unequal parameters,  $L_{R-P}^\#$ , where  $a$  and  $b$  are different. In the case where  $a > b$  and there is only one action with zero probability of penalty  $c_{opt} = 0$ , this algorithm is guaranteed to reach the pure optimal strategy, *i.e.*, the probability of the “optimal” action reaches 1. An automaton using such a learning scheme is said to be expedient and optimal<sup>28</sup>.

For an automaton with action  $a(n)$  at time  $n$ , the nonlinear learning scheme is may be defined as:

$$\begin{cases} p_i(n+1) = p_i(n) - \beta(n) \cdot k \cdot \theta \cdot H(n) \cdot [1 - p_i(n)] + [1 - \beta(n)] \cdot \theta \cdot [1 - p_i(n)] \\ p_j(n+1) = p_j(n) + \beta(n) \cdot k \cdot \theta \cdot H(n) \cdot p_j(n) - [1 - \beta(n)] \cdot \theta \cdot p_j(n) \quad \text{for all } j \neq i \end{cases} \quad (2)$$

where learning parameters  $k$  and  $q$  are real values and satisfy:

$$0 < q < 1 \quad 0 < kq < 1 \quad (3)$$

the feedback  $b(n)$  is the output of the function defined in Fig. 4, and the function  $H$  is defined as:

$$H(n) \equiv \min \left\{ 1; \max \left[ \frac{p_i(n)}{kq(1 - p_i(n))} - \varepsilon; 0 \right] \right\} \quad (4)$$

Parameter  $\varepsilon$  is an arbitrarily small positive real number. Also note that the function  $H$  includes  $p_i$  which is the action probability corresponding to the current action. This reinforcement scheme is shown to satisfy all necessary and sufficient conditions for absolute expediency<sup>29</sup> for a stationary environment. An automaton using this scheme is guaranteed to “do better” at every time step than at the previous step. The choice of the function  $H$  is due to the convergence issue as well as to the conditions on absolute expediency. This nonlinear algorithm defined here is found to converge to the “optimal” action faster than previously defined nonlinear schemes, especially when the actions receiving penalty from the environment have high probability values<sup>30</sup>.

The physical environment in this application (the highway) is of course changing with time. However, since the update rate for action probabilities is high (25 Hz or more), the automata environment can be considered as stationary. For a physical environment that does not change quickly, the automata would be capable of finding the necessary action by a learning process that does better at each time step. Here, the reinforcement scheme updates the probability values so that the expected value of the total penalty received from the environment decrease at each iteration.

To smooth the system output, the regulation layer carries out an action if it is recommended  $m$  times in the last  $k$  iterations where  $k$  corresponds to the total number of iterations in one second. Whenever  $m$  locations of this vector/buffer are filled with the same action, the action is fired. When an action is carried out (*e.g.*, shifting lane), the action probabilities in the controlling automaton are re-initialized (if linear scheme is used) to  $1/r$  where  $r$  is the total number of actions for an automaton. The value  $m$  corresponds to processing speed. Therefore, the regulation layer executes an action if it is sent consecutively over a period of 1sec.

More continuous regulation layer modeling is of course possible, but not considered here. A more realistic model would include delays, because the regulation layer cannot perform actions requested by the planning layer instantaneously. Our aim here is to introduce an intelligent vehicle control model based on learning. For lateral actions, we have assumed a lane transition interval which guarantees a lateral acceleration less than 0.5g. For longitudinal actions, the fact that the controlled variable is the headway distance, but not the speed enables us to employ this simple regulation layer model.

A minimum processing speed of 25, and a maximum of 200 iterations per second are assumed. This is related only to computation; the sensor data feeds have a different (and constant)

rate. This value is much slower than the limit of 200 Hz that is considered in current AHS research<sup>31</sup>.

#### **4. Simulation Results**

The simulations were written and run on Matlab<sup>®</sup> (Figure 5). The program can be run in Matlab version 4.2 or higher. Graphic user interfaces are designed on a Sun<sup>®</sup> workstation running Solaris 2.4<sup>®</sup> and OpenWindows<sup>®</sup>, but they can be displayed on any platform running Matlab with minor cosmetic changes in the GUI subroutines.

During a simulation run, sensor outputs are evaluated and action decisions are made at each iteration step. Then, those actions are carried out for each automated vehicle on the highway. Sensor modules and flag structures are implemented as subroutines that take the vehicle index and return the corresponding output. If the vehicle is not automated, no output is generated; the vehicle position is updated according to predefined speed and lane parameters. Learning, decision (planning), and regulation subroutines work similarly. When evaluations are complete, the highway environment is updated. The main program repeats the loop until the final time step is reached or the program is interrupted by the user.

The learning parameters, learning algorithm, processing speed and memory vector sizes cannot be changed during the simulation run. Other parameters such as current vehicle speed, desired speed, current vehicle lane, desired lane, permitted speed variations, as well as other display parameters, can be changed during the simulation run. The command line interface can also be used to change parameters before starting the run; it also displays several parameters and/or actions during the simulation (Figure 5). The control structure we introduce here is

incorporated in the Dynamic Visual Micro/Macroscopic Traffic Simulation (DYNAVIMTS) package as part of the planning layer simulator<sup>32</sup>.

In the following sections, we will first give results of sample runs for the model described in the previous section. Several additions to the control model based on the results of our initial simulation runs are then described. The vehicles travel left to right on a 500m straight highway segment with annular topology: a vehicle leaving the segment from the right enters it from the left. The number of lanes and the length of the highway is adjustable. The results are given by plotting the relative positions of all vehicles: all vehicles except one are plotted in their relative positions with respect to the vehicle that is assumed to be “followed by the observer’s eye” (Figures 6, 9 and 10). In some cases, snapshots of the simulations are given. In these plots, the traffic moves left-to-right, and the vehicles in question are indicated with different gray tones (Figures 11, 13, 15 and 17).

#### **4.1 Single Autonomous Vehicle in Mixed Traffic**

Our first simulation shows the behavior of the single automated vehicle in mixed traffic. As seen in Figure 6, the vehicle (third vehicle from left) travels with 28 other vehicles cruising at a constant speed of 80 kmh. The desired speed for this automated vehicle is 86kmh. Sensor limits  $d_1, d_2, fsr, sr_1, sr_2$  are 10, 20, 30, 10, and 10 meters respectively. The linear learning scheme  $L_{R-P}^\#$  with  $a = 0.15$  and  $b = 0.10$  is used. The length of the memory vector is set to 1sec for longitudinal, and 0.5sec for lateral actions. All other vehicles are traveling at constant speed, and cannot change lanes.

The vehicle is initially in lane 3 (third from the bottom), traveling at 86kmh. It eventually decreases its speed and shifts to lane 2 (left) due to the presence of a slow-moving vehicle ( $t = 14$ sec). It then increases its speed to 86kmh until it ‘senses’ another slow-moving vehicle in front

(the fluctuations in speed are due to the fact that all longitudinal actions receive reward when the vehicle speed is in the desired range). Due to the penalty response from the headway module, vehicle speed decreases again, although not fast enough. Thus, the lateral action SiL receives a penalty from the front sensor and the vehicle shifts lanes again at approximately  $t = 72\text{sec}$  (Figure 7). At this point, both SL and SR are possible lateral actions, since the vehicle uses only local data from its sensors. The probabilities of both actions reach 0.5. The regulation layer chooses the single action that occupies the most locations in the memory vector. As seen in Figure 6, the choice is wrong because the vehicle eventually gets stuck in the pocket in lane 1. The vehicle is able to slow down in order to keep a headway distance between  $d_2$  and  $f_{sr}$ . However, the penalty response from the speed module for action *decelerate* makes it impossible for the vehicle to slow down and “escape” from the pocket.

Since all other vehicles are traveling at constant speeds, and the inter-vehicle communications are not established, the only solution to this problem is to slow down and shift left twice to lane 3. For this, it is necessary to devise a system to force the vehicle to slow down and shift to the desired lane. A new module for lane sensing is defined (Figure 8). Assuming that the vehicle is capable of sensing its current lane and receiving/computing the desired lane, the lane module keeps track of the time to reach the desired lateral position. If the time interval in the undesirable lane is more than a predefined value, a *flag* is set. This flag is used by the speed sensor to adjust the desired speed value. Currently, we simply decrease the desired speed by a predefined amount (which can be a function of the speed of the vehicle in front and/or the automated vehicle). This simple addition shows the difficulty of obtaining “optimal behaviors” for vehicles using only local information. Higher level hierarchy intervention is necessary for optimal path decisions.

## 4.2 Multiple Autonomous Vehicles

Our second example uses the new lane sensing module defined above as well as another module we call the “pinch module.” Just like a driver who checks his rearview mirrors and looks to his side before shifting lanes, it is imperative for an automated vehicle to make sure that the next lane is not “claimed” by another vehicle. This problem occurs when two vehicles one lane apart shift to the same spot in the lane between them. Besides the side sensors returning local data, vehicles “signaling” to shift lanes must be checked in order to make sure that a pinch condition does not occur. In the simulations we use the memory vector to check for other vehicles’ intention to shift lanes. If the number of memory locations containing SR or SL is more than half the size of the vector for that vehicle, it is assumed that the vehicle is likely to shift lanes. This corresponds to a signaling vehicle, vehicle-to-vehicle communications indicating intended actions, or a roadside-to-vehicle communication relaying the positions of the vehicles (Figure 8).

Figure 9 shows the relative trajectories of 24 vehicles in a 4-lane highway (the first vehicle on the left is taken as the reference, and all other vehicle positions are plotted relative to that in time). Initially, all vehicles are in the third or fourth lane (gray color). The desired lane for all vehicles is the second lane. In the first 6 seconds, all vehicles move to the second lane. However, although the desired speed for all vehicles is the same, the speeds are initially distributed randomly around the desired value, and they change during the simulation (as seen in Figures 9 and 10). After  $t = 6\text{sec}$ , some of the vehicles (numbers 15 and 18; see Figures 10 and 11) change lanes to avoid a collision. As seen in Figure 10, once they shift lanes, they are unable to return to the desired lane since the gap between vehicles decreases. Four seconds later, the lane modules set the flags for the speed modules to adjust the desired speed values. Both vehicles decrease their speeds (Figure 12), and then are able to find a gap in the second lane and shift to that lane.

Immediately after the shift, their speeds increase (Figure 12). Snapshots of this simulation are given in Figure 11. Vehicles 15 and 18 are indicated with darker gray tones.

Once all the vehicles are in the desired lane, they adjust their speeds to maintain the headway between the desired values ( $d_2$  and  $fsr$ ). Figure 13 shows the snapshots of such a scenario. Fifteen vehicles with random headway and speed values are able to form a “platoon” with desired speed and spacing. As seen in Figure 14, the speeds and the headway distances reach the desired values after approximately 1min. The convergence to the desired values depends on the update rates and the learning speed of the automata as well as the physical constraints. The desired speed is 83kmh, predefined headway sensor limits are  $d_1 = 11m$ ,  $d_2 = 15m$ , and  $fsr = 20m$  (shown in Figure 14).

Note that only current lane detection is needed for the lane changing behavior described in this section. Simply adjusting the speed may be sufficient for a vehicle to shift to the desired lane without any higher level hierarchy intervention. More examples of multiple autonomous vehicles situations can be found in a recent work by the first author<sup>30</sup>.

There are many parameters affecting the behavior of an automated vehicle. It is obvious that the learning would be faster for larger learning parameters and faster processing speeds. Also, a short memory vector results in faster action firing, thus creating a more agile vehicle. The length of the memory vector can be taken down to 1 for longitudinal actions provided that the speed changes are continuous. For lateral actions, there is a minimum for the length of the memory vector since continuous lane changes are not desired. Vehicle separations in a platoon are found to be more uniform for larger memory vectors. However, there is a tradeoff between uniform inter-platoon distances and the stability of the responses. More uniform vehicle separation is the result of quickly changing vehicle speeds.

Sensor ranges are also important factors in vehicle behavior. By increasing the region around the front sensor range  $d_2$ , it is possible to obtain steady-state speeds in a very short amount of time. Sensor ranges, especially those related to headway measurement, must be carefully adjusted to avoid collisions and oscillatory responses. Detailed simulation results showing effects of the learning parameters, sensor ranges, and memory vectors can be found in the above mentioned work<sup>30</sup>. Adjusting sensor parameters can be thought of as another level of learning where the methods described by Sukthankar *et al* can also be used<sup>4</sup>.

## **5. Multi-Vehicle Path Coordination using Automata Games**

In previous sections, we introduced an intelligent control method for autonomous vehicle navigation and path planning. The decision system uses mostly local information, and as a result, the actions learned by the controller are not globally optimal; the vehicles can survive, but may not be able to reach some of their goals. To overcome this problem, we visualize the interaction between vehicles as sequences of games played between pairs of automata. Every game corresponds to a “state” of the physical environment as described below. By evaluating these games, it is possible to design new decision rules, and analyze the interactions by predicting the behavior and position of each vehicle. The additional modules given in Figure 8 are actually the results of the approach describe here.

In order to find the necessary modules and decision factors for autonomous vehicles, we define a “game” of automata. We know that for a given physical situation, the longitudinal and lateral automata will converge to the optimal action provided that the update rate is fast enough. Consider the situations shown in Figure 15. Defining a game matrix for this “2-person non-zero sum” game, we can guarantee that both automata in both vehicles will converge to an optimal

action pair. It has been shown that for two automata using the linear reward-penalty scheme with  $a > b$ , the system will asymptotically reach a unique equilibrium for a zero-sum game<sup>33</sup>. Furthermore, it is possible to extend the same result to  $N$ -person non-zero sum games<sup>14</sup>.

Using the modules described in previous sections, it is possible to define a 2-person non-zero sum game matrix for all situations in Figure 15. For example, for situation A2, the game matrix is given in Table 4. All payoffs are binary, and this constitutes a non-zero sum, unidentical payoff game. It is assumed that the vehicles are already cruising at the desired speed (steady-state). Each square in the matrix is separated into lower and upper triangles, to show the environment responses for vehicle 2 and 1 respectively. A dark triangle indicates a penalty response while a white triangle corresponds to reward. From this game matrix, we can deduce that only idle actions (SM and SiL) for both vehicles, and SR for vehicle 2 are permitted, *i.e.*, white rectangles indicate equilibrium points in the game matrix. (Note that it is possible to combine the four squares in the middle of the matrix into a single *idle* action.) From this matrix, we find that vehicle 2 may shift right or keep its present lane while both vehicles keep their speeds. These actions correspond to transition A2→A1 and no transition respectively.

Consider a scenario where vehicle 1 is in the leftmost lane (lane 3), and vehicle 2 is in the rightmost lane (lane 1). Then assume that the first vehicle needs to shift to lane 3 while the other attempts to shift the lane 1. If they are cruising at the same speed and have the same lateral position, the “game” is at state A1. This and all other possible states are given in Figure 15 as well as the state transition diagram. Depending on the first few actions chosen by the vehicles, they may stay at their initial lanes (due to a penalty from the pinch module; no transition), or one of them may shift to the middle lane (A1→A2 or A1→A3). Without a lane sensing module described

in the previous section, the game ends there. Otherwise, the vehicles will slow down or speed up in order to shift lanes.

Two example runs are shown in Figures 16 and 18. These are two of the many possible state transitions to reach the desired state (or states; A1, B1, or C1). Also note that all the states shown in Figure 15(a) have two substates: the positions of the vehicle are switched to obtain the second possible substate. Figures 17 and 19 show the speed and lateral positions of the vehicles during the simulation. States are also indicated in the figures. The transitions during these two example runs are also marked in Figure 15(b).

Highway scenarios for three or more vehicles can also be defined as state transitions; however in this case, the number of states and therefore possible transitions is much larger than a simple two-vehicle scenario. However, multiple vehicle scenarios can be considered as a superposition of multiple simultaneous two-vehicle interactions<sup>30</sup>. Consider the situation in Figure 20. The speed and lateral positions of the vehicle are the same, and their desired lane parameters create a conflict. Similar to the first scenario, the solution to the problem lies in changing the relative speeds of the vehicles. Again, the lane flag is used to decrease the speeds of vehicles 1 and 2 to a smaller value than that of vehicle 3. Figure 21 shows some states of this three-vehicle scenario, and possible transition from initial state to a solution state. The relative positions of the vehicles are shown in a simplified matrix form where each row corresponds to a lane, and each square illustrates a road section that falls into the side sensor range of an automated vehicle. Not all possibilities are considered; only the situations that are of interest for this specific scenario are listed. Dark squares indicate the presence of a vehicle.

The key transition leading to a solution of the conflict is the first one (Figure 21) where vehicles 1 and 2 slow down and move away from the side sensor range of vehicle 3. All other

transitions are automatic under current circumstances. For the first transition, the lane flag needs to be set for at least one vehicle (if it is vehicle 3) breaking the symmetry. The problem and the solution for this case is similar to the two-vehicle situation. This is not a coincidence; it is due to the superposition of two two-vehicle situations. In terms of two-vehicle states, the state transitions of the three-vehicle case can be written as separate transition diagrams as shown in Figure 21.

It is therefore possible to analyze more complex highway situations by decomposing them into simpler cases. Transitions that need to be forced by the lane flag are shown in gray, and they are (and must be) between corresponding states in both three-vehicle and two-vehicle transition diagrams. The two-vehicle scenario including vehicles 1 and 2 is automatic, *i.e.*, there are no conflicts. The other two scenarios both have a synchronous ‘forced’ transition. A detailed analysis of more complex scenarios indicates that for an  $N$ -vehicle situation, the forced transition(s) must be ‘synchronous’ with key transition(s) of at least one of the 2-vehicle forced transitions<sup>30</sup>.

## 6. Concluding Remarks

The intelligent controller for vehicle path planning here consists of two stochastic learning automata. Using the data received from on-board sensors, each automaton can learn the best possible lateral/longitudinal action to be sent to the lower layer in the control hierarchy in order to avoid collisions. This non-model based method would be especially useful in situations wherein the complete knowledge about the flowing traffic is not provided by the higher levels of the control hierarchy (if such levels exist). Simulations for simultaneous lateral and longitudinal control provided encouraging results. This method is also capable of capturing the overall dynamics of the system that includes the vehicle, the driver and the roadway. Definitions of

learning parameters and update rates determine the behavior of the each vehicle as well as the sensor modules designed for path decisions.

Using this method, we also defined simple scenarios as games of multiple learning automata. The games are “played” between vehicles, and the result is a transition from one state to another state that defines the physical condition of the highway. Once the convergence to the optimal action for any given state is guaranteed, the problem is then to find necessary and sufficient rules to ‘force’ the vehicles to switch from state to state in order to reach the goal. The technological requirements for the models used in the simulations are no different from those defined in the current AHS research. Communication requirements, on the other hand, may be less. The more complex the scenarios, the more the need for a higher level intervention or more global information. Many maneuvers easily carried out manually prove to be extremely difficult even for a capable computing/communication system.

## **7. Acknowledgments**

This material is based upon work supported in part by the NRL under Grant no. N00014-93-1-G022, and in part by the CTR/VDOT under Smart Road Project.

## **8. References**

1. DOT’s IVHS Strategic Plan Report, in *11/10/1993 Hearing before the Subcom. on Invest. and Oversight of the Comm. on Sci., Space and Tech., 103 Congress, 1st Session*, pp. 35-36.
2. “Knowledge-Based Systems and Learning Methods for AHS,” Tech. Rep. FHWA-RD-95-045 by Raytheon, *AHS Precursor System Analyses*, Compendium of Research Summaries, compiled by Information Dynamics, Inc., and Vigen Corp., McLean, VA, February 1994.
3. Forbes, J., T. Huang, K. Kanazawa, and S. Russell, “The BATmobile: Towards a Bayesian Automated Taxi,” *Proc. of the 14<sup>th</sup> International Joint Conf. on AI*, Montreal, Canada, 1995.

4. Sukthankar, R., J. Hancock, S. Baluja, D. Pomerlau, and C. Thorpe, "Adaptive Intelligent Vehicle Modules for Tactical Driving," *Proc. of the 13<sup>th</sup> National Conf. on AI*, Portland, OR, 1996.
5. Varaiya, P., "Smart Cars on Smart Roads: Problems of Control," *IEEE Trans. on Auto. Ctrl.*, Feb. 1993, Vol. 38., No. 2, pp. 195-207.
6. Mourou, P., and B. Fade, "Multi-agent Planning and Execution Monitoring: Application to Highway Traffic," *Proc. of the AAAI Spring '92 Symposium*, pp. 107-112, 1992.
7. Niehaus, A., and R. F. Stengel, "Probability-Based Decision Making for Automated Highway Systems," *IEEE Trans. on Vehicular Tech.*, vol. 43, no. 3, pp. 626-634, 1994.
8. Cremer, J., J. Kearney, Y. Papelis, and R. Romano, "The Software Architecture for Scenario Control in the Iowa Driving Simulator," *Proc. of the 4<sup>th</sup> Computer Generated Forces and Behavioral Representation*, May 1994.
9. Godbole, D.N., J Lygeros, and S. Sastry, "Hierarchical Hybrid Control: An IVHS Case Study," *IEEE Ctrl. and Decision Conf.*, 1994, pp. 1592-1597.
10. Lygeros, J., and D. Godbole, "An Interface Between Continuous and Discrete-Event Controllers for Vehicle Automation," Tech. Rep. PATH Prog. 93-8, ITS, UC Berkeley, 1993.
11. Puri, A., and P. Varaiya, "Decidability of Hybrid Systems with Rectangular Differential Inclusions," *Proc. 6th Workshop on Comp.-Aided Verification*, Springer-Verlag, 1994.
12. Narendra, K.S., and M.A.L. Thathachar, "Learning Automata—A Survey," *IEEE Trans. in Systems, Man and Cybernetics*, 1974, Vol. SMC-4, No. 4.
13. Marsh, C., Gordon, T. J., and Q. H. Wu, "Stochastic Optimal Control of Active Vehicle Suspensions using Learning Automata," *Proc. I. Mech. Eng. Part I, Jour. of Syst. and Ctrl. Eng.*, vol. 207, pp.143-152, 1993.
14. Narendra, K.S., and M.A.L. Thathachar, Learning Automata, Prentice Hall, New Jersey, 1989.
15. Ashby, W. R., Design for a Brain: The Origin of Adaptive Behaviour, J. Wiley and Sons, New York, NY, 1960.
16. Bush, R.R., and F. Mosteller, Stochastic Models for Learning, Wiley, New York, 1958.
17. Atkinson, R.C., G.H. Bower and E.J. Crothers, An Introduction to Mathematical Learning Theory, Wiley, New York, 1965.
18. Tsetlin, M.L., Automaton Theory and Modeling of Biological Systems, Academic, NY, 1973.

19. Fu, K.S., "Stochastic Automata as Models of Learning Systems," in Computer and Information Sciences II, J.T. Lou, Editor, Academic, New York, 1967.
20. Najim, K., "Modeling and Self-adjusting Control of an Absorption Column," *International Journal of Adaptive Control and Signal Processing*, 1991, Vol. 5, pp. 335-345.
21. Gilbert, V., J. Thibault, and K. Najim, "Learning Automata for Control and Optimization of a Continuous Stirred Tank Fermenter," *IFAC Symp. on Adap. Syst. in Ctrl. and Sig. Proc.*, 1992.
22. Rajaraman, K., and P .S. Sastry, "Finite Time Analysis of the Pursuit Algorithm for Learning Automata," *IEEE Trans. on Syst., Man and Cyber., Part B*, 1996, Vol. 26, No. 4, pp.590-598.
23. Najim, K., and A. S. Poznyak, "Multimodal Searching Technique Based on Learning Automata with Continuous Input and Changing Number of Actions," *IEEE Trans. on Syst., Man and Cyber., Part B*, 1996, Vol. 26, No. 4, pp.666-673.
24. Najim, K., and A.S. Poznyak, eds., Learning Automata: theory and applications, Elsevier Science Ltd, Oxford, U.K., 1994.
25. Kushner, H.J., M.A.L. Thathachar, and S. Lakshmivarahan, "Two-state Automaton—a Counterexample," Dec. 1979, *IEEE Trans. on Syst., Man and Cyber.*, 1972, Vol. 2, pp.292-294.
26. Chandrasekharan, B., and D.W.C. Shen, "On Expediency and Convergence in Variable Structure Stochastic Automata," *IEEE Trans. on Syst. Sci. and Cyber.*, 1968, Vol.5, pp.145-149.
27. N. Baba, New Topics in Learning Automata Theory and Applications, Lecture Notes in Control and Information Sciences, Springer-Verlag, Berlin, 1984.
28. Ünsal, C., John S. Bay, and P. Kachroo, "On the Convergence of Linear Reward-Penalty Reinforcement Scheme for Stochastic Learning Automata," submitted to *IEEE Trans. in Syst., Man, and Cyber., Part B*, August 1996.
29. Ünsal, C., P. Kachroo, and John S. Bay, "Multiple Stochastic Learning Automata for Vehicle Path Control in an Automated Highway System," submitted to *IEEE Trans., in Syst., Man, and Cyber., Part B*, May 1996.
30. Ünsal, C., "Intelligent Navigation of Autonomous Vehicles in an Automated Highway System: Learning Methods and Interacting Vehicles Approach," Ph.D. Diss., Virginia Tech, February 1997 (Available online from Virginia Tech library system).
31. T.L. Lasky, and B. Ravani, "A Review of Research Related to Automated Highway System (AHS)," Interim Report for FHWA, No. DTFH61-93-C-00189, UC Davis, October 25, 1993.

32. Kachroo, P., K. Özbay, and R. Nagarajan, "Dynamic Visual Micro-macroscopic Traffic Software (DYNAVIMTS) Package for Studies on Automated Highway Systems (AHS)," 3<sup>rd</sup> ITS World Congress, Orlando, Florida, 1996.

33. Lakshmivarahan, S., Learning Algorithm Theory and Applications, Springer-Verlag, NY, 1981.

**Table 1.** Output of the *Left and Right Sensor* blocks.

Current Action	Sensor Status (L/R)	
	Vehicle in sensor range	No vehicle in sensor range
SiL	0 / 0	0 / 0
SL	1 / 0	0 / 0
SR	0 / 1	0 / 0
ACC, DEC, SM	# / #	# / #

**Table 2.** Output of the *Front Sensor* block (Regions defined in Figure 3).

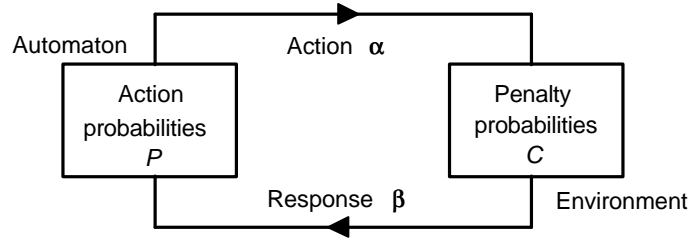
Current Action	Sensor Status				
	Vehicle in Region A (dist. < $d_1$ )	Vehicle in region B ( $d_1 < \text{dist.} < d_2$ )		Vehicle in region C ( $d_2 < \text{dist.} < fsr$ )	No vehicle in range (dist. > $fsr$ )
		Vehicle is approaching	Vehicle is NOT approaching		
SiL	1	1*	0	0	0
SL	0	0	0	0	0
SR	0	0	0	0	0
ACC	1	1	1	1	0
DEC	0*	0*	0	0	0
SM	1	1	0	0	0

**Table 3.** Output of the *Speed Sensor* block.

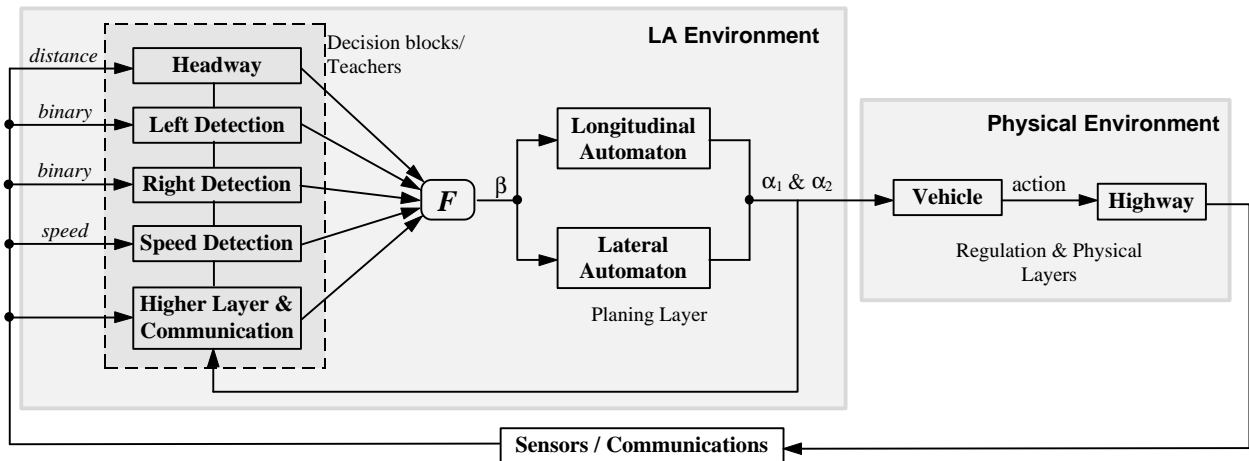
Current Action	Sensor Status		
	$dev < -ds$	$ dev  < ds$	$dev > ds$
ACC	0	0	1
DEC	1	0	0
SM	1	0	1
SL, SR or SiL	#	#	#

**Table 4.** Game matrix for situation A2 in Figure 14(a).

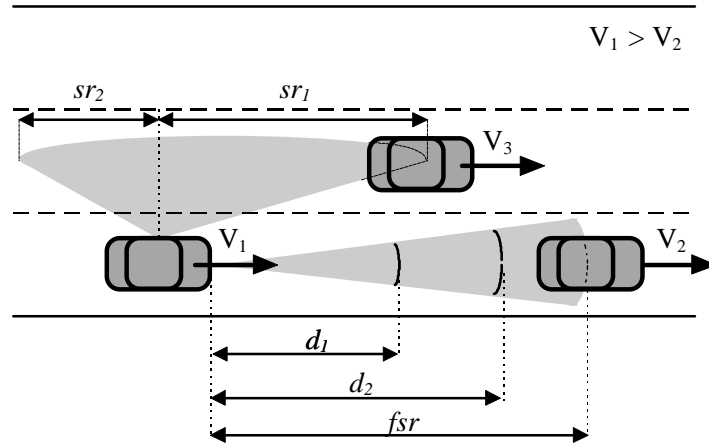
v2 \ v1	ACC	DEC	SM	SiL	SL	SR
ACC	Grey	Grey	Grey	Grey	Grey	Grey
DEC	Grey	White	Grey	Grey	Grey	Grey
SM	Grey	Grey	White	White	White	White
SiL	Grey	Grey	White	White	White	White
SL	Grey	Grey	White	White	White	White
SR	Grey	Grey	White	White	White	White



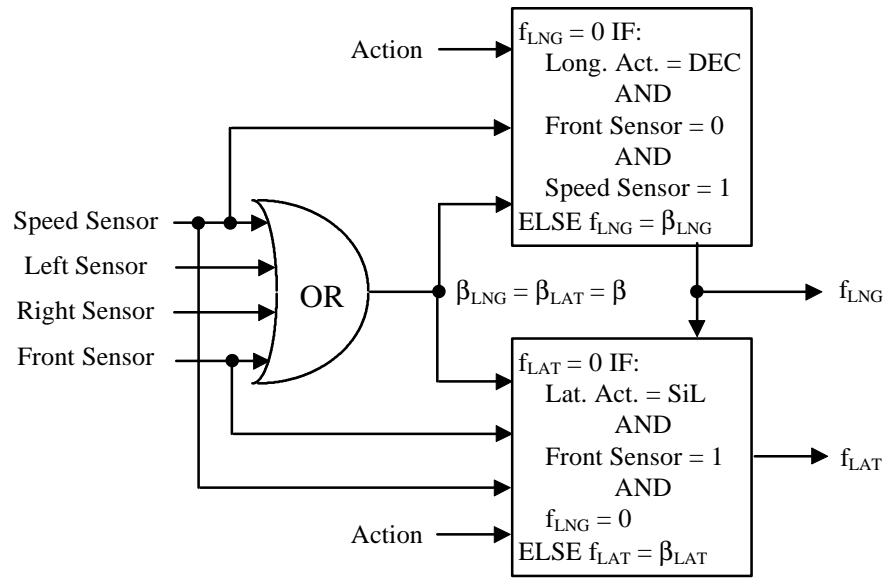
**Figure 1.** The automaton and the environment.



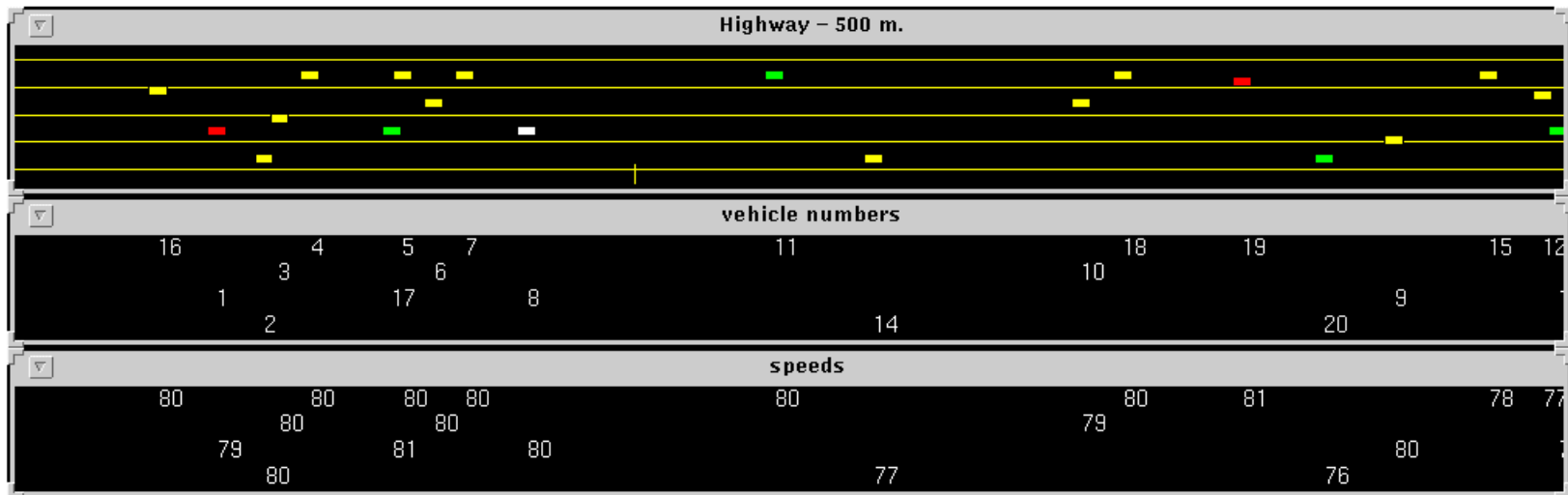
**Figure 2.** Automata in a multi-teacher environment connected to the physical environment.



**Figure 3.** The vehicle, sensing the approaching vehicle in front, can avoid a collision by shifting to the leftmost lane.



**Figure 4.** The definition of the mapping  $F$ .



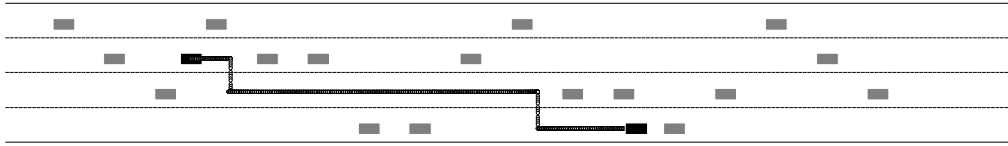
The bottom section contains four GUI panels. The "GUI" panel has sections for "sensors", "algorithm" (with "Lr\_p" slider), "parameters" (with "a: 0.15" and "b: 0.1" sliders), "Current" (with "Speed", "Des. Spd.", "Lane", "Des. Ln." fields), and "Automata" (with "longitudinal" and "lateral" checkboxes). The "Scenario" panel lists actions like "memvec", "mvec", "Spd&Ln", "T. Lap.", "Nvec", "2vec", "Flow", "Traj", and "Movie". The "Plot GUI" panel has similar controls. The "cmdtool - /bin/csh" panel shows a terminal log with simulation events.

```

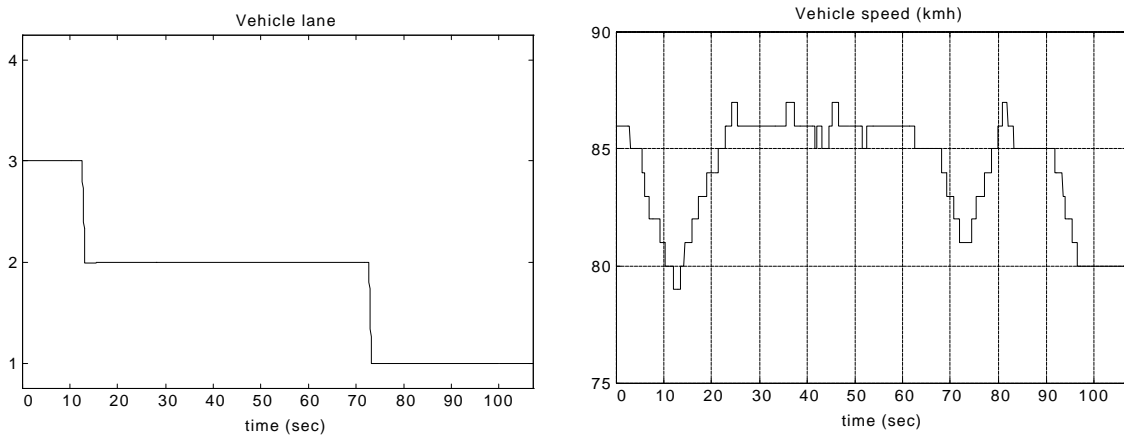
dec at 39 for 9, headway is 20 speed was 81
acc at 39 for 16, headway is 20 speed was 79
dec at 39 for 19, headway is 20 speed was 82
  left shift at 39 for 19
tt =
  40
  left shift at 41 for 9
  left shift at 44 for 3
  left shift at 44 for 16
vehicle 1 now in lane 2-spflag reset
vehicle 4 now in lane 4-spflag reset
  left shift at 47 for 12
vehicle 15 now in lane 4-spflag reset
pinch 3 - vehicle 1 in range and wants to shift left
tt =
  50
Simulation stopped at 2 sec.

```

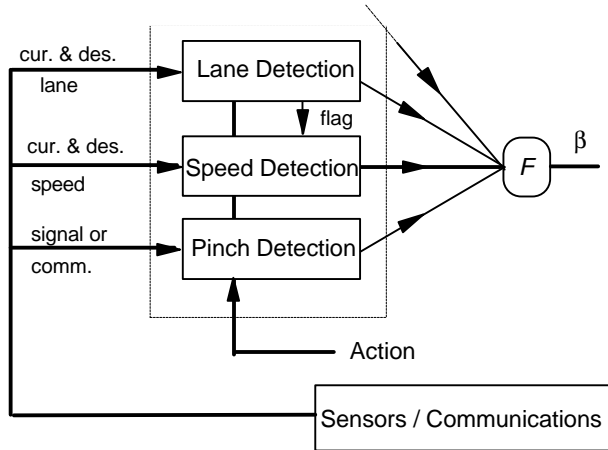
Figure 5. Simulation program.



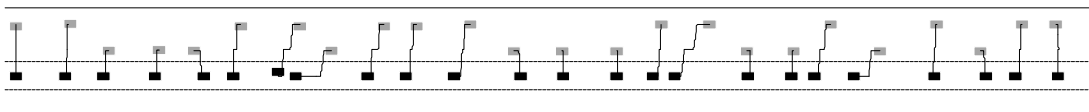
**Figure 6.** Trajectory of the controlled vehicle in 107 seconds.



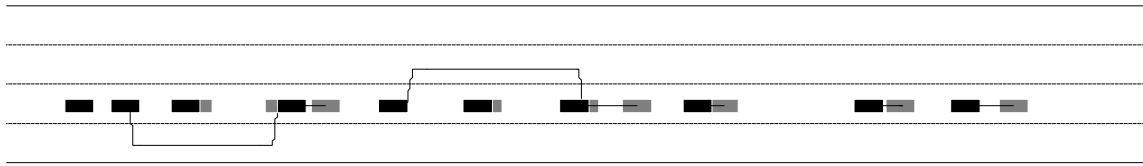
**Figure 7.** Lateral position and the speed of the controlled vehicle.



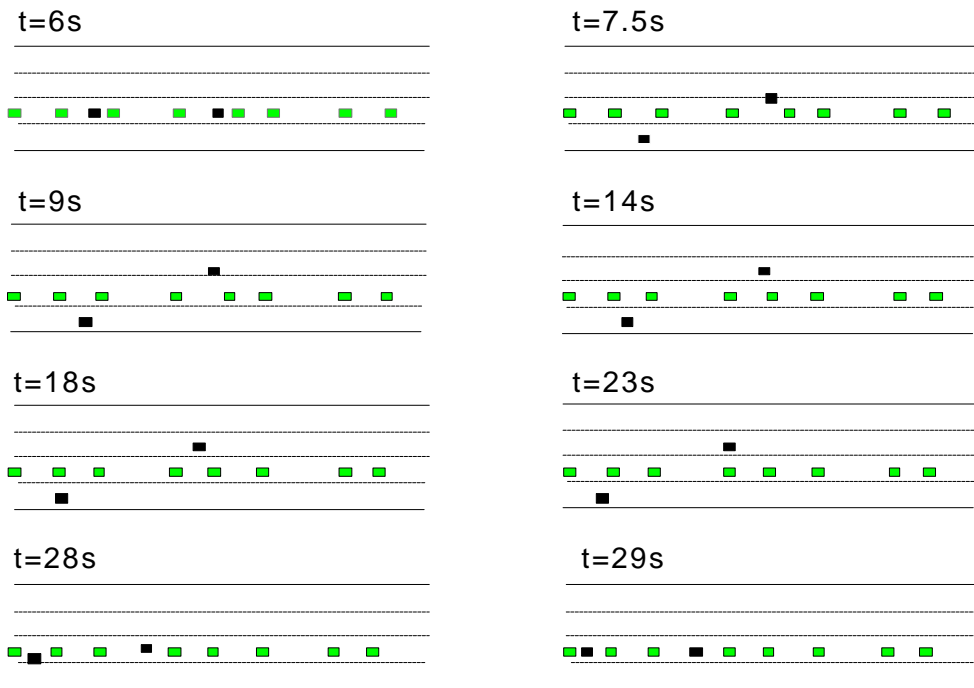
**Figure 8.** Modification of the vehicle decision blocks in Figure 2.



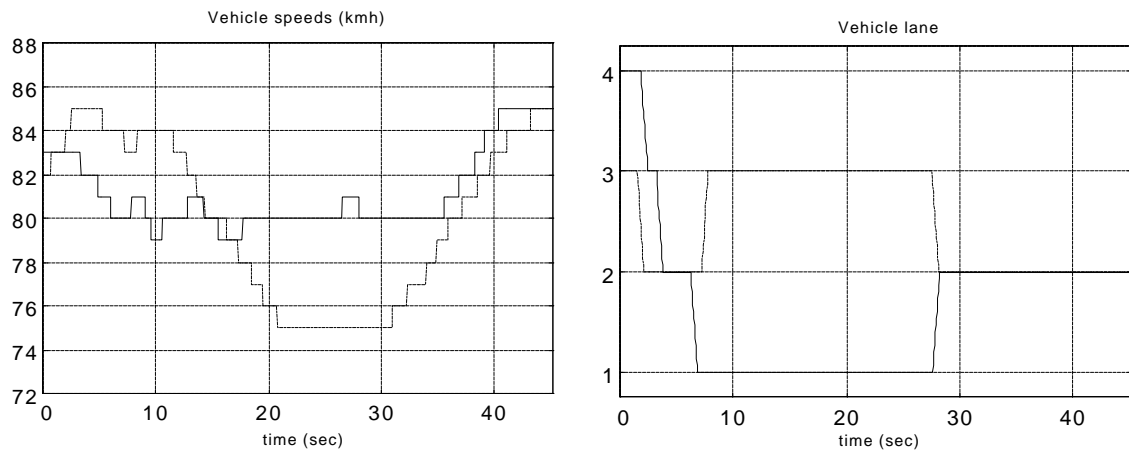
**Figure 9.** "Relative" trajectories of (24) vehicles from  $t=0$  to  $t=6$  sec.



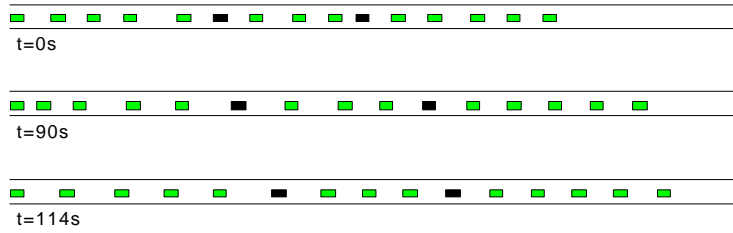
**Figure 10.** “Relative” trajectories of vehicles 13-22 from  $t=6$  to  $t=29$  sec (Vehicle 13, the first from left, is the reference).



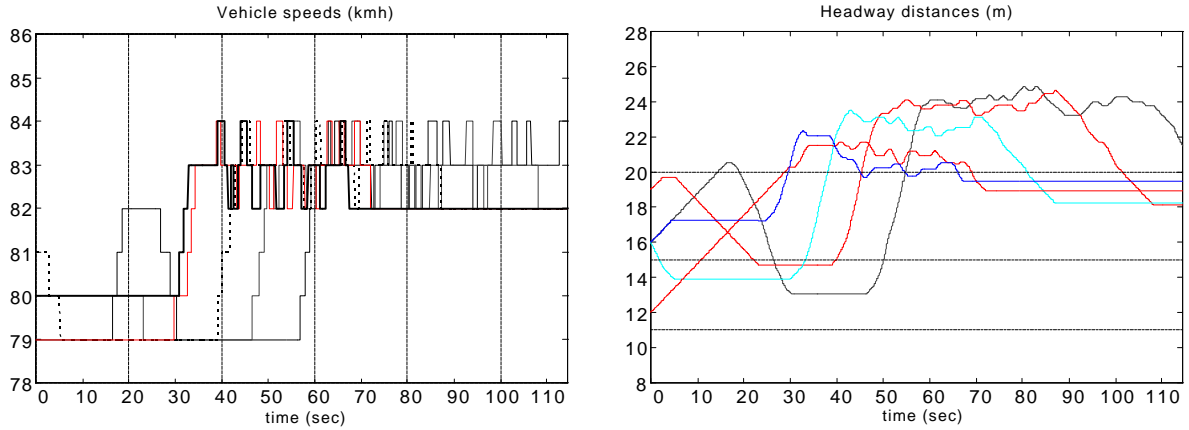
**Figure 11.** Positions of vehicles 13-22 on a 4-lane highway.



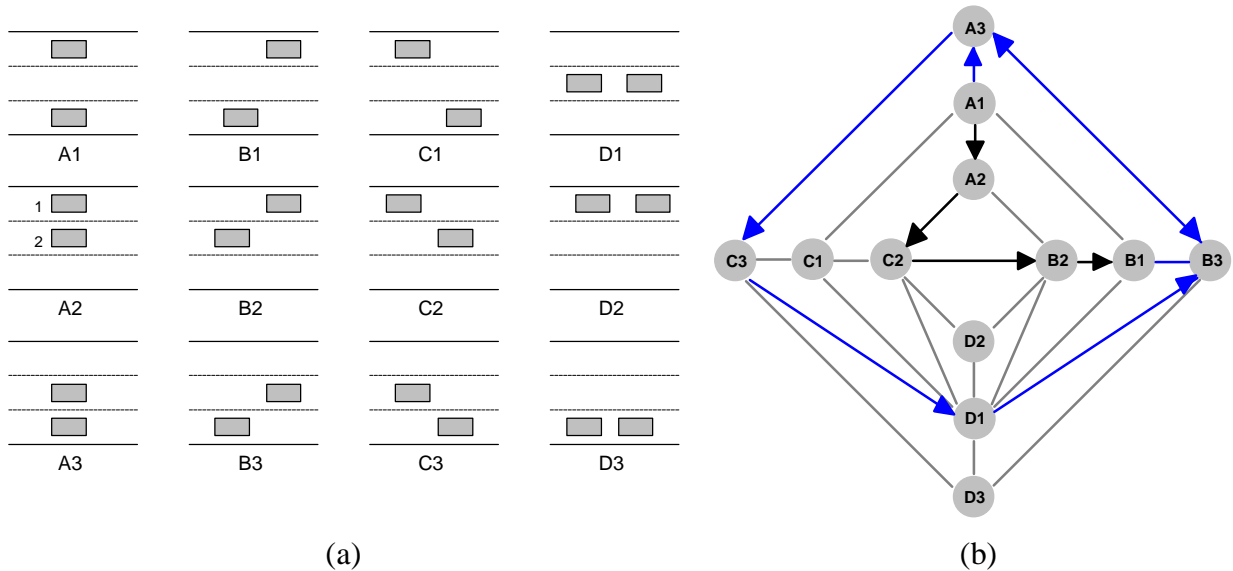
**Figure 12.** Speed and lateral positions of vehicles 15 and 18 (indicated with darker gray tone in Figure 11).



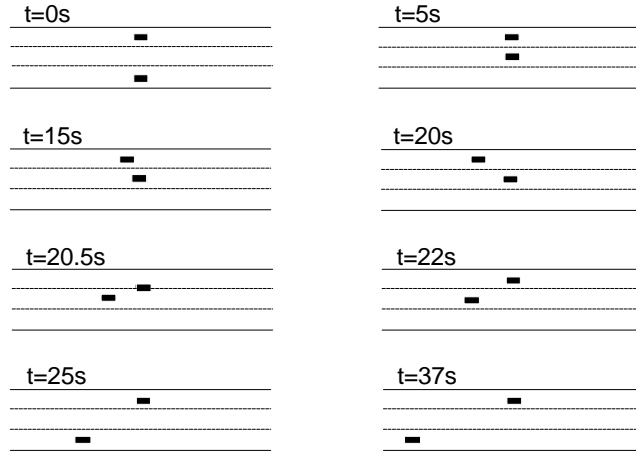
**Figure 13.** Relative locations of (15) vehicles at different time steps.



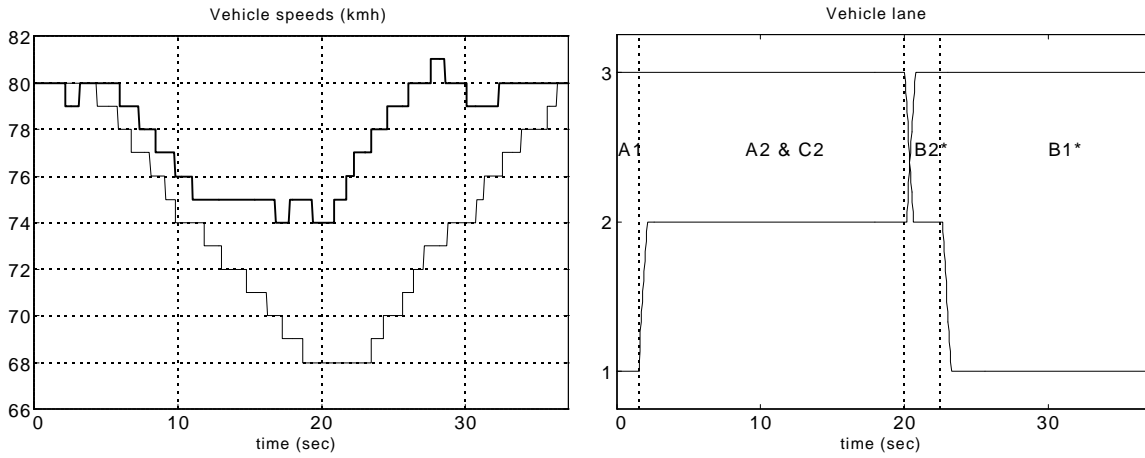
**Figure 14.** Speed and headway distance changes for vehicles 6,7,8,9 and 10 (Vehicle 6 and 10 are shown in darker color in Figure 13).



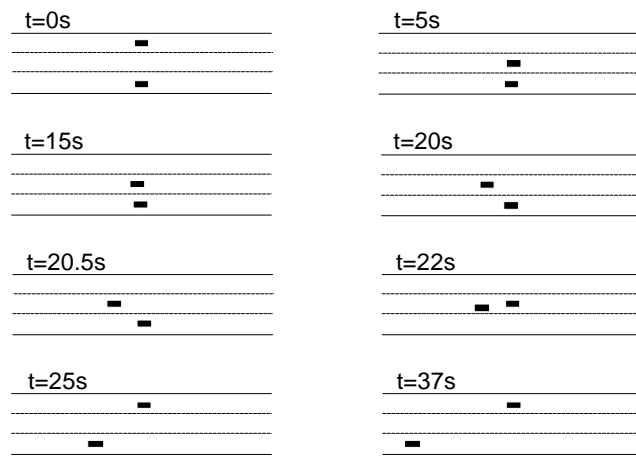
**Figure 15.** (a) Possible states and (b) state transitions for 2-vehicle 3-lane game.



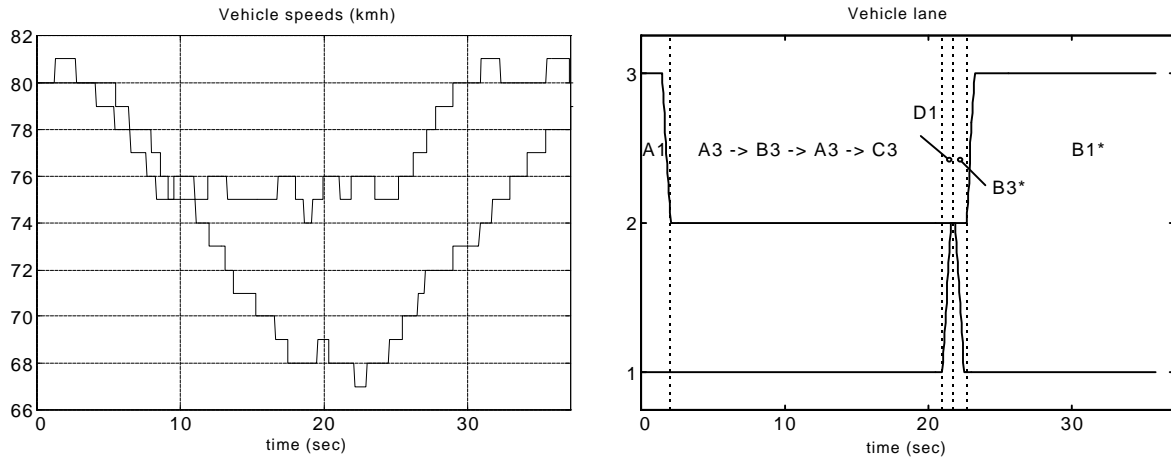
**Figure 16.** Positions of two vehicles in a 3-lane highway.



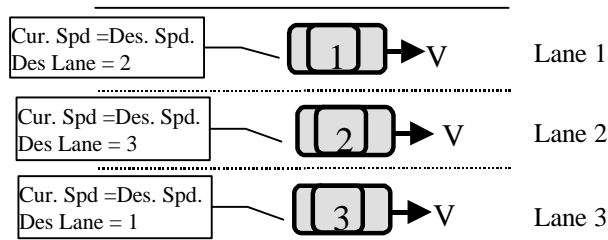
**Figure 17.** Speed and lateral position of two vehicles in a 3-lane highway (See Figure 16).



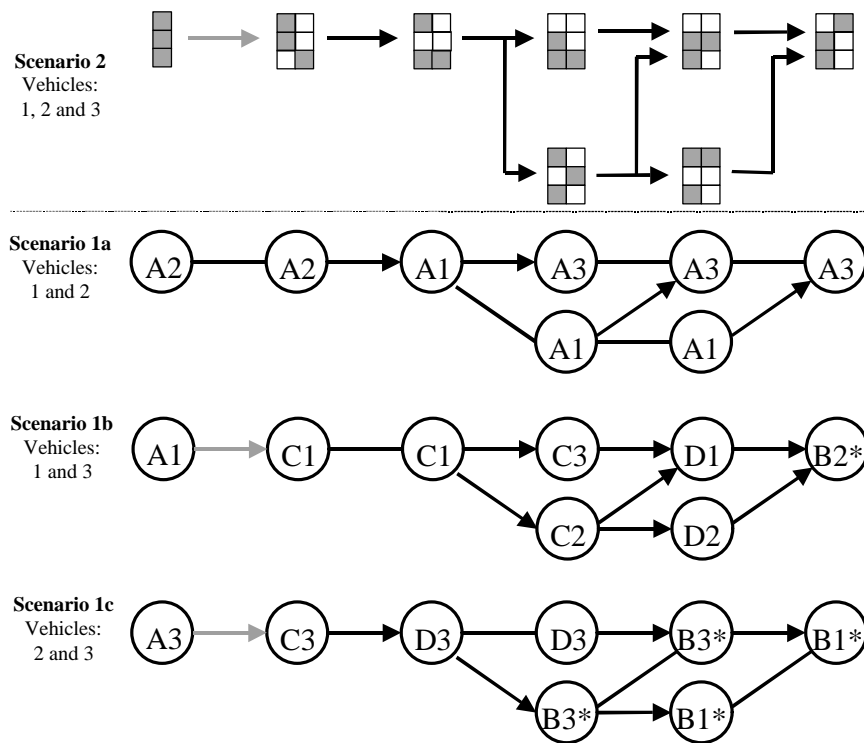
**Figure 18.** Positions of two vehicles in a 3-lane highway.



**Figure 19.** Speeds and lateral positions of two vehicles in a 3-lane highway (See Figure 18).



**Figure 20.** Three vehicles with conflicting desired paths.



**Figure 21.** Three-vehicle transition diagram can be represented as three separate 2-vehicle transition diagrams using the definitions in Figure 15.

### Author Biographies:

Cem Ünsal  
Postdoctoral Fellow  
Robotics Institute  
Carnegie Mellon University

Cem Ünsal received the B.S. degree in electrical engineering from Bogazici University, Istanbul, Turkey in 1991, and M.S. and Ph.D. degrees in electrical engineering from Virginia Polytechnic Institute and State University (Virginia Tech), in 1993 and 1997 respectively. He is currently working as a postdoctoral fellow at the Robotics Institute, Carnegie Mellon University, Pittsburgh. His research interests include sensor modeling and simulation, automated highway systems, intelligent vehicle control systems, learning automata, and nonlinear observers. Mr. Ünsal is a member of IEEE and ITSA.

Pushkin Kachroo  
Assistant Professor  
The Bradley Department of Electrical Engineering  
Virginia Polytechnic Institute and State University

Pushkin Kachroo is an Assistant Professor at the Bradley Department of Electrical Engineering at Virginia Tech. He was a research scientist at the Center for Transportation Research before that for about two and a half years. Prior to that, he worked at Lincoln Electric Company in the robotics R&D laboratory. He received his Ph.D degree from U.C.Berkeley in 1993, his M.S degree from Rice University in 1990, and his B.Tech. from I.I.T Bombay in 1988. He is the chairman of the Control Systems Society workgroup on Intelligent Vehicle Highway Systems, chairman of Automatic Traffic and Vehicle Control Systems, SPIE Conference 1997, and was the chairman of the Mobile Robots session of the SPIE conference 1996.

John S. Bay  
Associate Professor  
Bradley Department of Electrical Engineering  
Virginia Polytechnic Institute and State University

John S. Bay received the B.S.E.E. degree from Virginia Polytechnic Institute and State University (Virginia Tech), and the M.S. and Ph.D. degrees in electrical engineering at The Ohio State University in Columbus, OH. He is an Associate Professor of Electrical Engineering at Virginia Tech, where he has taught and performed research in control systems and robotics since 1989. His research interests include kinematics and dynamics, nonlinear dynamical systems and control, distributed artificial intelligence and sensor-based robotics. He is currently working in the area of intelligent, many-agent dynamics and multi-robot coordination and control. Dr. Bay is a member of the IEEE and the ASEE, and is a speaker in the Distinguished Visitor Program of the IEEE Computer Society. He also serves as Associate Editor for *IEEE Control Systems Magazine*.