Precursor Systems Analyses of Automated Highway Systems

RESOURCE MATERIALS

Knowledge Based Systems and Learning Methods for AHS

U.S. Department of Transportation
Federal Highway Administration

Publication No. FHWA-RD-95-106
February 1995

FOREWORD

This report was a product of the Federal Highway Administration's Automated Highway System (AHS) Precursor Systems Analyses (PSA) studies. The AHS Program is part of the larger Department of Transportation (DOT) Intelligent Transportation Systems (ITS) Program and is a multi-year, multi-phase effort to develop the next major upgrade of our nation's vehicle-highway system.

The PSA studies were part of an initial Analysis Phase of the AHS Program and were initiated to identify the high level issues and risks associated with automated highway systems. Fifteen interdisciplinary contractor teams were selected to conduct these studies. The studies were structured around the following 16 activity areas:

(A) Urban and Rural AHS Comparison, (B) Automated Check-In, (C) Automated Check-Out, (D) Lateral and Longitudinal Control Analysis, (E) Malfunction Management and Analysis, (F) Commercial and Transit AHS Analysis, (G) Comparable Systems Analysis, (H) AHS Roadway Deployment Analysis, (I) Impact of AHS on Surrounding Non-AHS Roadways, (J) AHS Entry/Exit Implementation, (K) AHS Roadway Operational Analysis, (L) Vehicle Operational Analysis, (M) Alternative Propulsion Systems Impact, (N) AHS Safety Issues, (O) Institutional and Societal Aspects, and (P) Preliminary Cost/Benefit Factors Analysis.

To provide diverse perspectives, each of these 16 activity areas was studied by at least three of the contractor teams. Also, two of the contractor teams studied all 16 activity areas to provide a synergistic approach to their analyses. The combination of the individual activity studies and additional study topics resulted in a total of 69 studies. Individual reports, such as this one, have been prepared for each of these studies. In addition, each of the eight contractor teams that studied more than one activity area produced a report that summarized all their findings.

Lyle Saxton
Director, Office of Safety and Traffic Operations Research and Development

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof. This report does not constitute a standard, specification, or regulation.

The United States Government does not endorse products or manufacturers. Trade and manufacturers' names appear in this report only because they are considered essential to the object of the document.

1. Report No. FHWA-RD-95-XXX	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Precursor Systems Analyses Of Automated Highway System Final Report Knowledge Based Systems and Learning Methods for AHS Volume Ten		5. Report Date 6. Performing Organization Code
7. Author(s) J. Schmoltz, A. Blumer, J. Noonan assisted by D. Shedd, J. Twarog, K. Assiter		8. Performing Organization Report No.
9. Performing Organization Name and Address Tufts University EE/CS Department Medford,Ma 02155 Under Subcontract to: Raytheon Comppany Missile Systems Division 50 Apple Hill Drive Tewksbury, Ma 01876		10. Work Unit No. (TRAIS) 11. Contract or Grant No. DTFH61-93-C-00196
12. Sponsoring Agency Name and Address The Federal Highway Administration 400 Seventh Street, S.W. Washington, D.C. 20590		13. Type of Report and Period Covered Final Report Sept 1993 - Feb 1995 14. Sponsoring Agency Code

15. Supplementary Notes

Contracting Officer's Technical Representative (CTOR) - J. Richard Bishop (HSR-12)

16. Abstract

This document is the final report of the Automated Highway System.The activities of Knowledge Based Systems and Learning Methods for AHS are reported on in this document. This document is resource materials. This volume is tenth in a series.

FHWA-RD-95-XXX Volume 1 Executive Summary
FHWA-RD-95-XXX Volume 2: Automated Check In
FHWA-RD-95-XXX Volume 3: Automated Check Out
FHWA-RD-95-XXX Volume 4 Lateral and Longitudinal Control
FHWA-RD-95-XXX Volume 5 Malfunction Management and Analysis
FHWA-RD-95-XXX Volume 6 Commercial Vehicle and Transit AHS Analysis
FHWA-RD-95-XXX Volume 7 Entry/Exit Implementation
FHWA-RD-95-XXX Volume 8 Vehicle Operational Analysis
FHWA-RD-95-XXX Volume 9 AHS Safety Issues
FHWA-RD-95-XXX Volume 10 Knowledge Based Systems and Learning Methods

17. Key Words Automated Highway System, Check in, Check out,Lateral and Longitudinal Control, Malfunction Management, Entry/Exit, Safety, Commercial, Transit Knowledge Based Systems	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 35	22. Price

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AASHTO	American Association of State Highway and Transportation Officials
ABS	anti-lock brake system
AHS	Automated Highway System
AICC	Autonomous Intelligent Cruise Control
AICC	Adaptive Intelligent Cruise Control
ALK	automatic lane keeping
ANSI	American National Standards Institute
APTS	Advanced Public Transportation System
ASR	anti wheel spin regulation
ASTM	American Society for Testing and Materials
ATIS	Advanced Traveler Information System
ATMS	Advanced Traffic Management System
AVCS	Advanced Vehicle Control Systems
AVI	automated vehicle identification
AVL	automated vehicle location
BAA	Broad Agency Announcement
CMAC	Cerbellar Model Articulation Controller
CLT	crossing left turn
CVO	Commercial Vehicle Operations
CVSA	Commercial Vehicle Safety Alliance
DOT	Department of Transportation
EIA	Electronics Industry Association
EMS	emergency medical services
ERSC	Evolutionary Representative System Configuration
ETC	electronic toll collection
EVM	emergency vehicle management
FARS	Fatal Accident Reporting System
FHWA	Federal Highway Administration
FLIR	forward looking infrared radar
FTA	Federal Transit Administration
GES	General Estimates System
GPS	Global Positioning Satellite navigation system
HAR	Highway Advisory Radio
HAZMT	hazardous materials
HDS	headway detection system
HOV	high occupancy vehicle
HUD	head-up display
ICAS	intersection collision avoidance system
ICC	intelligent cruise control

ISTEA	Intermodal Surface Transportation Efficiency Act
ITE	Institute of Transportation Engineers
IVHS	Intelligent Vehicle Highway System
KBCon	Knowledge Based Controller
LCM	lane change / merge collisions
LED	light-emitting diode display
LORAN	Land-based Radio Navigation System
MCSAP	Motor Carrier Safety Assistance Program
NAB	National Association of Broadcasters
NEC	Northeast corridor
NEMA	National Electrical Manufacturer Association
NHTSA	National Highway Traffic Safety Administration
PATH	Program for Advanced Transit and Highways (California)
PCD	personal communication device
PDS	proximity detection system
PPT	personalized public transit
PRT	perception reaction time
PSA	precursor systems analyses
PSAP	public safety answering point
PSS	Public Safety Services
PTS	public travel security
RECA	rear end collision avoidance
RECW	rear end collision warning
RSC	representative system configuration
RSPA	Research and Special Programs Administration
RTTASC	real-time traffic-adaptive signal control
SAE	Society of Automotive Engineers
SCP	straight crossing path
SHMS	speed headway maintanence system
SOV	single occupancy vehicle
SRVD	single vehicle roadway departure
TCS	traction control system
TDM	travel demand management
TIA	Telecommunications Industry Association
TMC	traffic management center
TAHSK	Tufts Automated Highway System Kit
VMS	variable message sign
VMT	vehicle miles traveled
VRC	vehicle-to-roadside communication

1.0 INTRODUCTION

Managing each AHS vehicle and the AHS system as a whole is an extremely complex undertaking. We have investigated and now report on Artificial Intelligence (AI) approaches that can help. In particular, we focus on AI technologies known as Knowledge Based Systems (KBSs) and Learning Methods (LMs). Our primary purpose is to identify opportunities: we identify several problems in AHS and AI technologies that can solve them. Our secondary purpose is to examine in some detail a subset of these opportunities: we examine how KBSs and LMs can help in controlling the high level movements -- e.g., keep in lane, change lanes, speed up, slow down -- of an automated vehicle. This detailed examination includes the implementation of a prototype system having three primary components. The Tufts Automated Highway System Kit (TAHSK) discrete time micro-level traffic simulator is a generic AHS simulator. TAHSK interfaces with the Knowledge Based Controller (KBCon) knowledge based high level controller, which controls the high level actions of individual AHS vehicles. Finally, TAHSK also interfaces with a reinforcement learning (RL) module that was used to explore the possibilities of RL techniques in an AHS environment.

We begin with an executive summary (section 1), followed by introduction to knowledge based systems and learning methods (section 2). Next, we review the literature on how these methodologies have been applied to AHS to date (section 3). We then list the areas of AHS that we have identified as opportunities where AI technologies can help (section 4). Next, we describe an in-depth study of two of these opportunities, which includes the construction of the implementation described above (section 5). Finally, we summarize (section 6), and list our references (section 7).

2.0 EXECUTIVE SUMMARY

Our primary **conclusions** are that AI technologies, particularly knowledge based systems and learning methods, can make strong contributions to AHS, especially in the following four areas.

- *Management of malfunctions onboard the vehicle*: While onboard malfunction detection is probably left to more conventional technologies, determining the best response to malfunctions may need the power and flexibility that AI technologies can bring to bear.

- *Management of each automated vehicle in traffic* (i.e., controlling its high level movements such as stay in lane, change lanes, exit, speedup, etc.): In order to deal with the inherent complexity of AHS, a powerful and flexible high level control system is needed. AI technologies can help meet this need.

- *Overall traffic management*: For efficient and effective use of automated highways, AI scheduling and planning methods can be extremely beneficial.

- *Sensor interpretation and fusion*: In particular, there is vast potential for machine vision to collect valuable data about the environment. Moreover, the problem of fusing data from several sources into a coherent picture of the environment may be helped by AI technologies.

In any eventual AHS, there will be widely varying capabilities among the possibly millions of AHS vehicles because there will be a mixture of new and old models, plus there will be widely varying levels of maintenance. Also, with that many vehicles, there will be numerous malfunctions. Hopefully, these will mostly be minor, but some will be major. Moreover, the world is complex and unexpected things can happen, such as a moose entering the highway, or a refrigerator falling off of a truck?

The conclusion that we draw from the above intuitive argument is that, even with full automation, the AHS environment will not be highly structured. By that, we mean that it will not be so easily predicted nor subject to analysis. Moreover, if we allow for AHS environments that allow mixed populations of "drivers" -- i.e., mixing human drivers with automated vehicles -- then the structure of AHS and the predictability of any given situation drops enormously. As a result, in order to manage an automated vehicle in an AHS, the vehicle must be able to deal effectively with an enormous number of possible traffic situations. The number of situations is enormous because we include situations involving minor and major malfunctions, plus unexpected events (e.g., moose on the highway).

Our **recommendations** regarding the foregoing are as follows.

- We should incorporate the management of malfunctions, both onboard and in other vehicles, into *normal* operations as much as possible.

- We should design each AHS vehicle to have effective predetermined responses to a large number of traffic situations but

- We should also design each AHS vehicle to be able to respond effectively to new situations.

The last recommendation places a significant burden on AHS. The traditional engineering approach has the human engineer performing all of the analysis and design work. The result is a machine or piece of software that meets the original requirements but which does not include explicitly any of the knowledge that went into the design. In other words, the product typically cannot modify itself to handle situations unforeseen by the engineer. But this is precisely what we are recommending. We should give to each vehicle that capability to *design* a response to a new situation as well as to *execute* that response. For this type of capability, we must turn to AI technologies.

We note that our effort has focused primarily on the latter phases of AHS. In the context of our group's ERSCs, we have focused on ERSCs 4 and 5, where there is full automation of vehicles. There is a role for AI technologies earlier in the evolutionary process. However, given that this task had limited resources, we focused on these larger issues. Furthermore, of the four areas that we have identified where AI can be helpful, we have focused on the first and second. Namely, we have examined how to control an individual AHS vehicle at a high level in a variety of traffic situations, with a particular emphasis on situations where there are malfunctions either onboard the vehicle or in nearby vehicles.

One might ask: Why is controlling the high level actions of a vehicle so complicated?

- It is an expert behavior, as exemplified by the fact that there is wide variability in the way that people drive, and that there are recognized expert drivers.
- The available data about the environment is inaccurate and incomplete. For example, if the car to your left is beginning to swerve into your lane, you cannot tell if it is due to wind, due to a momentary lapse by the driver, or due an intentional lane change (albeit a dangerous one). Thus your interpretation of a given situation
- Relies upon predictions of the behaviors of others. Sometimes these are merely educated guesses because there are not only the laws of physics at play here, but the intentions of the other "driver", be they human or machine.
- We must handle our own malfunctions. Should we immediately stop, move to a breakdown lane, or "limp home". Which is safe? Which is most effective for the overall traffic flow?
- We must respond to malfunctions in nearby vehicles. The car to our left is swerving towards us. What should we do?
- Many different situations may arise and many combinations of situations may arise.

We are thus left with the question: How can a system respond effectively to a large number of situations? One way is with a structure similar to a decision tree. The tree is designed by engineers offline and the vehicle uses the tree to determine responses online. But is this possible? Is the space of situations small enough or regular enough so that complete coverage can be gotten? We think not. Also, will the decision tree work is if the vehicle's information is incomplete? Will it get "stuck" in the tree, or might it be unable to disambiguate among possibilities?

There AI technologies that attend to this type of situation are expert systems, knowledge based planning, and learning methods. The latter two enable the system respond to new situations. Planning methods handle these situations online. Learning methods typically discover these new situations and the best responses offline using a simulated environment.

We therefore **recommend** the following.

- Each AHS vehicle have the capability to plan its own maneuvers and execute them.

- Further investigate the following areas, particularly with regard to AHS applications: KB planning (to handle new situations), learning methods (to handle new situations and to improve responses to already known situations), and expert systems (for overall high level control).

By incorporating the ability to plan new maneuvers, we gain new flexibility but we do not lose the ability to use preprogrammed responses to certain known situations. The vehicle should be designed so that if the current situation is familiar and the vehicle knows how to respond, then it should perform the preprogrammed response. However, if a new situation is encountered, or if the vehicle determines that the preprogrammed response is unsafe, then the vehicle should plan its own response if there is time. If there is not enough time, it should perform a backup emergency maneuver.

In our investigation of KBCon, the prototype KB high level controller we developed, we found that online planning was not as time consuming as we feared because the search space was deliberately kept small. However, we worked with a small prototype. With full planning capability, it is not clear whether or not a real time response is possible. Also, the current research in planning is not up to the challenge presented by AHS. In KBCon, we took many shortcuts in the planner. Considerable more research in planning for AHS-like environments, both basic and applied, is needed. For the other parts of KBCon, such as the classification methods used to predict the motion of nearby vehicles, we found that a simple knowledge representation scheme plus a rule based system was adequate for the task. By scaling up to realistic AHS size, we do not expect to encounter fundamental problems, except those of validation and verification, which are difficult to attain in any AI system. Also, while we cannot be certain given our limited study, it is likely that this portion of the system can be made to respond in real time.

We recommend that reinforcement learning (RL) , in conjunction with an AHS simulator, be used as a tool to help design controllers. We do not recommend that an RL module be a component in an actual AHS system. Since an AHS system will be operating at velocities and time intervals unfamiliar to human drivers, we should not rely solely on human experience when designing an AHS vehicle controller. We believe RL can be used to help design controllers, or to improve existing controllers by adjusting parameters such as headways. As part of our study, we implemented an RL module and ran it in the simulated TAHSK environment. Our RL module uses a three-layer feedforward artificial neural network to evaluate traffic situations. This network operates quickly enough to be useful for this research, however we recommend exploring the possibility of replacing it with a Cerbellar Model Articulation Controller (CMAC) network, which may exhibit better generalization abilities in this domain.

3.0 GENERAL BACKGROUND: What are Knowledge Based Systems and Learning Methods?

We begin with a general introduction to knowledge based systems and learning methods, beginning with the former. Interested readers might wish to see other, more detailed, introductions (e.g., [50, 15]).

What are Knowledge Based Systems?

All computer programs manipulate data to achieve their intended goals. A knowledge based system (KBS) manipulates not only data but knowledge itself. But what is *knowledge*? How can it be manipulated by a program? By *knowledge* in a computer we mean a structured collection of symbols that have specific meanings. By *manipulation of knowledge* we mean the manipulation of these symbols to produce new collections of symbols that have new meanings. Take a simple example. We wish to encode the following facts: (1) all cars are vehicles and (2) all vehicles can move under their own power. In a KBS system, we can encode this information in a *declarative* fashion, i.e., we can encode this information symbolically.

$$\forall x \; car(x) \blacklozenge \; vehicle(x)$$
$$\forall x \; vehicle(x) \blacklozenge \; selfpowered(x)$$

Here we have used first-order logic to encode this knowledge -- there are a large variety of languages one could use. The important thing is that there is a regular pattern to the symbol structures and there is a clear meaning to them.
Now that the information is encoded symbolically, we can write computer programs to do (at least) 2 things:

1. Apply it to specific situations and
2. Infer new knowledge from it.

Case 1 occurs when we are asked the question: Is a particular car self powered? By applying standard rules of logic, we can use the above to determine "Yes". For example, let CAR1 be a symbol representing a given car. We are given that:

$$car\,(CAR1)$$

and we infer that:

$$vehicle\,(CAR1) \text{ and } selfpowered\,(CAR1)$$

Case 2 is more powerful.. In this case, we can infer that all cars are self powered, i.e.,

$$\forall x\ car\,(x) \Leftrightarrow selfpowered\,(x)$$

It is the ability of KB systems to draw inferences -- i.e., to determine new knowledge about the world by combining bits of existing knowledge -- that gives them their power.
Of course, this power comes with a price. One must be very careful when using inference because inference, in general, is very complicated. It can take an unbounded amount of time and memory if left unfocused. Thus, KB systems must be carefully controlled in order for them to perform this type of high level reasoning and to have them do so in a reasonable amount of time.

Structure of a Knowledge Based System
A traditional computer program is typically designed to operate on a set of data, as sketched in figure 1. A KBS has a different overall design. While there is still a program and data, there is also a knowledge base. The knowledge base contains special types of data structures that represent *knowledge*, as discussed above. The capabilities of a traditional computer program arise solely from the program itself. The capabilities of a KBS arise from a combination of the program plus the knowledge base. In the most general case, the program in a KBS would be a theorem prover and the knowledge base would contain general rules, such as the ones listed above. The data would consist of information about particular objects, such as vehicles on the highway. The theorem prover plus the knowledge base plus the data, when put together, would cause new information to be inferred. The inferred information would cause the vehicle to make intelligent choices. For examine, it might infer that a particular action is dangerous while another action is prudent.

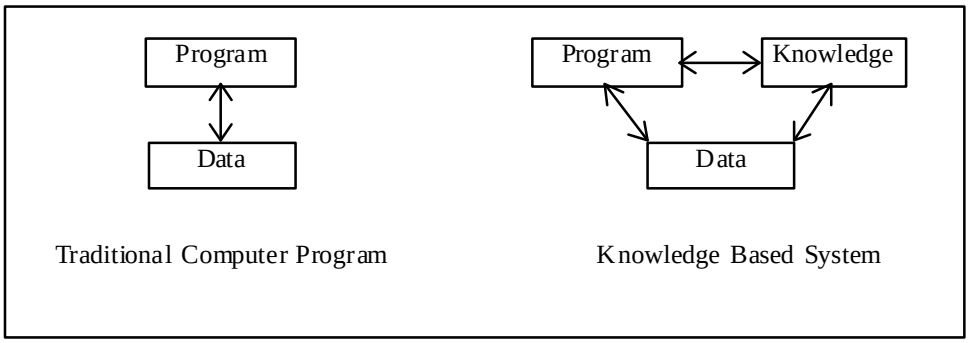

Figure 1. A traditional computer program versus a Knowledge Based System

Of course, a KBS based on a general theorem prover would be a poor choice because such programs are very difficult to focus, and keeping automated reasoning focused is essential when time is critical, such as with AHS. Fortunately, there are other methods for representing and reasoning about knowledge that are more focused and more efficient. One such method uses rules to capture knowledge. We briefly review this topic after presenting the needs that AHS has for KBSs.

Why does AHS need Knowledge Based Systems?
The main reason is that the domain of AHS is extremely complex -- sufficiently complex that we cannot identify in advance all of the situations that an AHS will encounter. It is therefore unlikely that traditional programming techniques will suffice. We should try our best to identify all possible scenarios and to program the AHS to handle each one effectively and safely. But in order to prepare for the unexpected, we will probably need to give to the AHS more general knowledge like the kind shown above. We will likely need to arm it with the ability to figure out, by itself, how to handle novel situations. Thus, AHS will probably need the ability to reason about its actions, and that requires the techniques that KBSs offer.

There is another advantage to using declarative (i.e., symbolic) knowledge structures. This type of knowledge can be used in a variety of ways. For example the two rules above were used to infer information about a particular car, and also to infer information about all cars. It could also be used to help interpret data. For example, if an object appears to be moving under its own power, we could use the above structures to *suggest* that it is a vehicle (we could not infer this conclusively from only this knowledge). With more information, we might be able to suggest that it is a car. Once an object is identified as a car, the system is better able to predict its behavior (e.g., so as to avoid colliding with it).

Another advantage KBSs bring is the ability to represent knowledge at various levels of abstraction. When reasoning about overall traffic patterns, a vehicle can be abstracted to simply be a moving point. When reasoning about precisely when to change lanes, a more detailed representation of each vehicle can be used, such as one that takes into account the sizes, positions and velocities of the vehicles. All of these levels of abstraction can refer to the very same individuals and yet the differing levels can be kept distinct. In this way, we use the simplest representation for each task so as to simply the task. When we need more detail, we have it available, but we need not be swamped with detail all of the time.

A final advantage is that declarative structures are often easier to write and maintain than programs that try to encode the same information. If there is much knowledge to be encoded, it is often easier to get it right if people can easily read and understand the encoding -- and declarative structures are typically much easier to read than computer programs.

Rule Based Systems and Expert Systems

Rules provide a way to represent knowledge that can be used to infer new information. Here is an example of a rule that a vehicle might use to determine when it might pass another vehicle.

IF V is the vehicle just ahead and
V is moving too slow and
there is a passing lane to the left and
there is time to plan & execute a passing maneuver before coming too close to V
THEN plan and execute a maneuver to pass V on the left.

In typical rule systems, the rules are always operating. In the above case, whenever some other vehicle V meets the criteria in the first part of the rule (often called the *left hand side* or LHS or the *if* part or the *antecedent*), then the system can execute, or infer, the second part (often called the *right hand side* or RHS or the *then* part or the *consequent*). Below is another rule that can determine when simply to stay behind and follow another vehicle.

IF V is the vehicle just ahead and
V is moving too slow and
either there is no passing lane to the left or
there is not enough time to plan & execute a passing maneuver
THEN plan and execute a maneuver to follow behind V.

Rules are typically used in either of two modes. *Forward-chaining* matches the LHS of each rule against a storehouse of information, whereupon some matching rule is selected as the one to be used next. This process repeats itself until no rules match any further. In the rule just above, forward-chaining would require that the knowledge base know about some other vehicle V that was just ahead and moving too slow, etc., before the rule system decided to use that rule. *Backward chaining* works in the opposite direction. It begins with someone or some part of the system asking a question. Rules that can answer the question via their RHS are invoked to see if their LHS is true. Each LHS is tested by the same process: for each clause in the LHS, a question is asked. Using the last rule as an example, the rule would be invoked only if there were some other vehicle V that the current vehicle was considering following behind. To see if indeed it should, it tries to show that the rule's LHS is true. It thus asks whether or not V is the vehicle just ahead. If that is true, it asks whether or not V is moving "too slow." This goes on until the LHS side is shown to be either true or false, during which time other rules might be invoked. If the LHS is shown to be true, then the RHS is true.

The power behind rule systems is that one does not need to program their execution explicitly. The LHS of each rule determines when the rule should be used, and so the selection of which rule to use at any given time can be fully automated. This not only greatly reduces the amount of programming needed, it reduces the potential for error. Moreover, since the rules select

themselves, the overall system is better able to deal with unexpected situations: the situations themselves trigger the proper rules to use, not the foresight of the AHS programmer.

Of course, using rules does not guarantee that the system will operate correctly: one must write rules that work. Thus, there is still a great amount of thought and consideration and testing that needs to be done.

Expert systems are KBSs that exhibit expert behavior. Typically, an expert system addresses a narrow but complex area and demonstrates considerable expertise in mastering that area. Expert systems control manufacturing processes, diagnose malfunctions in devices, assist in identifying likely places to mine, analyze the likelihood of finding oil, and a host of other tasks. Some estimates say that there are from hundreds to thousands of commercial expert systems either in use or in development [13]. Although any KBS technology can be used to implement an expert system, often the rule system technology is used at least in part.

Planning the Vehicle's Actions

Some KBSs *plan* their activities. By this, we mean that the system deliberates for a time and designs a sequence of actions that it should take in order to achieve its goals. An example might be to plan a passing maneuver. In this scenario, our vehicle is approaching a slower vehicle from behind. The question is: Do we pass or not, and if so, how?

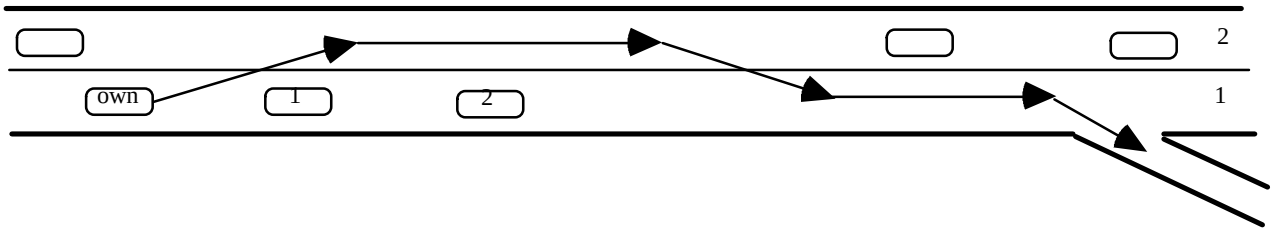

Figure 2. A Plan to Pass Two Vehicles

We might be able to construct a rule system, as suggested above, that would cover all possible scenarios and would makes the best choice in each case. As an alternative, we might let the vehicle use planning instead. A *plan* for the own vehicle (the *own* vehicle is the vehicle we are considering) to pass vehicles 1 and 2 might be a representation of the path in figure 2. Here, the own vehicle is in lane 1, will move to lane 2, will pass both vehicles 1 and 2, will return to lane 1, and will exit. The planner would construct this scenario, simulate it, and then evaluate the plan. Is the maneuver safe? Is it likely to succeed? Will the own vehicle make it to the exit? Finally, is it worth the effort?

For the vehicle to perform planning, we must give it considerable knowledge. It must know about the:

- *Physical world*: vehicles, roads, obstacles and motion.
- *Capabilities* of vehicles: move, brake, turn, communicate.
- *Requirements* for using these capabilities: e.g., to send a message, the transmitter must be operational, or to move, the vehicle must have gas (or battery power).
- *Effects* of using these capabilities: e.g., if the vehicle has gas and attempts to accelerate, then the vehicle will indeed accelerate.

- *Expected behaviors*: e.g., what should a “change lane” operation look like?

Moreover, the vehicle must be able to perform hypothetical reasoning. in that it must examine possible actions to take, and predict the outcome for each possible action based on simulation. Finally, it must be able to evaluate and select the best action to take.

Thus, a plan is developed dynamically with respect to the current situation and is represented declaratively. Multiple plans may be examined and evaluated, with some rejected. Planning can occur in background so that the on-line operation of the vehicle is unaffected until a given plan is implemented. Moreover, the vehicle’s plan can be developed so as to respect the plans of an overall traffic manager when the roadway has one.

What is Machine Learning?

Machine learning is the study of techniques that allow a machine to modify its behavior in response to unforeseen situations, or allow it to acquire behaviors for which it was not directly programmed. Machine learning techniques fall into several broad categories: supervised vs. unsupervised, and online vs. offline. A supervised learning system is presented with example situations together with the correct response to each example and is expected to learn to generate correct responses on its own. It is desirable that the system be able to generalize from the examples presented to it so that it can generate responses to unseen situations. A completely unsupervised learner just sees examples without being given any information that will allow it to infer a response, thus it is only able to classify examples into sets of similar examples or infer relationships between examples. Learning systems may fall between these extremes in several ways. A system may just be given a hint, rather than the complete response for a given situation. It may be given complete responses in some situations, hints in some others, and no information at all in some situations. A learning system may also contain an unsupervised component as well as a supervised component.

An offline learner is given all its examples at one time and infers a program to generate the correct response or a classifier that assigns examples to classes. Thus the learning phase is separated from the testing phase. An online learner must generate a response as each example is presented, gradually learning how to improve its performance. As with the supervised-unsupervised distinction, some learners will have both online and offline aspects. An offline learner can simulate an online learner if it is allowed to memorize all the examples and has enough time to reprocess the entire set of examples as each new example is seen. On the other hand, a learner may operate in an online fashion as it is learning, but be used in an offline fashion by a larger system. This may be because it is desirable to use the same sequence of examples repeatedly to ensure that as much as possible is learned from those examples, or because it is desirable to test or verify the program that the learner produces before using it in an application.

The particular learning techniques we have investigated are Artificial Neural Networks (ANNs) and Reinforcement Learning. These techniques work together in a synergistic way, as explained below. ANNs are actually a broad class of related machines, which can implement supervised, unsupervised, online, and offline learners. We will be interested in using them as a supervised online learners within the context of a reinforcement learning system. Reinforcement learning is a technique that allows offline or online learning in an environment that does not provide complete supervision. It assumes that a reward function is available, but this function may not

give an answer in all situations, and the answer may be incomplete. A good example is learning to play chess just by being told how many points were gained or lost in an exchange of pieces, and whether the game was a win, loss, or draw at the end of the game. This is not enough information to directly evaluate most moves, but together with some lookahead it can be used to select good moves.

An Artificial Neural Network is a network of interconnected neurons, each of which is a simple computational element [23, 10]. Neurons in ANNs are usually arranged in at least three layers: an input layer that receives data from the outside world, an output layer that gives the result of the network's computation, and one or more hidden layers that perform the internal computations. The connections between the neurons are used to transmit the output (usually a single number) of a neuron to the inputs of other neurons. Each such signal is modified by multiplying it by a weight associated with the connection over which it was transmitted. A neuron's output is computed by summing its weighted inputs plus a bias weight, and passing this sum through a nonlinear transfer function such as a sigmoid.

Despite their apparent simplicity, ANNs are capable of approximating any mathematical function. On the other hand, finding the proper settings of the weights to implement a given function may not be straightforward. This has led to a number of training algorithms that adaptively adjust the weights to approximate a function given by a set of input-output examples. An example of such a training algorithm is backpropagation, which uses a gradient-descent approach to iteratively adjust the weights. In a network where the neurons are arranged in levels such that neurons in a given level only send their output to neurons in the next level, backpropagation starts by calculating the difference between the desired output and the actual output. This difference is propagated backwards through the network to calculate the errors at each level. These errors are then used to adjust the weights in the network.

In an AHS environment, the inputs to an ANN might be the relative positions, velocities, and accelerations of nearby vehicles. The output could be a classification of the current state into a set of predefined states, such as {unsafe, safe to change lanes left, safe to change lanes right, etc.}. (Note that these states do not need to be mutually exclusive - neural networks naturally work well in a "fuzzy systems" setting.) Another possibility is that the output could be a number giving the relative safety of the current state. This could be used to select the safest next state from those states attainable from the current state by reasonable accelerations. A third possibility is that a network could be trained to output an objective measure of safety, such as the probability that the current situation will lead to an accident within 5 seconds if no change is made.

In order to train a network to do any of these, it needs to be presented with a lot of examples of traffic situations together with the desired output of the network. In some cases the desired output is easy to calculate. For example, if an accident is unavoidable it is clear that the state should be assigned a large negative value. In most cases, however, the desired output is not so easy to determine. One approach is to try to assign these values by hand for typical situations, and hope that when the network is trained from these examples it will generalize to similar situations. Another approach is to use the network in a learning algorithm that allows it to improve its performance from (simulated) experience.

Reinforcement learning is a class of such algorithms [43]. The version of reinforcement learning we have investigated is called Q-learning [47]. The Q-learner maintains an evaluation function (implemented by an ANN in our case) that evaluates proposed actions in any given state (configuration of nearby vehicles, including velocities and accelerations, in our case). This allows it to select the best action for any given state and to give a value to states according to the best action possible in that state. At each step the Q-learner selects and performs the best action and updates its evaluation function using the value of the new state and any reinforcement it received from the environment at that state.

4.0 Literature Review: Knowledge Based Systems and Learning Methods in AHS

The next section, 4, identifies a number of AHS areas where AI can make a significant contribution. Of these areas, we have focused primarily on the problem of controlling individual vehicles on an AHS, with particular concern to what we call *high level control*. Here, we are concerned with managing each vehicle in traffic -- e.g., deciding when to follow behind another vehicle, when to change lanes, when to exit, etc. We are not concerned with methods to maintain a safe following distance behind another vehicle (a longitudinal control systems manages that task) nor the overall flow of traffic (a centralized traffic manager handles that task). Thus, our focus is on AI applications to AHS in this particular area. While there are an enormous number of papers on KBSs and machine learning, only a small number are applied to AHS, and an even smaller number applied to high level control of individual vehicles. Before reviewing the most relevant works, we briefly review works that are less relevant.

In the summer of 1993, a workshop was held in conjunction with the national conference of the American Association for Artificial Intelligence. The overall conference is known as AAAI-93; the workshop was titled “Artificial Intelligence in Intelligent Vehicle Highway Systems”. Many of the papers were written by transportation engineers with the purpose of introducing IVHS and AHS to researchers in AI. Several papers were by AI researchers, though none addressed the problem of controlling individual automated vehicles. Wellman [49] presented a method for obtaining good performance of distributed algorithms using a scheme based on economic models. The type of distributed problem he analyzes is the overall traffic management problem, which he solves via a distributed algorithm -- i.e., there is no central controller. There were other papers on overall traffic management (e.g., [14, 24]). Other papers at the workshop addressed topics in IVHS that are not directly relevant to AHS.

There has been much AHS work performed by PATH, the group operated out of the University of California. In an important paper that summarizes the PATH work [40], there is no mention of AI applications. While attending a PATH course, held in Berkeley in early February 1994, some relevant work in AI was presented, though none of it directed towards high level vehicle control. In particular, Malik presented an algorithm that detected obstacles using stereo computer vision and alluded to another algorithm that performed lane keeping using machine vision [48, 21]. Also, Dickerson described a fuzzy controller that learned the throttle control function for a car [6, 7, 8].
Also at the PATH course, Varaiya presented his algorithm for controlling individual vehicles, which is summarized in [18, 46]. The work is well thought out and does not rely upon AI techniques except possibly at the highest level of path planning for each vehicle. Instead, it relies upon a highly structured environment, and using that, is able to control each vehicle using

fairly simple finite-state machine algorithms. For example, to change lanes, the vehicle's high level controller executes a simple finite state machine that finds a gap, signals its intention to change lanes, waits for an acknowledgment, performs the lane change, and terminates the operation. Complexities that might arise due either to malfunctions or to differences in protocols or expectations between vehicles are not dealt with at this level. Instead, all non-standard behavior is delegated to malfunction management, which we have not yet seen described in the literature. However, in the conclusion of [18], the authors note that AI technology will likely be of use here. In our view, this line of research may be assuming too much regarding the environment. Their methodology requires the environment to be both highly structured and highly predictable. But with the inherent complexity of the world (e.g., refrigerators falling off trucks), the variability of maintenance among AHS vehicles, and differences in both software and performance among AHS vehicles (many cars last for 10 to 15 years, during which time many software and hardware improvements will be made), we predict that these requirements might be too strong. Instead, we urge that a more flexible high level control system be developed, and we argue that KBSs and learning methods can play an important role here.

Gomi *et al* investigate using Brook's *subsumption architecture* [5] to perform both high and low level control for each vehicle [16, 17]. Without going into the details of this architecture, we point out two difficulties with this work. First, a subsumption architecture is difficult to analyze; for example, the authors measure its success by testing its performance in specific cases. This makes it difficult to judge its performance over large classes of situations, and thus hard to judge it overall. Second, the subsumption architecture is not designed for the inclusion of detailed knowledge. Thus, even if you know how to handle particular types of situations, it is difficult to encode this in the system. While it is premature for us to predict the eventual success of subsumption architectures for AHS, given the complexity of AHS and the need to encode specific safe behaviors, we are pessimistic as to its utility at this time.

We now move on to two pieces of research that are directly relevant to high level vehicle control. The first is by Okuno *et al* [37], who are developing an intelligent vehicle for the Mazda Corporation for use in an AHS type of environment. They call this functionality *autonomous cruising*. Their basic idea is to provide a set of *driving programs*, each of which is tailored to a particular type of situation. Situations include following behind another vehicle, changing lanes where the neighboring lane is empty, and changing lanes when the neighboring lane has vehicles at certain relative locations. Each driving program is written without using AI technology so as to insure its real time response. However, AI is used in the *cruise planner*: this module (1) identifies the current driving situation, (2) plans actions, and (3) activates the proper driving program.

Each driving program has a tight control loop involving perceptual inputs with updates at least every 0.1 sec. The cruise planner is a rule based system involving higher level information with updates about every 1 sec. This higher level information includes a highly symbolic model of the road and surrounding vehicles. The area around the vehicle of interest -- the *own* vehicle -- is split into 12 regions: near-back, near-front, near-side, back-side, etc. The speeds of surrounding vehicles are also expressed symbolically (e.g., fast, slow). The rules process this high level description of the current situation and determine the next action to take. Here is an example of a rule that notes when a lane change is dangerous due to a vehicle approaching quickly from behind and changes the current action so that the vehicle waits instead.

```
IF action = 'lane-change' and
   position[D] = 'back-side' and
   speed[D] = 'fast'
THEN action = 'waiting'
```

Okuno *et al* built a prototype system with a cruise planner that contained 140 rules. The inference speed of the cruise planner was about one second. In their experiments using simulation, the system operated well using a two lane highway. For their future work, they will supply more comprehensive knowledge to the cruise planner, such as traffic regulations, navigation strategies, and anticipating potential dangers. They note that inference speed is a very important issue, that the human interface is critical, including human override, and that they must treat obstacle avoidance as low level driving program, which they expect will be very difficult.

We have borrowed from Okuno *et al* in our work in that we also have a high level planner that selects *processes* for the vehicle to execute, which are similar to their driving programs. Also, our system abstracts information about the roadway and vehicles into high level abstraction whenever possible so as to simplify both the processing and the complexity of the logic needed. However, our approach is more flexible because our system will eventually have a greater ability to synthesize the processes it executes, and thus hopefully can respond effectively to a wider range of situations. Also, the planner that we are constructing can invoke numerous processes that communicate and operate in parallel, sequentially or a mixture of the two. At this time, we are less sensitive than Okuno *et al* to the real time requirements because we are still developing our ideas. With planning as a basic activity in both our approaches, it is possible that we will need vast amounts of time consuming computation to take place. However, it appears that the search space needed for the AHS domain will be relatively small, making the computations tractable. Further research is needed to confirm this.

The second piece of research that is directly relevant is that of Stengel and Niehaus [42, 34, 35], who have built expert systems for high level control of autonomous vehicles in an AHS environment. They assume the use of sophisticated lateral and longitudinal controllers, which we do also. Numerous components in their overall system are rule based systems. In [34], they developed a stochastic version of their system to deal with uncertainty in the data the vehicle collects about surrounding vehicles. Finally, they tested their system against a simulator and found that it performed well.

We find that the basic approach of Stengel and Niehaus is sound. They make useful abstractions in order to facilitate reasoning. The rules shown in the articles are clear and compelling, making it fairly plain how the overall system operates. The only difficulties with their approach is that the system is not very flexible nor robust. All configurations that the system handles must be pre-determined. For example, they have the concept of "waiting for a gap in left lane", which must be pre-programmed into the rules. Moreover, it appears as if there is no dynamic planning capability. If this is true, then they cannot plan to pass a vehicle in front and then exit from highway. Instead, each of these two actions is determined and then handled separately, one after the other. As we stated above, we consider a planning capability to be important for AHS, and so we have incorporated that capability into our design. Other than that difference, our approach is similar to that of Stengel and Niehaus.

Another area of relevant research is that of real time AI systems. Lockheed's L*STAR KB system [20] is a knowledge based system that performs satellite telemetry analysis for the Hubble space telescope. Ingrand *et al* report on a KBS that performs real time reasoning and control for process engineering applications [19]. Finally, numerous real time AI systems are presented in [25]. While none that we know of apply directly to AHS, these systems show that AI techniques can be successfully deployed in real time environments.

We have identified the importance of the ability to have each vehicle plan its actions in order to handle situations not foreseen by AHS designers. It is also important for the efficient use of the roadway. We distinguish macro-level traffic planning from micro-level planning of individual vehicle actions. Overall traffic planning is crucial to the efficient use of the highways, as others have noted (e.g., PATH is committed to this type of planning [46]). In this study, we focus instead on micro-level planning and will now review a small portion of the relevant AI planning literature.

Classical planning (e.g., [11, 39, 41]) addresses the following problem: given an initial situation, a goal, and a set of action types, devise a sequence of actions to achieve the goal. Over two decades of research have been invested here, yielding hundreds of papers and results. Much of this work makes the assumptions made by the STRIPS system [11], namely, that the robot (or agent or program) is in complete control of the world, that the world does not change except via the actions of the robot, that changes happen instantaneously, that the robot knows everything about the world that it needs to know, that the world is completely deterministic and predictable, and that the actions available to the robot are fairly simple. Even with all these assumptions, classical planning is intractable in general and computationally expensive in practice because the search space -- i.e., the set of possible action sequences that could be taken -- is typically huge.

In the past decade, planning researchers have relaxed many of these assumptions and made important contributions. For example, in a dynamic and changing environment -- one that changes in ways beyond the control of the robot and in ways that the robot cannot predict -- the classical approach of generating a complete plan in advance is pointless [33] given that much of the future is unpredictable. There are two approaches here. The first is incremental or partial planning (e.g., [22]), where one plans for a short time into the future, possibly leaving out many details, and expects to continue planning when more information is available. An example is planning to drive across town. One may plan the general route in advance, but details about when precisely to touch the gas pedal or to signal right are left to later planning when the situation arises. The second approach is to build a reactive system (e.g., [5, 1]) which is a system that does less planning but more reacting to the stimuli it receives from the environment. In AHS, we have a dynamic, somewhat unpredictable environment, and actions can occur over a considerable amount of time. We therefore found it necessary to use processes as the basic elements of our plans, similar to [36, 26], and we have chosen an incremental approach that retracts existing plans and replans when necessary.

We have not found any papers on reinforcement learning in an AHS setting, however there has been much recent research on RL for robotics in general. A number of papers at the 1994 MLC-COLT Workshop on Robot Learning present ideas that may be useful in an AHS setting [3, 9, 30, 31, 44, 45, 4].

5.0 OPPORTUNITIES FOR KNOWLEDGE BASED SYSTEMS AND

LEARNING METHODS TO HELP AHS

We have examined three broad areas of AHS where knowledge-based systems (KBSs) and machine learning methods (MLMs) can help.

- *Malfunction Management* -- Identifying on-board malfunctions and determining appropriate responses.

- *Management of the Vehicle in Traffic* -- Determining and controlling traffic-level operations (e.g., lane changes, entries/exits, merges, splits, obstacle avoidance, emergency maneuvers). We have referred previously to this function as *high level control*, a term taken from the field of Artificial Intelligence (AI).

- *Overall Traffic Management* -- Controlling the overall flow of traffic through the highway system.

In addition, we claim that AI systems will also be extremely valuable in the area of:

- *Sensor Interpretation and Fusion* -- Interpreting data from sensors, such as machine vision, and the fusion of that sensory data.

In this section, we present a number of specific problems from the first three areas that may benefit from the use of KBSs and MLMs. We do not examine that fourth area. Note that portions of the first two areas are studied in some detail, as explained in section 5.

5.1. Malfunction Management

We separate malfunction management into the two tasks of, first, detecting malfunctions and, second, taking the best response. We examine four avenues.

Malfunction management using classification

If we enumerate all possible malfunctions beforehand, then malfunction detection is essentially a classification problem. There is a fixed set of categories -- the possible malfunctions -- and the task is to classify the vehicle into zero, one or more of these categories at all times. KBSs, which include the category of systems known as *expert systems*, offer powerful techniques for solving classification problems, particularly when the rules for classification are complex, vague or involve judgments.

Many types of malfunctions are easily detected and will not require the power of KBSs. However, some malfunctions may be quite complex to diagnose. For example, imagine that the vehicle is in a platoon, is not leading the platoon, and is maintaining close headway. The malfunction system detects an unacceptable amount of fluctuation in the vehicle's speed. The cause might be in several places, such as the vehicle's longitudinal control system or in the sensors that detect the distance to the car in front. Let's assume that further malfunction checks on these subsystems conclude that they are operating properly. There is another possible cause: the car in front might be experiencing a similar fluctuation of speed. One option is to communicate with the car in front to see if it is experiencing the same problem. If so, then this

communication can go forward in the platoon until we identify a car in the platoon that does not have this fluctuation -- the problem probably lies with the car just behind it. In a sense, the car is diagnosing malfunctions in other cars. This is difficult and requires careful judgment. For high degrees of safety, vehicles must assess the functionality of nearby vehicles (see below).

Like detecting, determining the best response to a malfunction is also a classification problem if we enumerate the possible responses to each malfunction beforehand. However, this determination will frequently be complex because it may depend upon the surrounding environment of roadways and vehicles. In most cases, one can respond by bringing the vehicle to a stop in such a manner that it doesn't endanger vehicles behind. However, this creates an obstruction on the highway. Can the vehicle "limp along" until the next exit? Is there an emergency lane? Can the vehicle make it to the emergency lane safely? What if the vehicle has to cross lanes of traffic to get to the emergency lane -- can it safely do so? Some of these are difficult judgment calls.

Solving classification problems that are complex and that involve making careful judgments are the strengths of expert systems and KBSs. The problem of detecting malfunctions that originate on-board the vehicle may not be sufficiently complex so as to require a KBS. However, the problem of detecting malfunctions in nearby vehicles is quite complex and may well require a KBS approach. This is examined further in section 4.3 below.

Improving malfunction detection and responses using machine learning

When detecting a malfunction or when determining the best response to one, it is essential for the system to operate very quickly. Learning methods can speed up these classification procedures. One idea is to examine the use of decision tree induction to minimize the number of variables that must be examined, to identify the best order for examining these variables, and to minimize the complexity of expressions involving these variables. This learning process is performed off line; it does not operate on-board the vehicle. The input to the learning program is the set of variables that are considered to be relevant plus a large number of example scenarios (probably generated by simulations). The output is a decision tree that will guide the implementation of the malfunction management program such that it operates more efficiently and quickly.

Automated diagnosis of malfunctions

Automated diagnosis techniques can complement the malfunction management effort described above. There is a large body of results on automated diagnosis in the AI literature where the computer uses a model of, in this case, the mechanics of the vehicle plus data about its current operating status, and determines any existing malfunctions, much as a human diagnostician would. In a sense, these systems generate their own malfunction classification schemes automatically. Their strength is that they can implement malfunction detection quite readily and can be updated easily if the vehicle's mechanics change.

Automated synthesis of responses to malfunctions

Synthesis is the creation of a new solution to a given problem at hand. There are many techniques for automated synthesis, particularly in the literature on AI planning. Some of these techniques can be expanded and adapted to apply to the AHS domain. Planning techniques are quite powerful; they would allow a vehicle to create its own set of maneuvers to handle the particular situation faced by the vehicle. However, planning techniques work best when there is

a considerable amount of time to respond;. Clearly this will not be the case in general for AHS, so careful evaluation will be needed here.

5.2 Management of the Vehicle in Traffic

This refers to managing the vehicle's traffic/roadway level operations such as lane changes, entries/exits, merges, splits, obstacle avoidance, and emergency maneuvers. We have referred previously to this function as *high level control*. It is essentially the systems management and decision analysis functions for the operation of the vehicle with respect to the roadway and other traffic. It also includes adaptive control in that it would set parameters of underlying control systems -- e.g., it might set the desired headway for the longitudinal control system.

Situation assessment using classification

In order to insure safe operation, the vehicle will need to assess its situation with respect to surrounding traffic in an ongoing manner. For example, assume that the vehicle is traveling in a platoon with a second platoon just to its left in another lane. The car immediately to its left begins to move laterally to the right. Should the vehicle be prepared to detect and respond to this? We think that the answer is "yes". But the vehicle needs to make an assessment of the intent of the other car and of its ability to operate properly. If indeed the other car is about to swerve, then it suffers from some sort of malfunction, and our vehicle must determine that fact and take appropriate avoidance action, such as slowing down immediately.

Situation assessment is another classification problem. Unfortunately, this problem is complicated because the actions of other vehicles depends not only upon the physics of the situation, but upon the intent of the vehicles involved. We expect that KBSs have much to offer to this problem.

Situation assessment using plan recognition

If we only take the approach described just above for the problem of situation assessment, then all of the possible scenarios that a vehicle might encounter must be identified and programmed in advance. Unfortunately, when we take the intent of other vehicles/drivers into account, then the variety of traffic situations is enormous and new situations are likely to be encountered. If we use predetermined classification rules, then the vehicle might assess a new situation incorrectly. A more robust approach is to give the vehicle a flexible capability to recognize the intent of other vehicles. Such a capability is called *plan recognition* in the AI literature. The basic idea is that the vehicle uses the data it collects about the actions of another vehicle, and infers to the best of its ability the intention of that vehicle/driver. Moreover, this is done from a knowledge base of vehicle maneuvers: the knowledge base might contain a set of maneuvers and the different ways to effect them. The knowledge base would also contain descriptions of vehicle behaviors that correspond to malfunctions. From that knowledge, and from the behaviors it witnesses, it tries to infer the maneuver (or malfunction) of the vehicle (e.g., changing lanes, swerving out of control, etc.).

For example, if a vehicle called V starts drifting left towards another lane, then V might be changing lanes, or it might have been blown by a gust of wind, or it might be losing lateral control. If there is a safe gap in traffic for vehicle V to change lanes, then it is likely that it intended to change lanes. If there is another vehicle in the lane to the left of V, then maybe something forced vehicle to swerve. Was there a recent gust of wind, and can the vehicle detect this? Was there a vehicle to the right of V that swerved, causing V to swerve in response? Was

there no reason for V to swerve, indicating a malfunction? Each of these situations might require a different response and so the vehicle should try to detect these differences.

Controlling the vehicle at the traffic level

The normal operation of the vehicle requires a module that makes decisions about when and how to change lanes, exit, join platoons, etc. First and foremost, this module is responsible for ensuring safe operation. Thus, if situation assessment reports an obstacle in the path of the vehicle, this module would immediately initiate an obstacle avoidance maneuver. If it appears that another car is about to swerve into the vehicle, then this module would immediately initiate an emergency avoidance maneuver, such as slowing down rapidly. Second, under normal operations, this module has the responsibility of route planning and of carrying out those plans. While the overall route plan of each vehicle is determined in cooperation with a centralized traffic manager, those route plans will necessarily be somewhat vague and the vehicle will be responsible for refining the plan and for carrying it out. For example, the central traffic manager may instruct the vehicle to change lanes to the left, but only the vehicle can determine exactly when to initiate the actual maneuver and control the progress of the maneuver.

These two responsibilities -- responding to emergencies and performing normal traffic maneuvers -- require different types of system support. The former can be viewed as a classification problem and would thus be similar to the malfunction management unit. The difference here is that the emergency arises from the environment -- other vehicles, obstacles, etc. -- instead of an on-board malfunction. The latter requires the ability to synthesize plans to some extent (route planning is essentially a synthesis activity), to refine plans and to carry them out. There is a vast amount of literature about using KBSs for these types of problems.

Improving management of the vehicle in traffic using machine learning

Learning methods can assist in this area in a number of ways. The most interesting method is to use *reinforcement learning* techniques to try to anticipate and avoid dangerous situations before they occur. For example, here is a two-level approach. The reinforcement learning module is at the upper level and an *artificial neural network* or similar classifier is at the lower level. The artificial neural network is used to classify different traffic situations. Overall this is a very difficult problem because of the potentially enormous number of variables involved. The output of the artificial neural network would be used by the reinforcement learning program to classify similar situations into states. The reinforcement learning program would also be able to model the world using physical laws and vehicle dynamics. This would have two benefits. First, it would be able to evaluate some states directly as being dangerous and estimate the cost and probability of potential damage. Second, it often takes a long time for a reinforcement learner to propagate its values backward and assign values to states and actions leading to unsafe states. If it has a world model, it can speed up this process by looking ahead to determine the best action in each state, and assigning the value of the subsequent state (minus any cost for taking the action) to the current state. Another possibility for speeding up learning is to replay past experiences, since it may take repeated runs through the same situation for values to be assigned to previous states.

5.3 Traffic Management

Traffic management has the responsibility of coordinating traffic over a regional highway system. This would include not only automated highways but the manual roads and highways as well. However, the task of traffic management for the automated roads is more challenging given the goals of high throughput and safety.

There are many functions within the traffic management task, and it is not yet clear which will be assigned to each vehicle, which to centralized units in the infrastructure, and which should be handled in a distributed fashion. In fact, it may be that these assignments change dynamically depending on the current situation. We predict that the AI technologies of search techniques, knowledge based scheduling and machine learning will be extremely helpful in implementing these functions.

Traffic management function assignment

The main function of traffic management is to prevent congestion and to ensure free flowing traffic whenever possible while minimizing the time of and distance traveled by each vehicle. For the automated highway, this can be managed in several ways. One approach is that proposed by PATH where the infrastructure's traffic management determines which cars are allowed in every lane in addition to the speed of each car/platoon. It controls the former by controlling when cars can change lanes. It controls the latter by posting speed requirements. It also performs route planning for individual vehicles; a vehicle requests a point to point route plan and the centralized traffic manager provides the route. These routes would be selected so as to maximize overall traffic flow while not placing undue time or distance burdens on individual cars.

However, this design might lead to high volume data management requirements. In particular, does the centralized manager need to maintain information on the current location and travel plan of every vehicle on the automated highway? Can a reasonable traffic manager be designed that does not have this burden but which still attains the primary goal of free flowing traffic? We think that the answer is “yes”, whether it uses AI-based or more traditional technologies.

Knowledge-based scheduling for traffic management

In any scheme, there will be a need for route planning for a large number of vehicles. This is essentially a synthesis and scheduling problem of a large magnitude. Finding optimal schedules so that all vehicles reach their destinations with an overall minimum travel time and distance is clearly intractable. Thus, sub-optimal solutions must be found. The field of operations research has much to offer in this area. But AI, particularly with the strengths of search techniques and knowledge-based scheduling, also has very strong techniques to offer.

Machine learning methods for traffic management

Genetic algorithms might also be very helpful to traffic management. Genetic algorithms have been shown to be effective in optimizing flows in other situations, and are probably an appropriate technique to use in a situation where local changes can have wide-spread effects. Genetic algorithms use operations analogous to mutation and recombination to modify non-optimal solutions to problems in an attempt to improve them. The idea of recombination allows it to search a very large space of solutions very quickly while minimizing the chances of getting stuck in a local minimum -- i.e., a region of the search space where no good solution exists.

6.0 INVESTIGATION OF TWO OPPORTUNITIES

For two of the opportunities listed in section 4, we have performed a detailed investigation.

- Section 5.2: Controlling the vehicle at the traffic level.
- Section 5.3: Improving management of the vehicle in traffic using machine learning.

Our goal was to elaborate the above modules to a limited extent so that the interested reader could see some of the detail needed to develop KBSs and learning modules. Moreover, our work might provide the seeds for a larger effort in this area. The Tufts effort was fairly resource limited, and so our intent was to provide a good sized sample rather than a fully developed module.

Both of the above efforts depended upon a micro-level traffic simulator, which we also developed as part of this effort. The simulator, called TAHSK (for **T**ufts **A**utomated **H**ighway **S**imulator **K**it, and pronounced "task"), is discussed in section 5.1, and is followed by the two subsections detailing the two efforts. Before beginning, we explain the overall structure of the software, as depicted in figure 3.

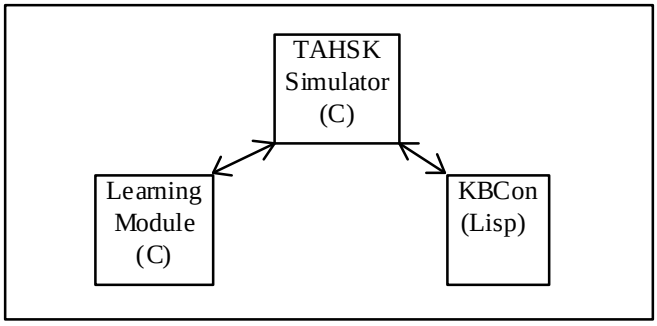

Figure 3. The three primary modules of our effort.

The TAHSK simulator is a discrete time micro-level traffic simulator with a graphic display of the roadway and vehicles, and is written in the C language using the public domain Tcl/Tk interface to the X Window System. The simulator can run without the other two modules, though its control of vehicles is rather simple. The learning module, also written in C, relies upon the simulator to provide feedback to its learning algorithms. It learns to control a simulated vehicle in a variety of traffic situations. The KBCon module is tightly integrated with the simulator and serves as the knowledge based high level vehicle controller for each simulated vehicle. When the simulator is run in the mode that uses KBCon, the simulator pauses at the end of each simulated time point (i.e., at the end of each *tick*), sends the current traffic data to KBCon and waits for KBCon to pass back high level instructions for each vehicle. These instructions set headway, control lane changes, etc. The connection between KBCon and the simulator is a TCP/IP network connection, and thus the two modules could be run on any two machines on the Internet. KBCon is written in Allegro Common Lisp, from Franz Inc., and uses

the free Loom knowledge representation system [27, 28, 29], licensed from the Information Sciences Institute of the University of Southern California. At this time, the learning module and KBCon are not connected, though that is an obvious area for future work. All of our software is copyrighted and available at no cost, but with some restrictions, via any of the tools that access the world-wide web, such as Mosaic. To obtain this software, begin at the home page for the Dept. of Electrical Engineering and Computer Science at Tufts University, http://www.cs.tufts.edu, and look under the listings for Prof. James G. Schmolze.

6.1 The TAHSK Simulator for AHS

We have designed and written a discrete time micro-level traffic simulator, TAHSK, for conducting AHS research. TAHSK uses some of the ideas from Reece's PHAROS simulator [38], but omits much of the complexity that is not needed for the current research. TAHSK is written in the C programming language using the Tcl/Tk toolkit, so it is portable to any machine that supports X windows. It contains a TCP/IP network interface to the Lisp programming language for knowledge based vehicle control. When TAHSK is started, it pops up an X window of a size specified on the command line, as shown in figure 4. The roadway is shown, in this case a three lane highway with barriers on either side and dashed lines between the lanes. Vehicles appear as moving rectangles. This window can be scrolled using the scrollbars to view any part of the roadway. The control bar at the top of the window contains buttons and fields that allow the user to stop and start the simulator (clicking and re-clicking on "Frz" -- for "freeze"), to step through the simulation by one tick at a time (the "Step" button), to have the simulator freeze if a crash occurs ("Frz on crash"), to zoom in or out (set the "Scale" factor), and to adjust the ratio of simulated time to real time (set the "Time ratio" factor). Clicking on a vehicle with the left mouse button allows the user to view the data structures associated with that vehicle. The right mouse button allows the user limited control of the selected vehicle, including lane changes, swerves, acceleration and deceleration.

Figure 4. The X window view of the TAHSK simulator

Roadways for TAHSK are specified by an input language which allows multiple links, lanes, and segments. Within a link, consecutive segments of different lengths may be specified, but all lanes have the same width. At this point, all segments are straight, but curved segments could be implemented if needed. Within a segment, lanes may have different speeds and lane markings. Lanes may be designated as *sources*, in which case vehicles are created at random at a specified rate and time headway at the beginning of the lane. Lanes may also be specified as nonexistent, which forces a vehicle to make a lane change before reaching that segment. This can be used to specify entry and exit lanes which do not exist on every segment.

The main loop in TAHSK calls a high-level, mid-level, and low-level control function for each vehicle. The low-level controller handles the basic physics of the vehicle, calculating where it will be at the next tick based on its current acceleration, velocity, and position. The mid-level controller performs both the lateral and longitudinal control functions, thereby performing lane keeping, handling lane changes and maintaining headways. The high-level controller determines how much headway should be maintained, who to maintain headway on, when to stay in lane, when to change lanes, etc. This modular design makes it easy to create a mix of zombie vehicles that operate in prespecified ways plus intelligent vehicles controlled by KBCon plus learning vehicles controlled by the learning module.

6.2 KBCon: A Knowledge Based High Level Vehicle Controller

KBCon is a knowledge based high level controller for fully automated intelligent vehicles (IV) in an AHS. KBCon manages the high level operations of the IV, such as getting the IV to its intended destination, changing lanes, staying in lane, and exiting. It relies upon several onboard subsystems for controlling lateral and longitudinal positioning, communications, sensing, and other functions. It also interfaces with controllers that may be operating in the roadside infrastructure. The goal of KBCon is not only to manage the normal high level operation of the automated vehicle but also to manage responses to situations arising from malfunctions -- malfunctions that occur either onboard the vehicle or in nearby vehicles. Thus, KBCon continually monitors its own status while simultaneously attempting to identify the intentions

and operational status of nearby vehicles so as to recognize possible threats and to respond to them.
Our approach has been to use AI technology sparingly, preferring to rely upon more traditional engineering approaches whenever possible. Our reasons for this are simple: most AI technologies are difficult to verify and validate to the extent necessary for safe highway travel. One function that eschews traditional engineering approaches is that of high level control, as described above, and to which we have taken a knowledge-based approach.

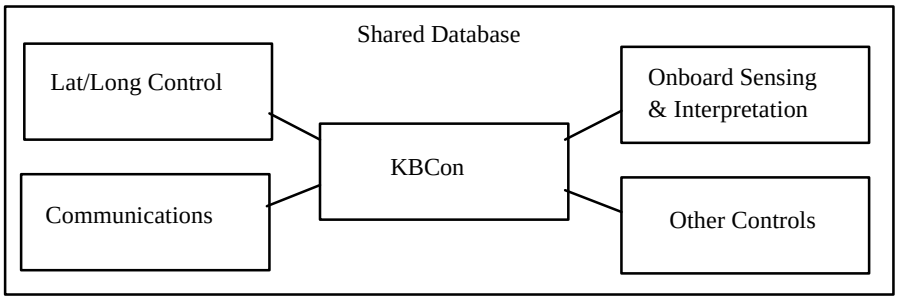

Figure 5. The modules that KBCon depends upon

Figure 5 shows the more important subsystems that KBCon depends upon. Lat/long control performs lane keeping and headway control. Communications controls vehicle-to-vehicle and vehicle-to-roadway message traffic. Onboard sensing controls various types of instrumentation to detect vehicles and obstacles. Other controls include functions such as headlight operation, etc.

The function of KBCon is to control these subsystems so that the vehicle reaches its destination safely and in a timely manner. To do this, KBCon performs three primary sub-functions. The first is to establish and maintain a plan of action. The plan describes the actions needed for the IV to reach its destination. The plan typically is not complete at any given time. For example, the plan might include an action to change lanes into the appropriate exit lane, but it typically does not say exactly when to do so until the vehicle is close to the exit. The plan is updated and modified as needed. For example, another vehicle might swerve into the IV's lane, which will probably lead to an immediate change in plans. Also, the plan is represented at various levels of abstraction. An example of plan abstraction occurs when changing lanes; the IV should communicate to nearby vehicles that it will change lanes, but the method of communication may be left abstract, so that the manner of communication can be selected when it is actually needed (options might include radio communication, infrared, or visual signals). The planning component of KBCon relies upon techniques from AI planning. While these techniques usually require considerable time to execute, the size of the search space for this application is small -- there are only a small number of maneuvers available -- and so we are able to perform this planning in a reasonable amount of time.

The second sub-function of KBCon is to assess the intentions and operational status of nearby vehicles and to predict the future motion of nearby obstacles, if any. For example, if the vehicle immediately to the left of the IV is moving laterally to the right side of its lane, then the IV must determine whether or not this is normal lateral drift or an unexpected and dangerous swerve. Moreover, if this type of swerving occurs frequently for a given vehicle, then that vehicle is determined to be malfunctioning and appropriate actions must be taken. These include reporting

the vehicle and avoiding it. For this first effort, we are using rule based classification techniques to assess other vehicles and objects. In the future, we hope to investigate the use of AI plan recognition techniques.

The third sub-function of KBCon is to assess the current situation, which depends upon the operational status of the IV , the assessments of nearby vehicles and obstacles, static information about the roadway layout, plus information and restrictions that come from a roadway controller. This assessment leads to one of four basic outcomes. (A) The current plan is being followed and there are no reasons to change the plan. (B) Some adjustment to a subsystem, such as the lat/long controller, is needed in order to follow the current plan but there is no reason to change the plan. (C) There is a need to fill in more details in the current plan. (D) There is a need to change the current plan. An example of B occurs when a lane change is just about to finish and KBCon instructs the lat/long controller to stop changing lanes and, instead, to follow in the current lane. An example of C occurs when the current plan calls for exiting at a particular place, but there is no specific time to begin the exit maneuver, and the exit is sufficiently close that a specific time must be set. An example of D occurs when a car in an adjacent lane suddenly and unexpectedly moves into the IV's lane -- an emergency maneuver is needed, such as slowing down or performing an emergency lane change. The methods used here are a mixture of rule based techniques that watch for certain conditions to arise along with plan monitoring techniques.

A diagram showing the different modules of KBCon along with brief descriptions of the data that flows between them is shown in figure 6.

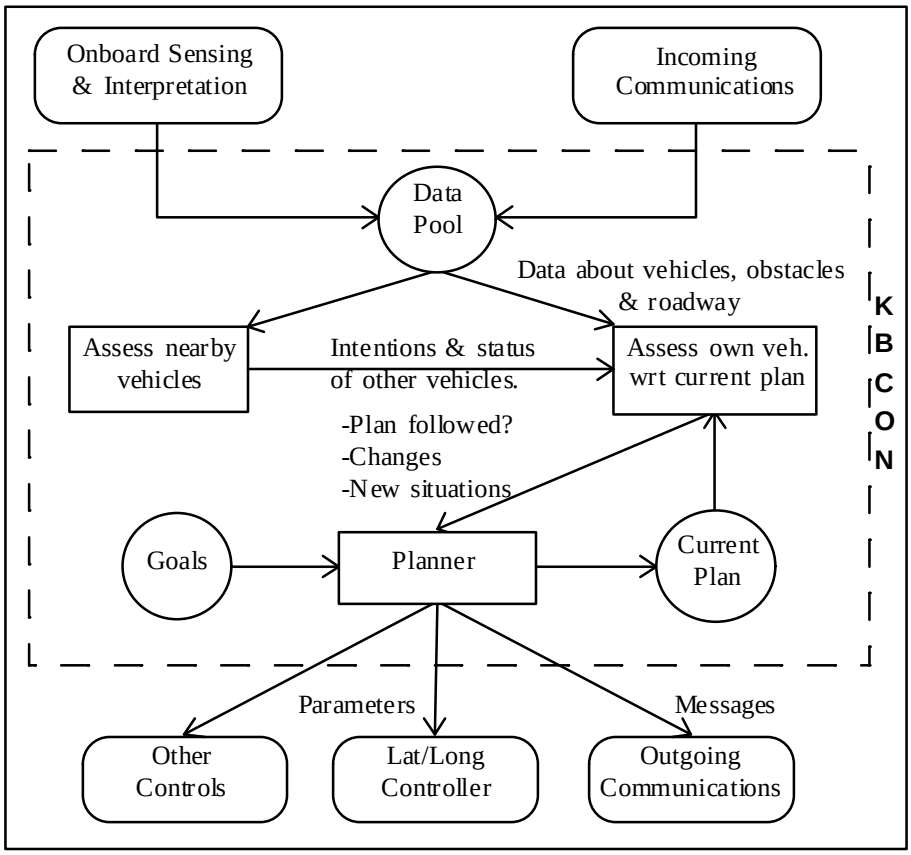

Figure 6. The basic modules of KBCon

We are assuming the following. All vehicles on the highway are IVs. Also, each IV has good estimates of the positions and velocities of nearby cars and obstacles as well as the layout of the nearby roadway. In future efforts, we will investigate relaxations of these assumptions.

As explained earlier, a prototype of KBCon has been implemented that is tightly integrated with the TAHSK simulator. The simulator runs as a process separate from KBCon. This simulator code, written in C, simulates the roadway and each vehicle. The only component missing from the simulation is the high level control function, which is supplied by the KBCon module. KBCon operates as a separate process -- in theory, one KBCon process for each IV. KBCon is written in Common Lisp and uses the Loom knowledge representation system, developed by the Information Sciences Institute of the University of Southern California. KBCon and the simulator communicate using TCP/IP network connections. The information sent to KBCon is that which we can reasonably expect the underlying subsystems to determine, such as the estimated position and velocity of nearby vehicles. In turn, KBCon sends back commands to the underlying subsystems, which are implemented in the simulator.

The remainder of this subsection is devoted to some of the details of KBCon. We begin with an overview of the knowledge representation scheme used, followed by a discussion of the functionality implied by figure 6.

Knowledge Representation Scheme used in KBCon

We use the Loom [27, 28, 29] knowledge representation (KR) system in KBCon. Loom is a frame-based terminological KR system. There is a fairly rich language for defining types, an assertion language for describing particular situations, plus three programming tools. The first tool is a production rule system (for forward chaining rules), a backward chaining rule system, plus an object-oriented programming tool. The Loom system distinguishes *definitions* from *assertions* about particular situations. For definitions, there is a the type language that allows the definition of *concepts*, each of which denotes a set, and *relations*. In frame systems, these are called, respectively, *frames* and *slots*. While the language for defining types is fairly rich, we have only used a few of its features, primarily that of subsumption. One term subsumes another if and only if the set denoted by the first term is a superset of that which is denoted by the second. For example, the concept for MAMMAL subsumes the concept for PERSON since all persons are mammals. The type language essentially allows on to create a domain specific language, which in our case includes concepts such as AHS-Vehicle. A fraction of the AHS type hierarchy is shown in figure 7.

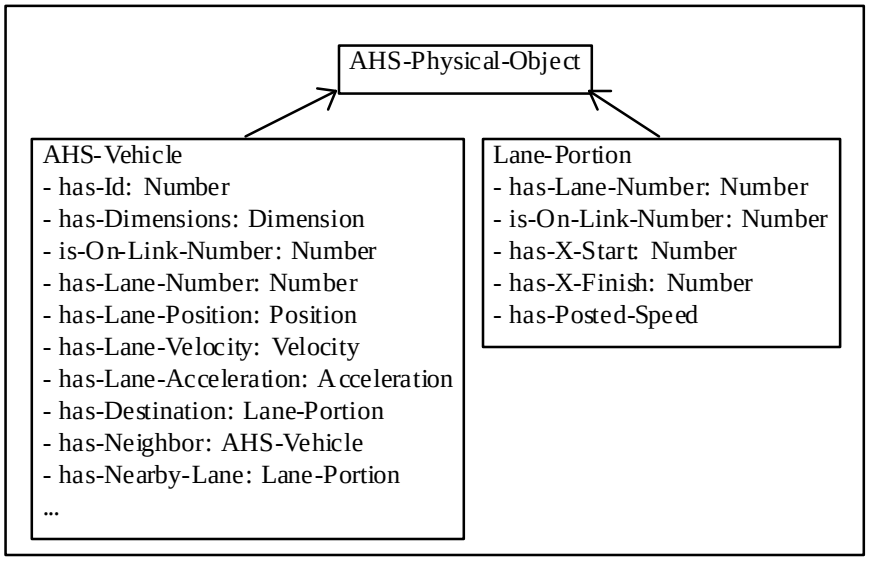

Figure 7. A Hierarchy of some of the types used by KBCon

Here, we have a high level concept named AHS-Physical-Object that represents all the physical objects in the AHS domain, including vehicles, roadways, and such. Throughout, we have tried to choose names that are meaningful, and we often include hyphens in the names. By using hyphens in Lisp, such as with "AHS-Vehicle", we can create a single name that is comprised of several words, leading to a more meaningful name. Also, the names of concepts have all words capitalized. Names for relations have all words but the first one capitalized. A specific type of AHS-Physical-Object is AHS-Vehicle, which is a concept representing all AHS vehicles. With each such vehicle, we associate certain other information, such as the vehicle's id number, the vehicle's dimensions, the id of the link it is on, the number of the lane it is currently in on that link, etc. The items listed below AHS-Vehicle in figure 7 represent the relations relevant to each AHS vehicle. has-Id is a relation that represents the id number of each AHS vehicle, and that id must be a Number, where Number is the concept representing all numbers. has-Dimensions denotes the dimensions of the vehicle -- i.e., its width and length. The Dimension concept is not shown here, but it has two relations, namely, has-Width and has-Length. has-Lane-Number denotes the number of lane the vehicle occupies. has-Lane-Position is the position of the vehicle relative to the reference frame of the link. The Position concept is not shown here; it is a vector that represents the X and Y coordinates of the origin of the vehicle. has-Lane-Velocity and has-Lane-Acceleration are similar. The relation has-Destination represents the given destination of the vehicle. It is a Lane-Portion, which is also shown in the figure. A Lane-Portion represents a portion of a lane: it has lane and link id numbers, positions as to where the portion starts and stops, plus the posted speed on the portion. For a destination, the Lane-Portion is typically the exit lane that the vehicle seeks to use, where the has-X-Start is the position on the link where the exit lane starts, has-X-Finish is the position where one can no longer change into it (i.e., where the exit lane barriers start), and has-Posted-Speed is the posted speed limit for the exit ramp. Each AHS vehicle also has a number of neighbors, represented with the relation has-Neighbor. A neighbor is another AHS vehicle in the vicinity of the original vehicle. A basic assumption we have made is that each AHS vehicle has relatively accurate position and velocity information for all neighbors, where neighbor is a parameter of the system, and which is currently set to be any vehicle within 500 meters. We point out that a given vehicle, which we refer to as the *own*

vehicle, has only limited information about neighbor vehicles, namely, it has estimates of its position, velocity and acceleration, plus it knows its width and length. In particular, the own vehicle does not have access to the plans of neighbors, and so it must infer the intended actions of these neighbors. has-Nearby-Lane denotes the Lane-Portion information of the lanes near to the vehicle. We assume that each vehicle has such roadway information available. Finally, there are numerous other relations concerning each vehicle -- we do not have space to list them all and will explain them as they arise in our later discussions.

Assertions in Loom are used to assert information about that current situation using domain specific types. There are three types of assertions in Loom: one can *create* individuals, one can *assert* information about those individuals, and one can *retract* information. For example, we can create an individual AHS-Vehicle within Loom, and can give it the name, say, V24 (i.e., this is the 24-th such individual vehicle).

```
(create  V24  AHS-Vehicle)
```

The above Lisp code creates the individual, asserts that it is of type AHS-Vehicle, and gives it the name V24.[1] The name given to an individual is simply a handle used to find it; the name carries no meaning of its own, and so we have chosen a very simple naming scheme for individuals. (During the actual use of the KBCon system, the names of individuals are never used -- we present them here primarily for pedagogical purposes.) We can assert information about V24 via the Loom function *tell*. Here we tell Loom that the vehicle id number of V24 is 6374.

```
(tell  (has-Id  V24  6374))
```

In order to tell Loom about the position of V24, we must first create a Position individual, tell Loom that this individual is the position of V24, and then assert the data regarding the Position.

```
(create  P57  Position)

(tell  (has-Position  V24  P57)
     (has-X  P57  (meters  507.4))
     (has-Y  P57  (meters  1.9)))
```

P57 is the name of this particular Position individual. Note that we have made several assertions in one call to tell. The first assertion states that the has-Position (this is the relation that relates each vehicle to its position) of V24 is P57. We then fill in the data for P57 by giving it an X coordinate, via the relation has-X, of 507.4 meters, and a has-Y of 1.9 meters.[2]

---

[1] For those readers who are familiar with Lisp, the “create” function evaluates its arguments, and so the call would actually be (create ‘V24 ‘AHS-Vehicle).

[2] Again, for Lisp programmers, we have taken another liberty with ““tell”. “tell” is a macro, and does not evaluate its arguments, so no quotes are needed. As such, the call to the meters function is also not evaluated. Loom allows a limited form of evaluation within a call to “tell”, namly via the use of Loom variables, which are symbols beginning with a “?”. These are the only quantities evaluated with a tell or forget macro call. Thus, this call to tell would actually look as follows.

There are a number of reference frames used in the simulator and KBCon. Relevant to this discussion are the reference frames for lanes and vehicles. The coordinates of each vehicle are relative to the reference frame for the lane that the vehicle is in. Each vehicle is "in" only one lane at a time, though its body may spill over into another lane. The origin of a lane is the center of the lane, beginning at the start of the link that the lane occupies. As you face down the roadway, positive X is in front of you and positive Y is to your left. A vehicle's reference frame is similarly directed, though its origin is at the rear, center of the vehicle.

The third type of assertion in Loom is the forget operation, which retracts an earlier assertion. For example, if the above vehicle, V24, moves in the next tick, KBCon does the following (remember, the position of V24 is stored in P57).

```
(forget  (has-X  P57  (meters  507.4))
         (has-Y  P57  (meters  1.9)))
(tell  (has-X  P57  (meters  508))
       (has-Y  P57  (meters  1.85)))
```

The "forget" operation forgets the earlier position information and the "tell" asserts new position information. The fact that P57 is the has-Position for V24 is still asserted.
There are numerous Loom functions to retrieve information from the knowledge base. Space does not permit our reviewing them here.

Our treatment of time is based on the situation calculus [32] and is similar to that found in STRIPS [11], namely, (nearly) the entire knowledge base of assertions in Loom is intended to be true of the current simulated time point. Therefore, time is not explicitly represented. In the future, KBCon will become more sophisticated, and an explicit representation for time will be introduced. Also, we rely the closed world assumption, in particular, negation as failure. In other words, explicit assertions stored in the knowledge base are assumed true. If an assertion is not explicitly stored in the knowledge base, then it is assumed to be false.

Loom supports contexts. A context is a "place" to store information. Contexts can be completely separate, so that assertions in one context are completely unknown in another. Alternatively, contexts can be structured in a parent/child relation -- assertions in the parent are automatically asserted in the child as well but assertion in the child are unknown in the parent. We use a single Lisp/Loom Unix process to perform the processing for all vehicles in the simulator. However, each vehicle has its own context, so no assertions are shared between vehicles.

Onboard Sensing, Interpretation and Communications
Each time we go to a new simulated time point -- we call this a *tick* -- a fairly large amount of information is retracted and new information asserted. At each tick, the simulator sends several TCP/IP messages to KBCon. The first such message informs KBCon about any new vehicles that were created during the previous tick, and it passes along information about the vehicle that

---

```
(let  ((?x  (meters  507.4))  (?y  (meters 1.9)))
   (tell  (has-Position  V24  P57)  (has-X  P57  ?x)  (has-Y  P57  ?y)))
```

does not change, such as its size and destination. KBCon creates a new context for each new vehicle and asserts this unchanging information.

The next message identifies vehicles that were *sunk* in the last tick -- to sink a vehicle is to remove it from simulation (typically, a vehicle is sunk when it drives off of the end of the simulated roadway). For these vehicles, all information is removed by destroying that vehicle's context.
Next, KBCon receives a message describing the current status of each vehicle being simulated. This includes the vehicle's position, velocity and acceleration, plus any other relevant data that changes on each tick. Our assumption is that each vehicle has an accurate model of its own position, velocity, and acceleration. Moreover, we assume that each vehicle has a fairly accurate model of this data for nearby vehicles, which we call *neighbors*. This data is meant to be the result of onboard sensing and interpretation plus communication, and is considered by the Ford/Raytheon team to be a reasonable set of assumptions. In addition, certain information is computed from this new data and asserted in Loom. Finally, some Loom assertions are forgotten -- these correspond to information that must be computed fresh for each tick since they depend upon time varying data. Thus, for this type of computed information, we retract the information at the beginning of each tick and re-compute it. Information of this type will be identified as it is introduced.

We note that simulated inter-vehicle messages are supported by KBCon.

Controlling the Lat/Long Controller
The lat/long controller (a combined lateral and longitudinal controller) is simulated in TAHSK. Its interface for a given vehicle is as follows.

- Target speed: The controller tries to maintain this speed, subject to headway constraints.
- Target lane number: The lane number that the vehicle is headed for (when lane keeping, this is simply the current lane).
- Offset within lane: The Y distance from the lane center about which the vehicle should be centered. Normally, this is zero.
- For each lane: a flag, a vehicle, and a headway time: The flag indicates that headway should be maintained for that lane. The vehicle is the one to keep headway on, and the time is the time headway to keep.
- Emergency flag: If false, which is normal, then the vehicle will only use comfortable accelerations (but will use any deceleration that is necessary for safety, though trying to use a comfortable one when possible). If set, then it will use any possible acceleration as well.

Thus, the controller can keep headway for up to N vehicles when there are N lanes, one vehicle per lane. This is very important when changing lanes, or when vehicles in nearby lanes are acting dangerously. Headway information is gathered by querying Loom for all *n* that match (keep-Headway-On *v n*) where *v* is the own vehicle. This is explained below.

Assessing Neighbor Vehicles
The primary purpose of assessing neighbor vehicles is to identify threats to the safety of the own vehicle. Thus, this module implements a type of *defensive driving*. If all operations are normal, this module is not needed. However, not everything will be normal all the time. There are two types of identification made in this module. The first identifies neighbors that are already in or

may soon be in the same lane as the own vehicle. We use the relation may-Occupy-Lane-Number to identify a lane that a given vehicle is either currently in or, in the judgment of KBCon, may soon be in. For example, if vehicle V24 is in lane 2 of the current link, then:

(tell (may-Occupy-Lane-Number V24 2))

is asserted. If V24 is not in lane 2 but it is determined that it may soon be in that lane, the above is also asserted. This type of determination is made, for example, if a neighbor has indicated that it will soon change lanes, or if it appears to be swerving into a neighboring lane. These assertions are made for the own car as well. Note that assertions regarding may-Occupy-Lane-Number are computed for each tick, and so all such assertions are retracted at the start of each tick.
Here is a rule that makes this determination trivially.

IF (AHS-Vehicle ?v) and (has-Lane-Number ?v ?l)
THEN (tell (may-Occupy-Lane-Number ?v ?l))

In this and all rules, variables begin with a "?". Thus, ?v and ?l can match anything in the knowledge base. Here (AHS-Vehicle ?v) will match against any assertion of an individual being an AHS-Vehicle. Similarly, (has-Lane-Number ?v ?l) matches against has-Lane-Number assertions that use the same ?v. The effect of this rule is to note that an AHS-Vehicle may occupy the lane that it is already in. As we said, this is trivial, but also necessary for our algorithm. Here is an example. Let us assume that our knowledge base had only the following four assertions.

(AHS-Vehicle V24)	(AHS-Vehicle V43)
(has-Lane-Number V24 2)	(has-Lane-Number V43 1)

The above rule would match two times. The first against the two facts on the left, in which case ?v becomes V24 and ?l becomes 2. The second match is against the two facts on the right, with ?v = V43 and ?l = 1. After the rule executes for both matches, the knowledge base would have two additional assertions, making the entire set of assertions as follows.

(AHS-Vehicle V24)	(AHS-Vehicle V43)
(has-Lane-Number V24 2)	(has-Lane-Number V43 1)
(may-Occupy-Lane-Number V24 2)	(may-Occupy-Lane-Number V43 1)

An assertion is made after we have received a message that a vehicle intends to change lanes. Recall that when such a message is received, we associate with the neighbor the following fact:

(has-Target-Lane-Number V43 2)

which asserts that V43 intends to change to lane 2. The following rule relies upon this. It simply states that if a vehicle intends to change into lane ?l, then it may soon occupy ?l.

IF (AHS-Vehicle ?v) and (has-Target-Lane-Number ?v ?l)
THEN (tell (may-Occupy-Lane-Number ?v ?l))

The final rule we will show here is one that attempts to detect a possible malfunction in a vehicle. If a vehicle is failing to keep to the center of its lane by a certain amount, and if it is not moving back to the center, then assume it is changing lanes. This is a conservative assumption, and one that will be refined in future versions of KBCon. Below is a rule that checks for a vehicle moving left.

IF (AHS-Vehicle ?v) and	*?v is an AHS vehicle.*
(has-Lane-Number ?v ?l) and	*?v is in lane ?l.*
(has-Position ?v ?pos) and	*?pos is the position vecotr for ?v.*
(has-Y ?pos ?Y) and	*?Y is the Y position of the center of ?v wrt the lane's center.*
(has-Velocity ?v ?vel) and	*?vel is the velocity vector for ?v.*
(has-Y ?vel ?Vy) and	*?Vy is the y component of the velocity of ?v.*
(has-Dimensions ?v ?dim) and	*?dim is the dimension vector for ?v.*
(has-Width ?dim ?Wv) and	*?Wv is the width of ?v.*
(is-On-Link ?v ?link) and	*?link is the link that ?v is on.*
(has-Lane-Width ?link ?Wl) and	*?Wl is the width of each lane on the ?v's link.*
?Vy _ 0 and	*?v is either moving left or not shifting left nor right.*
?Y > ((?Wl - ?Wv) / 2)* 0.8	*See below.*
THEN (tell (may-Occupy-Lane-Number ?v (?l + 1)))	

This is quite a long rule, but it is not complicated. The first 11 lines merely retrieve data, of which primarily ?Y, ?Vy, ?Wv and ?Wl are of interest. The second-to-last clause in the IF part makes sure that the vehicle is either moving left or is not moving laterally (if it is moving right, we ignore it for now). The last clause compares ?Y against a computed value. Remember that ?Y is the y-distance from the lane's center to vehicle's center. Now, given the width of the lane and the vehicle, the distance to the left of ?v when ?v is dead center in the lane is (?Wl - ?Wv) / 2. If ?Y is off center by more than 0.8 of that amount, then we determine that ?v is moving left. Of course, the use of 0.8 is a parameter of the overall system. This criteria is a bit simplistic and should be determined by a more thorough study. However, our purpose here is to demonstrate the capabilities of KBSs, and so we have chosen simple criteria when possible.

Note that, though this rule may seem overly long, it has the advantage of being entirely self-contained. Except for basic data about the vehicles and roadway, it does not depend upon other calculations. Moreover, its execution can be made quite efficient. A similar rule is added for shifts to the right.

The second type of identification concerns what to do about vehicles that may be in the same lane as the own vehicle. These vehicles fall into three categories: those we ignore, those we keep headway on, and those we avoid because we are in imminent danger of colliding with them. For those neighbors that we ignore, we simply do nothing. For those neighbors ?n of the own vehicle ?v that we keep headway on, we assert (keep-Headway-On ?v ?n). For those neighbors ?n with whom we might collide, we assert (may-Collide-With ?v ?n). We examine these two in reverse order.

If a neighbor may overlap lanes with the own vehicle and is too close, we determine that they may collide. For simplicity, we determine this by checking whether or not the X distance between their origins is less than a fixed distance away, which we have set to 25 meters. Again, this is a criteria that should be more complex, such as being sensitive to speed, vehicle length and plan of action, and more thoroughly studied.

IF(AHS-Vehicle ?v) and *?v is an AHS vehicle.*
(has-Neighbor ?v ?n) and *?v has a neighbor vehicle ?n.*
(may-Occupy-Lane-Number ?v ?l) and
(may-Occupy-Lane-Number ?n ?l) and *?v and ?n may soon occupy the same lane ?l.*
(has-Position ?v ?posv) and *?\posv is the position vecotr for ?v.*
(has-Position ?n ?posn) and *?posn is the position vecotr for ?n*
(has-X ?posv ?Xv) and *?Xv is the X position of the rear of ?v .*
(has-X ?posn ?Xn) and *?Xn is the X position of the rear of ?n.*
25 meters _ (?Xn - ?Xv) _ 25 meters *?n is within 25 meters of ?v (by comparing origins).*

THEN (tell (may-Collide-With ?v ?n))

Remember that there may be more than one lane that the own vehicle or neighbor vehicles *may* occupy. Thus, we examine the relative distance between vehicles that might both occupy the same lane. If we determine that the own vehicle may collide with another vehicle, then a more serious examination of the current plan of action is initiated, which may lead to a change in plan. We examine this in the next section.

Hopefully, the incidence of near collisions will be small. More commonly, there will be vehicles that the own vehicle must simply keep away from by maintaining a certain headway. In KBCon, we keep headway on neighbors that may overlap soon be in the same lane as the own vehicle's and are more than 25 meters in front but less than 500 meters away. If a neighbor is more than 500 meters away, it is too far away to worry about. Again, the distances used here, 25 and 500 meters, should be carefully chosen.

IF (AHS-Vehicle ?v) and *?v is an AHS vehicle.*
(has-Neighbor ?v ?n) and *?v has a neighbor vehicle ?n.*
(may-Occupy-Lane-Number ?v ?l) and
(may-Occupy-Lane-Number ?n ?l) and *?v and ?n may soon occupy the same lane ?l.*
(has-Position ?v ?posv) and *?\posv is the position vecotr for ?v.*
(has-Position ?n ?posn) and *?posn is the position vecotr for ?n*
(has-X ?posv ?Xv) and *?Xv is the X position of the rear of ?v .*
(has-X ?posn ?Xn) and *?Xvnis the X position of the rear of ?vn.*
?Xn - ?Xv > 25 meters and *?n is at least 25 meters ahead of ?v (by comparing origins)*
?Xn - ?Xv < 500 meters *If ?n is more than 500 meters away, ignore it.*

THEN (tell (keep-Headway-On ?v ?n))

The assertion of (keep-Headway-On ?v ?n) makes sure that ?v will stay behind ?n, as explained earlier when the interface to the lat/long controller was described. However, this rule makes these headway assertions for too many vehicles. For example, if there are two vehicles in front of the own vehicle, then we need to keep headway only on the rear one. The following rule removes these unnecessary headway responsibilities by forgetting those for which there is a vehicle behind that is also under headway control. Remember that the lat/long controller allows up to one vehicle per lane to keep headway on.

IF (keep-Headway-On ?v ?n1) and
(keep-Headway-On ?v ?n1) and
(has-Lane-Number ?n1 ?l) and
(has-Lane-Number ?n2 ?l) and *?v is keeping headway on ?n1 & ?n2, both in lane ?l.*
(has-Position ?n1 ?posn1) and
(has-Position ?n2 ?posn2) and
(has-X ?posn1 ?Xn1) and
(has-X ?posn2 ?Xn2) and
?Xn1 < ?Xn2 *?n1 is further back than ?n2.*

THEN (forget (keep-Headway-On ?v ?n2)) *Forget about keeping headway on ?n2.*

Processes and Plans

A process is an activity that occurs over time. Our representation for plans is based on processes, and on a process formation language. An example of a *primitive* process is Lat-Long-Keep-In-Lane. This process keeps the own vehicle in its current lane while maintaining appropriate headway and checking for dangerous situations (e.g., collisions). This process is *primitive* in that it is not defined in terms of other processes -- it is implemented directly in software. Note that this process is actually fairly simple because the real work of maintaining headway and lane keeping is performed by the lat/long controller. Thus, the Lat-Long-Keep-In-Lane merely prepares the proper instructions to be sent to the lat/long controller based on the headway determination described above.

Lat-Long-Change-Lanes is another primitive process that also transmits the headway information while instructing the lat/long controller to move into the target lane. Of course, changing lanes is considerably more complex that simply moving left or right; we will soon discuss how this is performed using *complex* and *abstract* processes.

An *abstract* process is one that has no direct implementation. Instead, it is a process that can be realized by a number of other processes. For example, when a vehicle is about to change lanes, it communicates that fact to all neighbor vehicles. There is an abstract process, Communicate-Changing-Lanes, that performs this. One decomposition of this process -- i.e., one way to accomplish it -- is to use the Broadcast-Changing-Lanes. This latter process is a primitive process that uses inter-vehicle radio communication to broadcast the lane change. However, the Broadcast-Changing-Lanes process has a prerequisite, namely, that the radio transmitter is operational. If it is not, then that decomposition cannot be used, and an alternative is used.

Though we have not implemented alternatives as yet, one might rely upon a visual signal (the equivalent of a signal blinker) to signal a lane change or infrared communications. In any case, one aspect of planning is to select the best decomposition for any abstract process to be executed.

A *complex* process is one that is implemented in terms of other processes using a process formation language. This differs from an abstract process in that there are no alternatives. It is more like a subroutine. The process formation language is as follows.

<process> ::= <named-process> |
(sequence <process$_1$> ... <process$_n$>) |
(parallel <process$_1$> ... <process$_n$>)

A <named-process> is simply a process that has already been defined. It can be primitive, complex or abstract. The sequence construct executes each of the processes in turn and in order. It completes when the last process completes. The parallel construct begins all the processes immediately and at the same time. It finishes when all have finished.

Here is the process to change lanes. We use the def-process form, which is part of KBCon and which defines types of AHS processes.

```
(def-process Change-Lanes (has-Target-Lane-Number) complex
  (sequence (parallel (Lat-Long-Keep-In-Lane)
                      (sequence (Find-Gap has-Target-Lane-Number is-Changing-Lanes-
Behind)
                                (Communicate-Changing-Lanes has-Target-Lane-Number
                                                            is-Changing-Lanes-Behind)))
            (Lat-Long-Change-Lanes has-Target-Lane-Number is-Changing-Lanes-Behind)))
```

When one creates a Change-Lanes process, one gives it the number of the target lane (i.e., has-Target-Lane-Number). The process of changing lanes has two steps. The first one finds the gap and communicates with neighbors. The second one, at the end of the form, performs the actual changing of lanes. The parallel form is used because we still want to maintain lane keeping while we seek a gap using the Find-Gap process. Thus, we execute in parallel the primitive process Lat-Long-Keep-In-Lane while performing the nested sequence. This sequence first finds a gap in which to change lanes, which is indicated by setting the parameter is-Changing-Lanes-Behind (i.e., the vehicle to change behind, which is set to a special value if the target lane is clear), and then broadcasts the vehicle's intentions via Communicate-Changing-Lanes. Note that nearly all use of parameters is as input parameters. However, the Find-Gap process treats is-Changing-Lanes-Behind as an output parameter. This parameter mechanism is actually a constraint satisfaction mechanism (e.g., [12]) that allows constraints to be posted on or between variables at any time. In this case, the constraint is that the vehicle that the own vehicle will pass behind must be in the proper lane, be at an appropriate distance away, etc. The identification of individuals that satisfy constraints is another important part of the planning process.

When a vehicle is first created, Loom is informed of its destination. Loom immediately lays out a high level plan to get to that destination. This usually involves a small number of lane changes and appears like a long sequence of processes (e.g. , stay in lane 2, change to lane 1, and then

exit into the desired exit lane). However, the exact times at which to change lanes and the details of those changes are not determined until the actions are imminent. Moreover, the plan may change due to instructions from a central traffic controller (not implemented yet in KBCon), or a possible collision, such a having a vehicle stray into the own vehicle's lane. When anything occurs to cause a change in plans, KBCon replans. At this time, KBCon's planning ability is fairly simple. The primary tool is search with simulation -- we heuristically search all possible plans and evaluate them using simulation. The first plan that meets the goals of no collision and reaching the destination is selected. For now, this works because the search space is small. More realistic simulations will lead us to use more sophisticated planning techniques.

Evaluation

The resources given to this project were quite limited, and so most of our effort regarding KBCon was spent in design and implementation. Most of the software is operational at this time, and we will continue to work on it. However, extensive operational tests with the simulator have not been performed. Our intent is for KBCon to respond effectively to all normal traffic situations and to a large number of abnormal ones, arising from malfunctions either in the own vehicle or other vehicles.

6.3 Learning to Improve the Management of the Vehicle in Traffic

We have set up a reinforcement learning system using a three-layer artificial neural network (ANN) as its evaluation function approximator. This learning system is integrated into a version of the TAHSK module that controls the motion of the vehicles. One vehicle is designated the *learner* and the remaining vehicles are designated *zombies.*

The output of the ANN is a number giving the evaluation of the proposed action in the current situation. The network has 56 inputs, as follows:

- -- 6 inputs give the lateral and longitudinal position, velocity, and acceleration of the learner vehicle
- -- 48 inputs give the positions, velocities, and accelerations of eight nearby vehicles, relative to the learner vehicle
- -- 2 inputs give the proposed action, specified as lateral and longitudinal accelerations

The learning system has a number of parameters and submodules as follows:

- -- The number of neurons in the ANN's hidden layer. Increasing this number allows the ANN to learn more complicated functions, but may decrease its ability to generalize.
- -- The backpropagation learning rate for the ANN, between 0 and 1. A higher rate speeds up learning, at the risk of learning too quickly from atypical examples or failing to adjust the weights to the best possible values.
- -- The reinforcement learning rate, between 0 and 1. A similar quantity for the reinforcement learning algorithm.
- -- The reinforcement discount rate, between 0 and 1. The amount by which future rewards are discounted. A zero value indicates that only the current reward is important, whereas a one indicates that all future rewards are weighted equally with the current reward.

-- A parameter which controls the probability with which the learner will take the best possible action. At the initial stages of learning the evaluation function is not expected to be very good, so selecting suboptimal actions allows the learner to explore a wide variety of possibilities. As learning progresses, the degree of randomness should be reduced so that the learner will only explore actions which are close to the best.
-- A parameter which controls the amount of randomness added to position measurements. This is done to simulate real sensors with limited accuracy.
-- The ANN is written as a function which takes the 56 inputs mentioned above and produces a real-valued output. This function is simple to modify (such as adding a fourth layer or changing the transfer function) or replace with another type of network (such as a CMAC [2]).
-- The reward function is written as a subroutine which takes the same 56 inputs as the ANN and returns a real-valued reward.
-- The mid and high level controllers are written as functions that can be different for each vehicle. This permits zombies that have differing amounts of randomness in their actions and that attempt to maintain differing headways. Zombie controllers can easily be programmed to simulate a variety of malfunctions.

A run of the reinforcement learning system consists of a number of trials, each of which ends when the learner reaches the end of the roadway or crashes into another vehicle or barrier. At the beginning of a run, the ANN is initialized either to all zeroes, to random weights, or to the result of a previous run. At the beginning of each trial, a number of zombie vehicles are created, then a learner is created, followed by more zombies. Each vehicle is created at a random time, with velocity equal to the free-flow velocity of its lane, and with a safe headway from the vehicle in front of it. At each time step each vehicle's controller adjusts its acceleration. The learner's controller uses the acceleration selected by the reinforcement learning system using the evaluation function implemented by the ANN. The weights of the ANN are then adjusted according to the results of that action.

At any time, the person running the learning system has the option of saving the weights which describe the ANN into a file. This file can be used to initialize a learner for a future run, or the network can be tested in a nonlearning setting to see how it responds to different types of zombie vehicles.

7.0 SUMMARY AND CONCLUSIONS

Our primary **conclusions** are that AI technologies, particularly knowledge based systems and learning methods, can make strong contributions to AHS, especially in the following four areas.

- *Management of malfunctions onboard the vehicle*:
- *Management of each automated vehicle in traffic*.
- *Overall traffic management*.
- *Sensor interpretation and fusion*.

Our primary **recommendations** are as follows.

- We should incorporate the management of malfunctions, both onboard and in other vehicles, into *normal* operations as much as possible.
- We should design each AHS vehicle to have effective predetermined responses to a large number of traffic situations but
- We should also design each AHS vehicle to be able to respond effectively to new situations.

We also **recommend** the following.

- Each AHS vehicle have the capability to plan its own maneuvers and execute them.
- Further investigate the following areas, particularly with regard to AHS applications: KB planning (to handle new situations), learning methods (to handle new situations and to improve responses to already known situations), and expert systems (for overall high level control).

In addition, we have investigated two major applications of Artificial Intelligence techniques for an Automated Highway System. The first is the use of a Knowledge Based Systems approach in an operating AHS to ensure that it has the flexibility to respond to unexpected situations. Under normal operation a vehicle might not need to call on these capabilities, just as under normal highway driving conditions a human driver operates "on automatic". Due to the complexity and diversity of an AHS system, a vehicle must be prepared to take reasonable actions when normal conditions do not hold, just as a human driver must be prepared to take appropriate actions if the vehicle in front slows down abruptly, etc. We have demonstrated a protoype KBS for an AHS, but there is still much further to go to ensure that such a system could operate quickly enough in a sufficiently wide variety of situations.

The second application is the use of Machine Learning techniques to help design controllers for an AHS. An AHS will operate at higher speeds and with smaller headways than those to which human drivers are accustomed, so it is desirable that ML techniques be used to explore the best way to control an automated vehicle. We have programmed a prototype reinforcement learning system for an AHS environment, using an artificial neural network to evaluate proposed actions. This system operates at close to real time on a Sun workstation, but may take many trials to learn reasonable behaviors. Although the speed of the network was not a problem, it may be that a CMAC network would exhibit better generalization. More work needs to be done to explore the applicability of these techniques to designing high-level controllers.

8.0 REFERENCE

1. Agre, P.E. and Chapman, D. Pengi: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87),* Seattle, WA, July 1987, pp. 268-272.

2. Albus, J.S. *Brain, Behavior, and Robotics,* Byte Books, Peterborough, NH (1981).

3. Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. Purposive Behavior Acquisition on a Real Robot by Vision-Based Reinforcement Learning. In *Proceedings of the MLC-COLT Workshop on Robot Learning,* Rutgers•University, New Brunswick, NJ, July 1994, pp. 1-10.

4. Baroglio, C., Giordana, A., and Piola, R. Learning Control Functions for Industrial Robots. In *Proceedings of the MLC-COLT Workshop on Robot Learning*, Rutgers•University, New Brunswick, NJ, July 1994, pp. 11-20.

5. Brooks, R.A. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation RA*, 2 (1986), 14-23.

6. Dickerson, J.A. and Kosko, B. Ellipsoidal Learning and Fuzzy Throttle Control for Platoons of Smart Cars. In *Fuzzy Sets, Neural Networks, and Soft Computing*. Von Nostrand, Yager, R.R. and Zadeh, L., 1993.

7. Dickerson, J.A. and Kosko, B. Fuzzy Functions Approximation with Supervised Ellipsoidal Learning. In *World Congress on Neural Networks (WCNN'93), Volume II*, 1993, pp. 9-17.

8. Dickerson, J.A., M., K.H., and Kosko, B. Hybrid Ellipsoidal and Fuzzy Control for Platoons of Smart Cars. In *Proceedings of the Third International Conference on Industrial Fuzzy Control and Intelligent Systems (IFIS '93)*, December 1993.

9. Dorigo, M. and Colombetti, M. The Role of the Trainer in Reinforcement Learning. In *Proceedings of the MLC-COLT Workshop on Robot Learning*, Rutgers•University, New Brunswick, NJ, July 1994, pp. 37-46.

10. Fausett, L. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ (1994).

11. Fikes, R. and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*(1971), 189-208.

12. Freuder, E. Exploiting structure in constraint satisfaction problems. In *Constraint Programming*. Springer-Verlag, Mayoh, B., To appear., 1994.

13. Giarratano, J. and Riley, G. *Expert Systems: Principles and Programming*, PWS-KENT, Boston, MA (1989).

14. Gilmore, J.F. and Elibiary, K.J. AI in Advanced Traffic Management Systems. In *Proceedings of AAAI-93 Workshop on AI in Intelligent Vehicle Highway Systems*, Washington, D.C., July 1993, pp. 57-65.

15. Ginsberg, M. *Essentials of Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA (1993).

16. Gomi, T. and Laurence, J.C. Behavior-Based AI Techniques for Vehicle Control. In *Proceedings of Vehicle Navigation and Information Systems (VNIS'93)*, Ottawa, Canada, October 1993.

17. Gomi, T. and Volpe, P. Collision Avoidance using Behavior-Based AI Techniques. In *Proceedings of Intelligent Vehicles Symposium (IV'93)*, Tokyo, Japan, July 1993.

18. Hsu, A., Eskafi, F., Sachs, S., and Varaiya, P., "The Design of Platoon Maneuver Protocols for IVHS," California PATH, no. UCB-ITS-PRR-91-6, University of California at Berkeley, April 1991.

19. Ingrand, F.F., Georgeff, M.P., and Rao, A.S. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert 7*, 6 (December 1992), 34-44.

20. Kao, S.M., Laffey, T.J., Schmidt, J.L., Read, J.Y., and Dunham, L. Real Time Analysis of Telemetry Data. In *Proceedings of the Third Annual Expert Systems in Government Conference*, Washington DC, 1987, pp. 137-144.

21. Koller, D., Weber, J., and Malik, J., "Robust Multiple Car Tracking with Occlusion Reasoning," University of California at Berkeley, no. UCB-ITS-PWP-93-36, PATH Working Paper, 1993.

22. Korf, R.E. Real-Time Heuristic Search. *Artificial Intelligence 42*, 2-3 (1990), 197-221.

23. Kosko, B. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, NJ (1992).

24. Kuwata, Y., Hart, D.M., and Cohen, P.R. Steering Traffic Networks. In *Proceedings of AAAI-93 Workshop on AI in Intelligent Vehicle Highway Systems*, Washington, D.C., July 1993, pp. 51-56.

25. Laffey, T.J., Cox, P.A., Schmidt, J.L., Kao, S.M., and Read, J.Y. Real-Time Knowledge-Based Systems. *AI Magazine 9*, 1 (Spring 1988), 27-45.

26. Lyons, D.M. and Hendriks, A.J. A Practical Approach to Integrating Reaction and Deliberation. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS-92)*, College Park, MD, June 1992, pp. 153-162.

27. MacGregor, R.M. A Deductive Pattern Matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, Minn., August 1988, pp. 403-408.

28. MacGregor, R. The Evolving Technology of Classification-based Knowledge Representation Systems. In *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan-Kaufman, Sowa, J., San Mateo, CA, In press., 1991.

29. MacGregor, R. and Burstein, M.H. Using a Description Classifier to Enhance Knowledge Representation. *IEEE Expert 6*, 3 (June 1991), 41-46.

30. Maclin, R. and Shavlik, J. Incorporating Advice into Agents that Learn from Reinforcements. In *Proceedings of the MLC-COLT Workshop on Robot Learning*, Rutgers•University, New Brunswick, NJ, July 1994, pp. 81-82.

31. McCallum, A. Reduced Training Time for Reinforcement Learning and Hidden State. In *Proceedings of the MLC-COLT Workshop on Robot Learning*, Rutgers•University, New Brunswick, NJ, July 1994, pp. 83-92.

32. McCarthy, J., "Situations, actions and causal laws," Stanford University, no. Stanford Artificial Intelligence Project: Memo 2, 1963.

33. McDermott, D., "Robot Planning," Yale University, no. YALEU/CSD/RR#861, August 1991.

34. Niehaus, A. and Stengel, R. An Expert System for Automated Highway Driving. In *Proceedings of the American Control Conference, Volume 1*, 1990, pp. 274-280.

35. Niehaus, A. and Stengel, R. Rule Based Guidance for Vehicle Highway Driving in the Presence of Uncertainty. In *Proceedings of the American Control Conference*, 1991, pp. 3119-3124.

36. Nilsson, N.J. Action Networks. In *From Formal Systems to Practical Systems*. Dept. of Computer Science, University of Rochester, Weber, J., Tenenberg, J., and Allen, J., pp. 21-52, October 1988.

37. Okuno, A., Kutami, A., and Fujita, K. Towards Autonomous Cruising on Highways. In *Automated Highway/Intelligent Vehicle Systems: Technology and Socioeconomic Aspects*, Society•of Automotive Engineers Pub. SP-833, 1990, pp. 7-16.

38. Reece, D.A., "Selective Perception for Robot Driving," Carnegie Mellon University, no. CMU-CS-92-139, Pittsburgh, PA, May 1992.

39. Sacerdoti, E.D. *A structure for plans and behavior*, Ph.D. dissertation, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1975.

40. Shladover, S.E., Desoer, C.A., Hedrick, J.K., Tomizuka, M., Walrand, J., Zhang, W.B., McMahon, D.H., Peng, H., Sheikholeslam, S., and McKeown, N. Automatic Vehicle Control Developments in the PATH Program. *IEEE Transactions on vehicular Technology 40*, 1 (February 1991), 114-130.

41. Stefik, M., “Planning with Constraints,” Stanford Computer Science Department, no. STAN-CS-80-784, 1980.

42. Stengel, R. and Niehaus, A. Intelligent Guidance for Headway and Lane Control. In *Proceedings of the American Control Conference,* Pittsburgh, PA, June 1989, pp. 307-313.

43. Sutton, R.S. The Challenge of Reinforcement Learning. In *Reinforcement Learning*. Kluwer Academic, Sutton, R.S., Norwell, MA, 1992.

44. Tadepalli, P. and Ok, D.K. Discounting Considered Harmful: A Comparison of Reinforcement Learning Methods for Automated Guided Vehicle Scheduling. In *Proceedings of the MLC-COLT Workshop on Robot Learning,* Rutgers•University, New Brunswick, NJ, July 1994, pp. 127-136.

45. Tham, C.K. and Prager, R.W. A Modular Q-Learning Architecture for Manipulator Task Decomposition. In *Proceedings of Eleventh International Conference on Machine Learning,* Rutgers•University, New Brunswick, NJ, July 1994, pp. 309-317.

46. Varaiya, P. Smart Cars on Smart Roads: Problems of Control. *IEEE Transactions on Automatic Control 38*, 2 (February 1993), 195-207.

47. Watkins, C.J.C.H. *Learning from Delayed Rewards*, Ph.D. dissertation, King's College, Cambridge, 1989.

48. Weber, J. and Malik, J., “Robust Computation of Optical Flow in a Multi-Scale Differential Framework,” University of California at Berkeley, no. UCB-ITS-PWP-93-4, PATH Working Paper, 1993.

49. Wellman, M.P. Transportation Applications of Artificial Intelligence (Extended Abstract). In *Proceedings of AAAI-93 Workshop on AI in Intelligent Vehicle Highway Systems*, Washington, D.C., July 1993, pp. 37-41.

50. Winston, P.H. *Artificial Intelligence*, Addison Wesley, Reading, MA (1984).