Situation Awareness for Tactical Driving

Rahul Sukthankar

January 27, 1997
CMU-RI-TR-97-08

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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

This research was partially supported by the U.S. Department of Transportation, under cooperative agreement *Automated Highway System*: DTFH61-94-X-00001.

Abstract

A primary challenge to creating an intelligent vehicle that can competently drive in traffic is the task of tactical reasoning: deciding which maneuvers to perform in a particular driving situation, in real-time, given incomplete information about the rapidly changing traffic configuration. Human expertise in tactical driving is attributed to *situation awareness*, a task-specific understanding of the dynamic entities in the environment, and their projected impact on the agent's actions.

In this thesis, I demonstrate how situation awareness may be used as a basis for tactical-level reasoning in intelligent vehicles. SAPIENT (Situation Awareness Planner Implementing Effective Navigation in Traffic) combines a knowledge of high-level driving goals with low-level reactive behavior to control vehicles in a custom tactical-level simulator, SHIVA. The simulated vehicles are based on the Carnegie Mellon Navlabs, sharing a common perception and control interface, allowing researchers to port systems from simulation to real life with minimal modification. The first implementation, MonoSAPIENT, uses explicitly encoded rules for competent driving, along with specialized algorithms for gap selection and lane changing to drive safely in the simulated world.

The second implementation, PolySAPIENT, is a distributed intelligence, built around the notion of *reasoning objects*, independent experts, each specializing in a single aspect of the driving domain. Each reasoning object is associated with an observed traffic entity, such as a nearby vehicle or an upcoming exit, and examines the projected interactions of that entity on the agent's proposed actions. Thus, a reasoning object associated with a vehicle is responsible for preventing collisions, while one associated with a desired exit recommends those actions that will help maneuver the vehicle to the exit. The results are expressed as votes and vetoes over a tactical *action space* of available maneuvers, and are used by a domain-independent arbiter to select the agent's next action. This loose coupling avoids the complex interactions common in traditional architectures, and also allows new reasoning objects to be easily added to an existing PolySAPIENT system.

I also introduce a new learning strategy, based on the PBIL evolutionary algorithm, that simultaneously optimizes internal parameters for multiple reasoning objects given a user-specified evaluation metric. This automated parameter exploration also enables rapid prototyping of new PolySAPIENT configurations.

Acknowledgements

Many thanks to my advisors, Chuck Thorpe and Dean Pomerleau, for guiding me through my Ph.D. research. From the initial experiments with RACCOON on the Navlab to the last-minute PolySAPIENT hacking, they provided invaluable insights into the theory, as well as practical advice on the implementation. Thanks also to the other members of my thesis committee: Joe Kearney, for valuable discussions and detailed feedback on the earlier draft of this dissertation; and Haris Koutsopoulos, for his perspective on how my work fits into the context of an intelligent transportation system.

John Hancock has collaborated with me on the SHIVA simulator for the last two years. His great ideas and solid code have helped make SHIVA into a useful research tool. Shumeet Baluja introduced learning to PolySAPIENT in the form of his elegant evolutionary algorithm, PBIL. John and Shumeet's harsh (yet constructive) feedback, given during numerous 3 a.m. debugging sessions helped this thesis greatly. Parag Batavia, Michelle Bayouth, Frank Dellaert, Dave Duggins, Jay Gowdy, Bala Kumar, Julio Rosenblatt, and Liang Zhao all gave useful suggestions for my research. Farokh Eskafi, Aleks Göllü, and others at UCB/PATH provided new perspectives and good ideas on simulation during my summer in Berkeley.

Thanks to Charalambos Athanassiou, Mei Chen, Ian Davis, Wes Huang, Dave LaRose, Dirk Langer, Rich Madison and other grad students in VASC for their help over the years. And thanks also to people in the SCS environment who kept things running: Marie Elm, Jim Moody, Marce Zaragoza, and Kris Hutchings (for fixing my Zeos Palmtop with a thumbtack).

My best friend, now my wife, Gita, participated in every aspect of this work. On the technical side, she wrote the Perl data manipulation programs, tamed my Linux machine, suggested research ideas, proofread several versions of this document, and generated the SGI videos for my thesis defense. She also lived my eccentric schedule, shared my dreams, and loved me unconditionally. It is good to be done.

Contents

Chapter 1

Introduction

Nowlan's Theory: He who hesitates is not only lost, but several miles from the next freeway exit [31].

Before the invention of the automobile, most forms of human transportation were, in some sense, intelligent. For example, a rider could rely on a horse's self-preservation instincts to avoid obstacles, or its sense of direction to find the way home [98]. Currently, the automobile possesses neither; a moment's inattention on a driver's part can cause the car to leave its lane, or crash into a nearby vehicle. Government studies attribute 96.2% of accidents in the U.S. to driver error [107]. A large fraction of these deaths could be prevented by the introduction of systems with the ability to make driving decisions in traffic, either to generate warnings for the human driver, or to control the vehicle un-

der autonomous control. These *intelligent vehicles* would recapture the "lost intelligence" in transportation systems.

1.1 Intelligent Vehicles

An intelligent vehicle is an automobile equipped with sensors, computers, and a control system. The sensors enable the vehicle to perceive the road, potential hazards, and other vehicles; the computers process this information and determine actions (such as steering or braking); the control system executes the chosen actions. In designs where the control system is not present (or is disengaged), the intelligent vehicle passively observes the traffic situation to issue warnings or recommendations for the human driver.

The idea of a car that drives itself is not new. Research in the 1960s focused on the problem of semi-autonomous vehicle control with promising results [35, 70, 19][1]. Significant progress was made when vision-based lane-trackers were integrated with vehicle control systems, enabling robot cars to drive on clear highways under controlled circumstances [23, 54, 73]. Simultaneously, research in automatic headway control [16, 22] and convoying [34, 55] led to vehicles capable of autonomous car following [52, 101]. In 1995, an intelligent vehicle, the Carnegie Mellon Navlab 5, steered 98% of the distance between Wash-

[1] Throughout this thesis, multiple references are listed in chronological order.

ington D.C. and San Diego (a distance of 2800 miles), demonstrating the maturity of this technology. One application of research in intelligent vehicles is an ambitious proposal known as the *Automated Highway System* — a concept where large numbers of intelligent vehicles could be used to improve highway capacities and reduce traffic accidents.

1.2 The Automated Highway System

The proposal for an Automated Highway System (AHS) consists of intelligent vehicles operating on a roadway that is free of pedestrians and traffic control devices [15, 65, 89, 91]. All vehicles enter the AHS under human control. Following a successful *check-in* procedure, the intelligent vehicles switch to computer control. While on the AHS, the automated vehicles travel at high speeds, possibly with reduced headways, until they reach their destination exit. At this point, the human driver regains control of the vehicle. Proponents of the Automated Highway System believe that such a roadway will have two important benefits (in addition to user convenience): 1) a reduction in traffic accidents; 2) an improvement in highway throughput. Both are briefly discussed below.

Computer-controlled vehicles, unlike human drivers, are not subject to boredom, fatigue, or distractions. Since single vehicle roadway de-

parture crashes (caused by driver inattention or impairment) account for almost 15000 deaths in the U.S. annually [115], even the introduction of a simple lane-keeping system (such as [74]) could reduce fatalities. Further improvements would require intelligent vehicles to understand the concept of *driving in traffic*: inhibiting unsafe lane changes, initiating swerving maneuvers (to avoid obstacles), and responding to tailgating. By combining an awareness of the traffic situation with super-human reflexes, it is conceivable that computer controlled vehicles will significantly improve safety on highways.

Intelligent vehicles could improve highway capacities in the following ways. First, intelligent vehicles could drive with reduced headways (in extreme cases, as small as one meter [110]), greatly increasing highway throughput. Second, the intelligent vehicles could help damp out traffic waves caused by abrupt acceleration and braking on the highway. Third, the intelligent vehicles could all be instructed to travel at a speed that is globally optimal for highway throughput.

Several competing designs (known as *AHS concepts*) promise these benefits to different degrees. Some advocate that all vehicles on the AHS be computer-controlled (dedicated-lane concepts) while others allow autonomous and human-controlled vehicles to share the highway (mixed-traffic concepts). In the former, the intelligent vehicles can rely on protocols established using communication with the infrastructure and other smart cars (as described in [110]) to execute lane-changing and exit maneuvers. In the latter, the intelligent vehicles must compe-

tently drive in uncontrolled traffic, much as human drivers do today.

Naturally, AHS concepts which require communication or substantial infrastructure modification cannot be deployed quickly on a large scale — particularly since the prohibition on human-controlled vehicles prevents the incremental conversion of existing highways into dedicated AHS roadways. Therefore, mixed traffic concepts must be investigated, not only as possible AHS goals, but as potential solutions to the dedicated concept deployment problem. Whether autonomous navigation in mixed traffic is possible is an open research question[2].

1.3 Driving in Traffic

Given that a robot van steered 98% of a cross-country highway journey autonomously, one could believe that the problem of driving in traffic had already been solved. This is not true for two reasons. First, it is important to realize that the intelligent vehicle was stable in the presence of other vehicles, but that it did *not* react to them. Thus, the Navlab did not pass slower traffic (unless directed by the human driver), nor did it change lanes in preparation for a chosen exit — the intelligent vehicle lacked any higher-level understanding of the traffic situation. Second, most highway drives have long periods of quiescence, interspersed

[2]Reddy [79] notes that the development of an intelligent vehicle capable of competent driving on real roads is one of the central problems in Artificial Intelligence today.

with brief bursts of intense activity (particularly near on-ramps or exits) where the driver changes lanes and speeds to achieve short-term goals. In the *No Hands Across America* trip, these maneuvers were performed by the human driver; the 98% statistic, which only considers raw distance, fails to capture this distinction.

Driving in traffic is very difficult for intelligent vehicles because good decisions need to be made given only incomplete information, in real time. Standard AI techniques such as search-based planning [30] are infeasible for several reasons. First, the effects of the robot's actions on the environment (especially other vehicles) cannot be determined. Second, most of these methods cannot function under noisy, uncertain conditions. Third, the state-space is extremely large if realistic maneuvers such as aborted lane changes are taken into account.

The scenario shown in Figure 1.1 illustrates some important elements of the problem of driving in traffic. Here, Car A, the vehicle of interest[3] , is following a slow-moving vehicle, Car B, in the right lane of a divided highway. Car A would like to overtake Car B, but would prefer not to miss its upcoming exit. The decision on whether or not to pass depends on a large number of factors including: the distance to the exit, the speeds of Cars A and B, Car A's preferred velocity, and the traffic in the area. Human drivers face such situations regularly and make good decisions with little conscious thought. Psychologists

[3]Throughout this thesis, the phrase "the intelligent vehicle" refers to the particular vehicle of interest (also shown consistently as Car A in scenario diagrams).

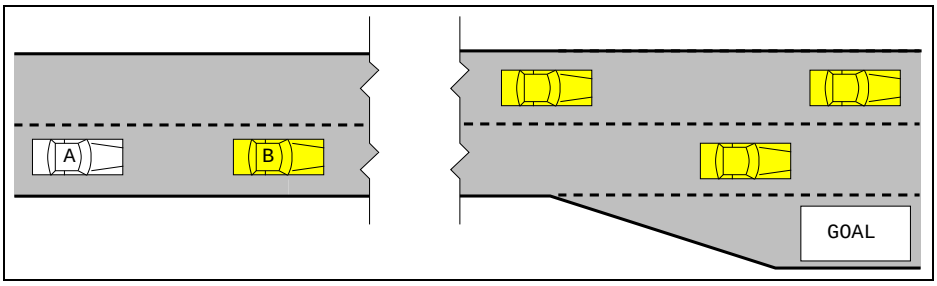

Figure 1.1: A scenario illustrating the problem of driving in traffic.

attribute this competence to a task-specific understanding of the situation, termed *situation awareness*. According to situation awareness theories, Car A's driver has been monitoring several entities in the traffic scene for some time (such as Car B, the desired exit, and vehicles in the passing lane), and has formed expectations on how the scenario will unfold over time [26]. This information enables the driver to select an appropriate maneuver. My thesis research develops cognition systems for intelligent vehicles that demonstrate a similar situation awareness.

The problem of navigating in traffic should be examined in the context of the entire driving task. In [67], driving is characterized as consisting of three levels: strategic, tactical and operational. At the highest (strategic) level, a route is planned and goals are determined. At the intermediate (tactical) level, maneuvers are selected to achieve short-term objectives — such as deciding whether to pass a blocking vehicle, as shown in Figure 1.1. At the lowest (operational) level, these maneuvers are translated into steering, throttle, and brake commands.

Mobile robot research has addressed these three levels to different degrees. Strategic-level route-guidance has been successfully tackled using standard AI search techniques on digital maps using GPS positioning information [24, 100, 83, 113]. Operational-level systems have become increasingly robust, allowing intelligent vehicles to be reliably controlled on real highways [106, 63, 36, 62]. However, the tactical level, critical to the deployment of intelligent vehicles, has not been rigorously addressed.

1.4 Thesis Overview

In this dissertation, I present solutions to the tactical driving task. My goal is to create cognitive modules which enable intelligent vehicles to drive competently in mixed-traffic highway situations by combining an understanding of high-level goals with low-level reactive behavior. Such systems:

- Assume the current state of sensing and computing technology
- Depend only on the existing roadway infrastructure
- Do not rely on communication with other vehicles on the highway

Chapter 2 presents an overview of current situation awareness (SA) theories. SA offers a cognitive model of the human real-time decision

making process. While SA research to-date has focused primarily on air combat applications, I illustrate how SA may be applied to tactical-level action selection for driving. I then explore how SA may be used as a basis for autonomous tactical driving systems for intelligent vehicles — and propose the design of such a novel system. SAPIENT (Situation Awareness Planner Implementing Effective Navigation in Traffic) is conceived as a tactical system interfacing with existing perception and control modules on the Carnegie Mellon Navlab [106] robots.

While SAPIENT is designed to be implemented on the Navlab, real traffic is (at least initially), an unsuitable testbed for several reasons. First, exploring new tactical algorithms in close proximity to other vehicles at high speeds is obviously unsafe. Second, the current configuration of car-sensing equipment on the Navlab does not provide adequate coverage of the areas beside and behind the robot. Finally, coordinating multiple, moving vehicles on the highway to create tactical scenarios is difficult. A natural solution is to develop reasoning systems in simulation. SHIVA (Simulated Highways for Intelligent Vehicle Algorithms), described in Chapter 3, is a traffic micro-simulator and design tool emphasizing tactical-level issues. SHIVA's simulated vehicles are functionally similar to the Navlabs and share a common control interface — allowing researchers to port algorithms from simulation to robot with minimal modification. Additionally, SHIVA offers specialized features for scenario creation, visualization, and interactive debugging of intelligent vehicle algorithms.

Chapter 4 describes the initial SAPIENT implementation, known as MonoSAPIENT. The system architecture is traditional: perception modules sense objects in the scene and update a global *world model* representing all of the relevant traffic entities; analytical algorithms such as *gap reasoning* manipulate this world model to derive desirable future positions for the intelligent vehicle; a decision tree selects the best action based on the current cognition state (e.g., "executing a left-lane change"); the chosen tactical action modifies the cognition state and generates operational-level commands; finally, these commands are sent to the robot's controller for execution. The competence of this implementation depends largely on the sophistication in the MonoSAPIENT decision tree. Unfortunately, the cognition states and interactions between the various rules explode exponentially in such monolithic systems as new functions are implemented [21]. Additionally, modifying the decision tree requires large numbers of non-local modifications — an error-prone and time-consuming task for researchers. To address these deficiencies, a distributed implementation of the reasoning system is proposed.

Chapter 5 details the distributed implementation of SAPIENT, termed PolySAPIENT. This architecture is built around the notion of *reasoning objects*, independent experts which monitors local aspects of the traffic scene. Each reasoning object is implemented using whichever AI techniques are natural for its task — for instance, some reasoning objects employ rule-based algorithms, others rely on potential field approaches, etc. The recommendations from each reasoning object are

expressed in a common language: as votes and vetoes distributed over a tactical *action space*. A knowledge-free arbiter examines the votes generated by each expert to determine the appropriate tactical-level response. This action, is translated into operational-level commands for execution by the intelligent vehicle controller. PolySAPIENT is successful because the tactical driving task can be decoupled into relatively independent subtasks (represented by the individual reasoning objects). Thus, adding functionality to the system only requires the implementation and addition of new reasoning objects — the existing objects and the arbiter do not need to be modified. While the distributed implementation of SAPIENT successfully addresses the criticisms raised in Chapter 4, it still leaves the problem of parameter optimization: the behaviors of individual reasoning objects depend upon a number of internal settings (such as thresholds or gains), and the votes from the different reasoning objects are scaled with external weights. Adjusting these parameters manually is a tedious task for humans, particularly when the meanings of these variables is not intuitive.

Chapter 6 discusses approaches for automatically tuning PolySAPIENT reasoning object parameters. Most conventional supervised learning techniques, such as neural networks trained using backpropagation [56], fail in this domain because tactical-level training signals are not available at each time step, but rather, only at the *end* of a scenario. I introduce a method relying on three components: 1) an evolutionary algorithm, termed PBIL (Population Based Incremental Learning) [14]; 2) a user-defined evaluation function which describes the de-

sired vehicle behavior; and, 3) a set of training scenarios generated in the SHIVA simulation environment. The learning progresses in an iterative manner summarized as follows. PBIL stochastically generates a population of candidate parameter values based upon its accumulated statistical understanding of the domain. These parameters are instantiated as PolySAPIENT reasoning objects for simulated vehicles, and tested over the set of SHIVA scenarios. The scores received by the parameter values are then used by PBIL to adjust its internal statistics — and this process is repeated. Over time, this technique generates high-scoring parameter values with a high probability. The tactical driving application challenges PBIL in a number of ways. First, since a vehicle's decisions depend on the behavior of other vehicles which are not under its control, the evaluation function is stochastic. Second, the PBIL algorithm is never exposed to all possible traffic situations. Third, since each evaluation takes considerable time to simulate, minimizing the total number of evaluations is important. The solutions to these problems are detailed in the chapter.

Chapter 6 also discusses a novel application for learning: as a tool to explore the viability of reasoning object configurations in PolySAPIENT. Researchers can quickly determine whether a particular set of reasoning objects will be able to drive successfully by seeing if PBIL is able to find high-scoring parameter settings. This rapid prototyping can also be applied to the debugging of reasoning object *representations*.

Throughout this thesis, I have used scenarios to illustrate tactical situations, train SAPIENT vehicles and evaluate potential solutions. Chapter 7 classifies tactical-level scenarios collected from real-life situations and examines the cognitive processing required to solve them. Scenarios have been successfully used in situation awareness literature to evaluate SA in human experts [25, 32, 90, 94, 86]. I extend this approach to assessing SA in tactical reasoning systems. Two types of experiments are proposed and performed: 1) a detailed examination of the decisions made by the intelligent vehicle over a set of *micro-scenarios*. 2) an observation of the behavior of large numbers of intelligent vehicles, driving using different tactical reasoning systems, on a *macro-scenario*.

Finally, Chapter 8 reviews the role of situation awareness in the tactical driving domain. The systems presented in the earlier chapters demonstrate that intelligent agents can exhibit competent tactical-level behavior. I briefly discuss the implications of intelligent vehicles in mixed traffic situations and present the contributions of this thesis in three main areas: artificial intelligence, machine learning, and simulation. This research can be extended in a number of interesting directions; I conclude with a brief look at some topics for future work.

Chapter 2

Human Driving Strategies

Let us not look back in anger or forward in fear,
but around in awareness.
—Unknown

Tactical-level driving is characterized by the constant battle between long-term goals and real-time constraints. Drivers must select appropriate maneuvers such as lane changing, accelerating, and car following given very little knowledge of the intentions of other drivers in their environment. In this fluid problem space, optimal solutions are rarely to be found, but the penalties for bad decisions are clear and severe. Unfortunately, safety cannot be guaranteed, even by conservative driv-

ing[1] — some level of risk-taking is unavoidable. Tactical driving thus forces a careful balance between competition and cooperation: aggressive maneuvering is successful, but not when it results in a crash.

The only examples of competent tactical driving to date are human drivers. This chapter examines models for human competence to see if similar methods may be used to create competent automated driving systems. The impatient reader is referred to Chapters 4 and 5, where these ideas are implemented.

2.1 Driver Models

Human driver modeling is an interdisciplinary endeavor involving a number of fields including robotics, psychology, control theory and statistics. In this section I examine models from some representative categories and focus on their relevance to tactical-level reasoning.

2.1.1 Task Models

Task models define the broad tasks involved in driving (e.g., car following), and decompose these tasks into detailed subtasks (e.g., headway

[1] A driver can be involved in collisions through no fault of his/hers; tailgating, reckless lane-changing, and animals jumping onto the highway can all precipitate a crash.

maintenance). McKnight and Adams [66] presented a comprehensive treatment of the situations and actions involved in driving. However, since their report was targeted towards human driver education, most of the recommendations, such as "[When passing, the driver] selects a lane relative to his car's speed, maneuvers, and traffic flow", are too vague to be directly used in tactical reasoning systems.

A second difficulty with most task models is that, like proverbs[2], the recommendations are often contradictory. As Reece [80] observes, the McKnight and Adams task list includes two subtasks that instruct drivers to "observe pedestrians and playing children" and to "ignore activity on the sidewalk that has no impact on driving" without providing insights as to which sidewalk activities have no impact on driving. Since these discriminating between these situations requires "common sense", encoding this knowledge in the form of driving rules for a reasoning system is challenging.

Task models are nevertheless useful in this thesis research for two reasons. First, they highlight aspects of the tactical driving task that should be need to be addressed by an intelligent vehicle. Second, they provide insights about mapping observable phenomena into specific conditions (e.g., the driver should initiate an overtaking maneuver in response to a slower vehicle ahead).

[2]For example, "haste makes waste", and "a stitch in time saves nine", prescribe opposite responses in similar situations.

2.1.2 Risk Models

Risk models for driving have emerged from psychological research in the area of perceived risk. By combining the decision theoretic notions of expected utility and the willingness of humans to take "acceptable risks", these models attempt to explain commonly observed phenomena such as speeding, aggressive driving styles and intoxicated driving. Although robot vehicles are not likely to engage in such behavior, intelligent systems demonstrating situation awareness may require sophisticated models of human drivers. Utility functions based on perceived risk (such as time-to-impact measures) can also be used by reasoning systems to select tactical-level maneuvers (See Chapter 5). A representative example of a risk model is Wilde's Risk Homeostasis Theory:

> Risk Homeostasis Theory maintains that, in any activity, people accept a certain level of subjectively estimated risk to their health, safety, and other things they value, in exchange for the benefits they hope to receive from that activity [120].

Counterintuitively, risk homeostasis theory predicts that humans adjust their behavior so as to *maintain* (rather than minimize) their perceived risk at a constant *set-point risk* level:

> The degree of risk-taking behavior and the magnitude of loss ... are maintained over time, unless there is a change in the target level of risk [120].

A case study, known as the Munich Taxicab Experiment [7] was conducted to test the implications of risk homeostasis theory under controlled conditions. Some vehicles in a taxi fleet were equipped with an anti-lock braking system (ABS) that allowed drivers to maintain steering control during hard braking on slippery roads. Conventional wisdom predicted that the ABS-equipped vehicles would be safer than unequipped vehicles. Surprisingly, the results showed that: [7, 120]:

- Among the accidents involving the company's taxis, there was no statistically significant difference between the involvement rate of the two vehicle types (in fact, the ABS vehicles were involved in slightly *more* accidents).
- Accident severity was independent of the presence or absence of ABS in the taxi.
- Accelerometers installed in the taxis measured more extreme decelerations (associated with hard braking) in vehicles equipped with ABS.
- Drivers in ABS cabs made sharper turns in curves, were less accurate in lane-keeping behavior, maintained shorter headway distances, made poorer merge maneuvers and created more "traffic conflicts". All of these differences were statistically significant.

Thus risk homeostasis theory, as supported by such experiments, has pessimistic predictions for any attempt to improve highway safety solely

through technology. However, in the context of tactical-level reasoning, risk homeostasis theory provides support for utility-based approaches to situation awareness (See Chapter 5).

2.1.3 Perceptual and Motivational Models

Perception models have been used to describe driver behavior in accidents [109], suggest methods for safer driving [117, 118] and motivate new collision warning devices [8]. In the tactical driving domain, perceptual models are particularly relevant in two areas: sensor modeling and driver intentions.

Sensor modeling at the operational level is primarily concerned with tracking objects and segmentation (low-level actions which humans typically take for granted). At the tactical level, the focus shifts to reasoning about object-to-lane mapping, blind spots and occlusions — tasks which human drivers perform more consciously. Unsurprisingly, novice drivers are less adept at these higher-level tasks: for example, inexperienced drivers are likely to forget about vehicles which are not currently visible [118]. Perceptual models also lead to heuristics for safer driving which can be exploited by both humans and intelligent vehicles (e.g., “At night, don’t over-drive the range of your headlights”). The perceptual motivation for positional maneuvering is further discussed in Section 2.2.3.

In partially automated systems, the intelligent vehicle must be sensitive to its driver's intentions. Perceptual models can be used to gain some insights into this area: recent research [71] shows that drivers' eye fixation patterns are strongly correlated with their current mental state. In Pentland and Liu's system, the driver's internal state is modeled as a four-state Hidden Markov Model (HMM). Once the HMM has been trained, the system is able to predict when the driver is about to brake or turn. This knowledge may then be used by the intelligent vehicle to optimize its behavior for the expected maneuver — in some sense, the situation awareness is shared over the driver-vehicle system.

The notion of a driver's internal state is central to motivational models. In this framework, perceptual information is integrated with discrete mental states in an attempt to predict the actions that a human driver would take in that given situation [108]. This description of human cognitive activity can also involve aspects from utility theory — generally in the form of a "perceived risk" factor. Although some effort have been made to specify motivational models in a symbolic programming language [1], no successful implementation currently exists. In MonoSAPIENT (See Chapter 4), discrete internal states, analogous to mental states here, are used to differentiate the different modes of driving (e.g., car following vs lane-changing behavior).

2.1.4 Control Models

Control models for drivers are primarily important when modeling operational level phenomena. For example, the well-known "Two-Second Rule" for car following [92] is based on the observation that humans require approximately 1.75 seconds to identify and react to a potentially dangerous situation [66]. Lane-keeping and steering models such as pure-pursuit tracking [114] are valuable at the tactical-level. First, such models can help the intelligent vehicle predict the future likely positions of observed vehicles. Second, such models can allow the reasoning system to estimate the time needed to execute a given maneuver (such as a lane change). Control models can also be applied to plan recognition — for example, a Hidden Markov Model (HMM) could be used to predict the "internal states" of the other vehicles in the area.

Other control models have been developed in the traffic simulation domain. The ones of most interest to tactical driving research are those which model lane-changing [116, 2], car following [124], and emergency maneuvers [3]. Since these models are computational, they can be directly incorporated into tactical reasoning system, as described in Chapters 4 and 5.

2.1.5 Situation Awareness

Models of human competence have been developed to describe expert performance in other domains. While these are not generally classified as driver models, they are nevertheless applicable to the tactical driving task. The primary example of such a theory is situation awareness (SA), described as:

> [An expert's] perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future [25].

Situation awareness was first discussed in connection with pilot performance in air-to-air combat and was seen as the critical difference between fighter aces and ordinary pilots [45, 50, 87, 72]. Subsequent research has also connected SA with the ability of commercial airline pilots to fly in difficult conditions [95], the success of commanders to make decisions in tactical battle simulations [99] and the effectiveness of medical technicians during emergency room procedures [46]. In this thesis, I show how the principles behind situation awareness are applicable to driving in traffic, and how the methods used to test situation awareness in human experts can be extended to assessing SA in intelligent vehicles (See Chapter 7).

Situation awareness theory has motivated intelligent systems in other domains. TacAir-Soar [85] is an intelligent automated agent for simulated fighter planes. The goal of the project is to develop automated pilots whose behavior in simulated battlefields is indistinguishable from experienced human pilots. Current versions have demonstrated some competence in one-on-one scenarios similar to those used in real training missions. An interesting feature of this system is that the automated pilots can be "debriefed" using a natural language interface. The researchers claim that the ability to explain why the agent made critical decisions is vital in demonstrating the system's situation awareness. Other work in this area includes: a proposed "Information Manager" for pilot decision support [95] and a proposal for a blackboard architecture-based combat helicopter system [9].

2.2 Situation Awareness in Driving

To see how SA theories originating in tactical combat may be applied to tactical-level driving, it is instructive to note the similarities between air-to-air combat and navigating in traffic. First, both tasks require real-time reasoning in dynamic, uncertain environments. Second, pilots and drivers face information overload since extracting (only) the relevant information from the available sensors is challenging. Third, positional maneuvering is critical in both arenas: combat aircraft seek to achieve desirable configurations relative to the enemy such as higher

altitude, or an advantageous geometric potential [125] while vehicles in traffic seek to maintain space cushions, find suitable gaps and avoid driving in blindspots [66].

2.2.1 Real-time reasoning in dynamic environments

Traditional artificial intelligence techniques are well suited to solving problems where initial and goal states can be clearly specified. Given sufficient time, powerful search methods can efficiently find optimal solutions. Unfortunately, the agent's knowledge of the environment, in both air-to-air combat and tactical driving, is very incomplete (and constantly changing). Optimality is ill-defined and time constraints require that satisfactory actions be taken at every instant.

The number of possible states, given a particular action on the agent's part, is very large. In the absence of a higher-level structure, the search space quickly explodes. Furthermore, new information is revealed as the agents change their configurations, and selected plans may become invalid with the changing situation. Maintaining detailed plans about long-term goals is impractical – rather the agent must focus on short-term planning given the available information.

2.2.2 Information overload

Pilots and drivers receive information about their environment not as a concise list of symbols, but as a constantly changing set of sensor readings. Since time constraints prevent processing all of this information at every time instant, the agent must intelligently select the information most critical to the immediate task. Reece addressed the issue of information overload for robot driving through selective perception [80]. His system, Ulysses exploited the fact that reasoning could be simplified by first focusing on those objects in the environment that most constrain the agent's available actions. For example, when approaching an intersection with a Stop sign, the driver can safely ignore the trajectories of vehicles beyond the intersection, since the Stop sign forces the ego-vehicle to come to a halt. Further improvements were possible by making some assumptions about objects in the environment: for example, upon first observing an oncoming vehicle the agent could note its position and velocity, then "forget" about the vehicle for some time interval — knowing that the vehicle would not be able to close the distance in that time. Reece showed that selective perception techniques reduced average perceptual costs in simulated scenarios by several orders of magnitude. However, when an object in the environment violates these assumptions, selective perception can cause the agent to unintentionally place itself in a more dangerous situation. For example, vehicles that have stopped on the highway are extremely hazardous to drivers who "over-drive" their headlights.

2.2.3 Positional maneuvering

Drivers, unlike fighter pilots, are not typically interested in maneuvering to gain an offensive advantage over other vehicles as a prelude to combat. However, according to defensive driving theory [118, 68], certain configurations of vehicles in traffic are inherently risky while others are relatively safe. This is due to several factors: perceptual limitations; incomplete knowledge of driver intentions; and, availability of escape routes.

The first factor can be caused by either incomplete sensor coverage (e.g., blind spots) or environmental aspects (e.g., glare for vision sensors). Under these circumstances, the driver may not be able to accurately sense the positions or velocities of nearby vehicles. The resulting increase in uncertainty restricts safe options and motivates well-known heuristics like "Avoid staying in a vehicle's blind spot" and "Don't overtake as you approach a hill". The second factor is caused by imperfect communication between vehicles. Although turn indicators, brake lights and hand signals can all provide some indication of other drivers' intentions, it is clear that these are not perfect predictors of the future states of nearby vehicles. While all drivers need to make assumptions on the behavior of other vehicles, relying on these assumptions is unwise — an adjacent driver may suddenly change lanes, or a tailgater may fail to react in time. The third factor is a direct result of overcrowding: for example, driving beside a vehicle limits lane chang-

ing options. Defensive driving theory thus stresses the importance for each driver to maintain a *space cushion* (an area free of other vehicles) around the vehicle whenever possible.

2.3 Discussion

Human driving strategies, and in particular, situation awareness theories, offer valuable insights to the driving researcher. The task-level descriptions structure the driving problem into smaller, manageable components while SA provides a domain-independent framework for processing the perceptual inputs to the system. The tactical-level scenarios used to teach and evaluate defensive driving in drivers' education courses can be salvaged to provide challenges for intelligent vehicles (See Chapter 7). Chapters 4 and 5 describe how the ideas discussed in this chapter can be used to implement the SAPIENT tactical driving systems. However, a design environment is needed where these scenarios can be easily created and repeatably tested. SHIVA, described in Chapter 3 addresses these demands.

Chapter 3

The Simulation Environment

If it's not in the computer, it doesn't exist.
—Murphy's Laws of Technology

Simulation is essential in developing intelligent vehicle systems because testing new algorithms in real world traffic is risky and potentially disastrous. SHIVA (Simulated Highways for Intelligent Vehicle Algorithms) not only models the elements of the domain most useful to tactical driving research but also provides tools to rapidly prototype and test algorithms in challenging traffic situations. Additionally, the scenario control tools allow repeatable testing of different reasoning systems on a consistent set of traffic situations.

3.1 Related Work

While traffic simulators have been in use for over thirty years [17, 20, 37, 122], no existing tool has all the capabilities desirable for designing intelligent vehicle algorithms. Simulators are generally created to study traffic patterns and consequently fail to provide adequate facilities for adding or debugging new vehicle control algorithms.

Few simulators are available for tactical-level research[1]. Of these, Pharos [81], SmartPath [28], and SmartAHS [41] possess most of the necessary characteristics: microscopic vehicle modeling, support for different road geometries, and realistic lane changing models. Unfortunately, since none of these tools were designed specifically for tactical-level simulation, each has significant deficiencies in the area. Pharos makes some unrealistic sensor assumptions such as transparent vehicles; SmartPath, though well suited for modeling platoon concepts (where groups of very closely spaced automated vehicles drive along dedicated lanes) [110], cannot support reasoning systems which violate its state machine architecture; SmartAHS shows significant promise as a general-purpose microscopic simulator, but is currently only a framework.

[1] A number of high-fidelity simulators are currently under development at various industrial research labs. However these have not made available to the general research community.

Design choice	SHIVA implementation
Modeling level	Microscopic
Kinematic accuracy?	Yes, with velocity over dt
Dynamic accuracy?	Available in extended versions
Road geometry	Full 2-D geometry.
Highway support?	Yes
City streets support?	No
Vehicle types	Functionally equivalent
Cognition models	Multiple, user-defined
Engine models	No
Suspension models	No
Tire/surface models	Available in extended versions
Sensor models	Multiple, detailed, user-defined
Driver models	Multiple, detailed, user-defined
Traffic mix	Heterogenous, mixed traffic
Offline simulation?	Yes, on multiple platforms
Integrated visualization?	Yes, on SGI only
Simulation persistence?	Limited: dump & restore to file
Interactive debugging?	Yes
Interactive scenario generation?	Yes
Can users drive vehicles?	Only at the tactical level

Table 3.1: SHIVA, like all simulators, is a compromise between reality and efficiency. This table summarizes the important design decisions.

3.2 Tactical-level Simulation

Every simulation is a compromise between realism and tractability. Task-specific requirements dictate the design decisions[2]. A summary of these important issues is given in Table 3.1.

[2]Serious scholars of simulation are encouraged to examine related work, such as the Papyrus™ NASCAR Racing Simulation.

Motion Model	Benefits	Drawbacks
Slot Car	Executes quickly, simple to code	Unrealistic lane changes
Kinematic 2D	Fast enough for interactive simulation and display.	Unrealistic cornering
Dynamic 2D	Allows tire models, engine models, skidding	Unrealistic collisions, slow if many vehicles
Dynamic 3D	Realistic collisions, banking, suspension	Compute-intensive, very complicated

Table 3.2: A summary of the tradeoffs involved with different choices of motion model.

The choice of motion model proved to be the most difficult design choice. This is because inaccuracies in modeling vehicles at the operational level can create changes in vehicle behavior at the tactical level. Most motion models used in vehicle simulation can be classified into three categories: slot car, kinematic, and dynamic. The tradeoffs in each choice are shown in Table 3.2.

In *slot car* models, lateral positions are assumed to be integral: each vehicle clearly occupies a single lane at all times. One consequence of this is that lane changes are atomic (instantaneous) and vehicle orientations are always tangential to the local road curvature. While these characteristics may be acceptable at the strategic level, they lead to undesirable simplifications at the tactical level.

Kinematic models deal with aspects of motion apart from considerations of mass and force. Thus, these models implement smooth motion, even during lane changes. Furthermore, the vehicles are constrained

to move along realistic trajectories. However, since forces are not modeled, the vehicles will perform maneuvers without skidding or slipping. Such models are typically sufficient for tactical-level research.

Dynamic models are used in simulators where vehicles must respond realistically to forces. In 2-D simulations, these may often include tire-surface interactions and may also involve engine and transmission models. In the 3-D dynamic simulations, the effects of gravity (e.g., accelerating on downhill stretches), and cornering stability may also be added. This level of detail is often needed to accurately model the behavior of operational-level systems.

Realistic simulation requires more computing time, especially when large numbers of vehicles are involved. Since SHIVA is primarily designed for tactical-level research, I chose to use a realistic kinematic model.

SHIVA has since been extended to model 2-D dynamics (for situations such as emergency obstacle avoidance, where skidding and collisions are more common). These extensions are not discussed in this thesis. More information is available in [44].

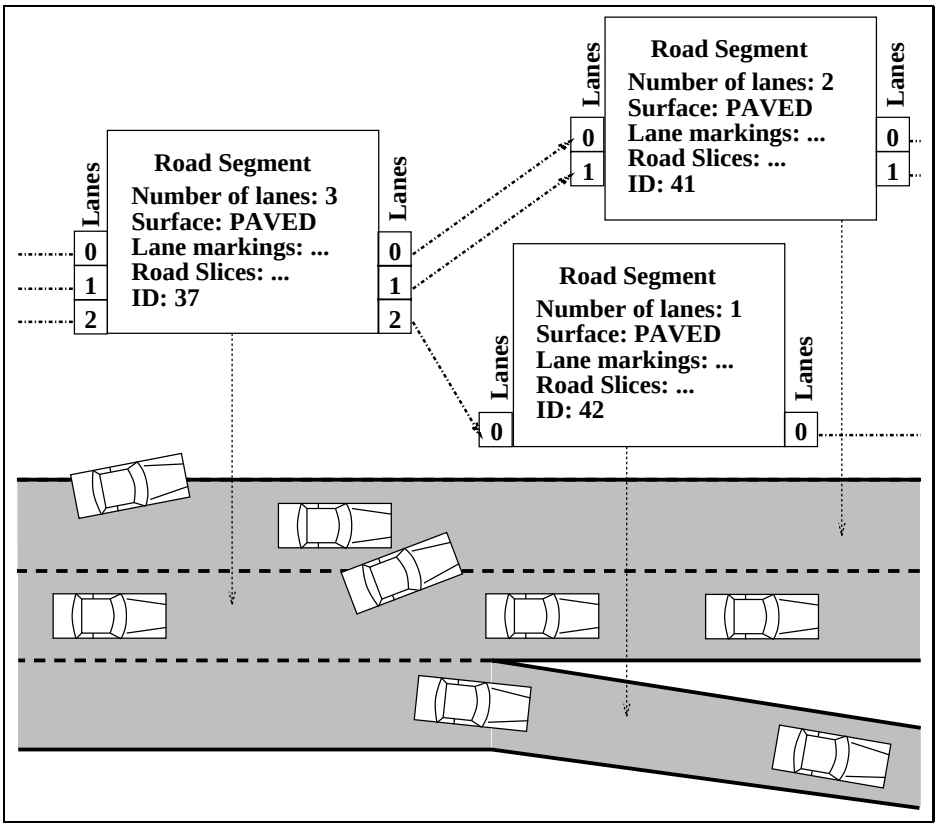

Figure 3.1: SHIVA's road representation consists of a connected set of *Road Segments* — stretches of road with an arbitrary shape and common properties. Vehicles are not required to stay on the road.

3.3 Road Representation

SHIVA represents a highway as a connected set of *Road Segments*, where each segment is a stretch of road with arbitrary shape, a fixed number of lanes, and common properties such as speed limits (See Figure 3.1). Segments connect to other segments on a lane-by-lane basis,

allowing users to model most highway topologies[3]. Vehicles typically drive along road segments, smoothly crossing the boundaries between two segments as they take exits or merge. However, vehicles are not *required* to stay on the road — given bad control algorithms (or to avoid obstacles), vehicles may swerve off the road. This is also shown in Figure 3.1.

Expressing tactical maneuvers in a local *road-coordinate* frame allows vehicle controllers to be invariant to the road geometry. SHIVA represents these coordinate frames explicitly and provides simulator objects with a transparent interface to the road representation. Points represented in road coordinates can be seamlessly expressed in world coordinates as necessary. SHIVA's notion of real-valued lane displacement (rather than integral "lane numbers") is a feature lacking in many existing traffic simulators [21] and allows accurate modeling of lane change maneuvers.

Some realistic simulators [80, 28, 41], while modeling continuous lateral motion, still ignore the impact of lane tracking on vehicle orientation (i.e. they assume that the vehicle is always parallel to the local tangent of the road). Unfortunately these effects are relevant during lane changing. For example, the effects of a slight yaw (1 degree) at a range of 60 meters corresponds to an lateral error of 1 meter in sensing — possibly causing an object in an adjacent lane to appear to be in the

[3]The current implementation does not model intersections, tapered lane merges, or over-passes.

current lane. SHIVA's perception modules (See Section 3.5) correctly account for the vehicle's current orientation when mapping obstacles to lanes. While SHIVA's road geometry is general enough to describe highway geometries, it is not designed to model urban streets. Since the thesis focuses on highway driving, the restriction is not limiting.

3.4 Vehicles

SHIVA models vehicles as kinematically accurate, two-axle, front-wheel-steered mechanisms. Three vehicles classes are modeled, each with its own physical characteristics (See Figure 3.2). The intelligent vehicles are designed to be largely compatible with the Carnegie Mellon Navlab [106, 76] robot vehicles (See Figure 3.3). Over the last decade, the Navlab robots have been used as testbeds for a large variety of experiments in autonomous navigation [61, 73, 101, 104, 74]. Since many of the systems that have emerged from this research will be critical in the automated highway system project, SHIVA minimizes the modifications needed to port cognition modules to the robot. In particular, the interface to the lane-trackers and steering/speed controller (See Sections 3.5.2 and 3.7) is virtually identical to existing systems on the Navlab.

Functionally, intelligent vehicles are decomposed into three subsystems: perception, cognition and actuation (See Figure 3.4). This archi-

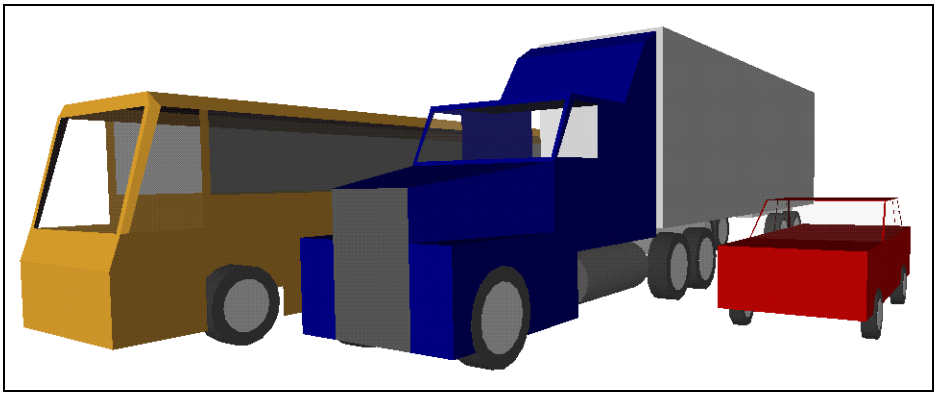

Figure 3.2: SHIVA models a variety of vehicle classes including cars, trucks and buses. Within each class, vehicles may also be equipped with their own configurations of sensors, reasoning algorithms and actuators.

Figure 3.3: The Carnegie Mellon Navlab robot vehicles are testbeds for research in autonomous navigation. SHIVA vehicles are designed to be largely compatible with their real counterparts.

tecture, given appropriate choices for sensors and algorithms, describes both human drivers and autonomous vehicles. Also, since SHIVA is designed in an object-oriented manner, researchers may derive new vehicle types as desired, inheriting the essential attributes and methods from the existing vehicle models.

3.5 Perception

Largely ignored in most simulators, perception remains one of the most difficult problems in mobile robotics. Control algorithms which make unrealistic perceptual assumptions are destined to remain unimplementable on real systems. On the other hand, modeling realistic sensors perfectly is infeasible since the simulated world cannot match the complexity of real life. Each simulator must therefore select the appropriate level of detail suitable for its task. In the tactical driving domain, issues such as sensor field of view, occlusion and obstacle-to-lane mapping are important. The perception subsystem consists of a suite of functional sensors (e.g., GPS, range-sensors, lane-trackers), whose outputs are similar to real perception modules implemented on the Navlab vehicles. SHIVA vehicles use these sensors to obtain information about the road geometry and surrounding traffic. Vehicles may control the sensors directly, activating and panning the sensors as needed, encouraging active perception. Perception objects are grouped into an open-ended sensor hierarchy (See Figure 3.5) which allows de-

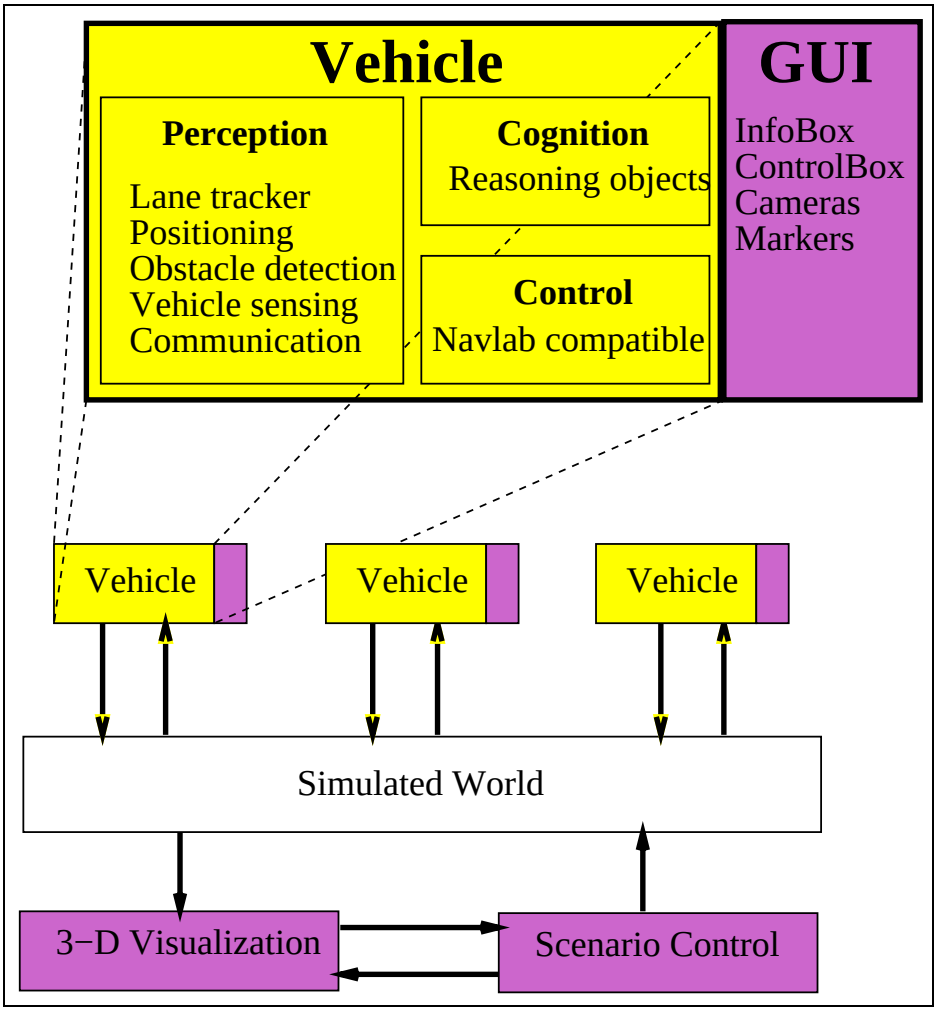

Figure 3.4: Each vehicle is composed of three subsystems (perception, cognition and actuation) which interact with the simulated world. The design tools automatically adapt to each vehicle's internal configuration and provide access to the various components at an appropriate level of detail.

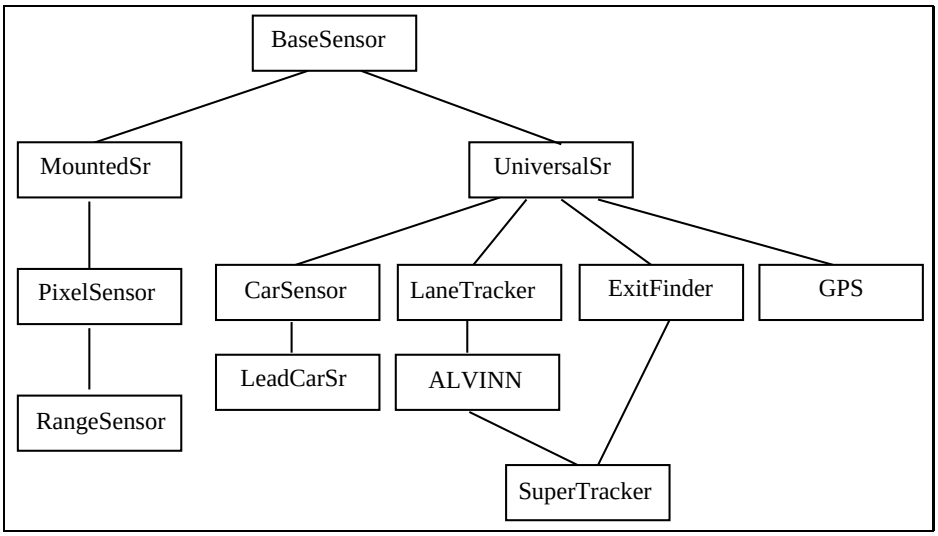

Figure 3.5: The perception subsystem consists of a suite of sensors whose outputs are similar to modules available on real robots. The hierarchy allows designers to create models at the appropriate level of detail and swap them into existing vehicle configurations with minimal modification.

signers to create appropriate models and swap them into existing vehicle configurations with minimal modification. Some of these sensor models are presented below.

3.5.1 Car Detection and Tracking

Since car tracking is an important perceptual task for tactical driving, SHIVA provides two types of car tracking sensors: realistic range sensors and functional car tracking modules. Both types of car trackers have a limited field of view, and range. Sensors cover different areas of interest around the vehicle, and can be activated (or panned) by the

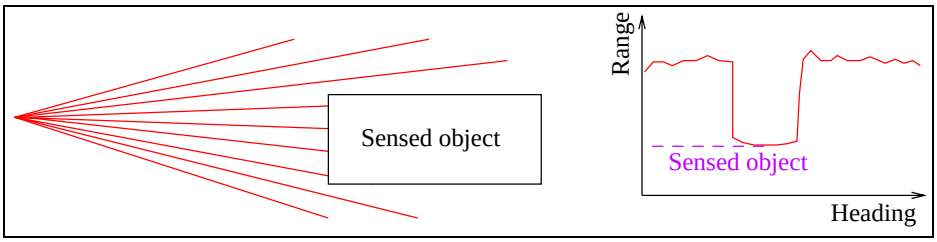

Figure 3.6: The range sensor model simulates a scanning laser sensor with a limited field of view and range. The detection array returns range values for each ray, corrupted by noise. Segmentation, tracking and velocity calculations need to be performed by the perception module.

cognition modules as needed during tactical maneuvers to provide relevant information about surrounding traffic. Since perception is expensive, this selective perception enables cognition modules to minimize the computational load [80].

Range Sensors

SHIVA's range sensor is a 1-D scanning laser sensor with a user-defined range ($\rho_{\max}$), field of view ($\theta_{\max}$) and resolution (given by number of rays, n). Vehicles may have different sensors with individual characteristics, mounted on different locations. See Figure 3.6 for an example of a particular range sensor's output.

The simulated sensor casts n rays, spaced equally in angle over the field of view, and determines each ray's intersection with objects in the environment. Each ray (r_i) returns the distance (ρ_i) to the closest in-

tersection point (or $\rho_{\max}$ in the case of no intersections). Since each r_i corresponds to a pixel in the sensor's retina, the sensor returns an array of pixels (R):

$$R = [\rho_0, \rho_1, \ldots \rho_i, \ldots \rho_{n-1}]$$

To add realism, each ray's scan angle (θ_i) and returned value (ρ_i) are corrupted by noise. Each (θ_i, ρ_i) is subsequently mapped into sensor coordinate frame using: $x = \rho \cos\theta_i$ and $y = \rho \sin\theta_i$. Thus the injected noise translates into errors in the sensed object's position. Note that this method is not equivalent to adding independent Gaussian noise to both x and y — in SHIVA, as in a real range sensor, the errors in x and y are correlated! Cognition modules which use these sensors typically use a Kalman filter [121, 96] to update perception estimates. Thus, the position of a sensed vehicle is represented not as a single point, but rather as an *uncertainty ellipse*.

Since information about objects in the environment is provided only through discretized range readings, cognition algorithms which maintain explicit models of objects in the surroundings must perform segmentation and tracking on this noisy data. Similarly, since velocity information is not provided directly, it must be inferred from differential range measurements[4]. Furthermore, because all range readings are in sensor coordinates, cognition algorithms must perform their own object-to-lane mappings (i.e. convert observed vehicles into road coor-

[4]When direct velocity measurements are provided, (e.g., Doppler radar), they can easily be incorporated.

dinates).

Because this processing is computationally expensive, most tactical-level experiments only use a few vehicles with such sensors, supplemented by larger numbers of vehicles with functional perception modules (discussed below).

Functional Car Trackers

SHIVA's functional perception modules simplify the processing required by cognition modules. The functional models support the following enhancements:

Car tracking: detected vehicles are segmented and correctly tracked in subsequent frames. Optionally, each new vehicle is tagged with a unique ID number to simplify reasoning.

Object-to-lane mapping: all position measurements are converted into road coordinates (distance along the road and relative lateral offset), enabling cognition modules to largely ignore the effects of the local road geometry.

Vehicle type detection: the class of vehicle (e.g., car, bus, truck) as well as vehicle size is directly reported, simplifying gap reasoning calculations.

Direct velocity measurement: the true velocity of the observed object is reported, enabling accurate calculation of time-to-impact or acceleration constraints.

Note that the functional car tracker does not provide the higher-order derivatives such as acceleration or jerk, since these (very noisy) measurements cannot be reliably obtained in real life. While the functional car tracker can also corrupt its output with simulated noise, the output is not equivalent to that obtained from a realistic sensor model (since the error rates for ρ and θ are not explicitly represented in the functional model).

A danger with using functional sensor models is that the simulator may provide unrealistically complete data of the environment. This tendency is common in earlier work in tactical simulation. For example: perfect measurement of sensed vehicles' acceleration [69]; transparent vehicles [80]; and, straight road assumptions [27]. SHIVA's functional sensors only return information that is available given existing perception technology assumptions.

3.5.2 Lane Trackers

Since realistic lane tracking is an operational level task (and beyond the scope of this simulation), SHIVA's lane tracking module is a functional model of a complete vision-based lane tracking system. The in-

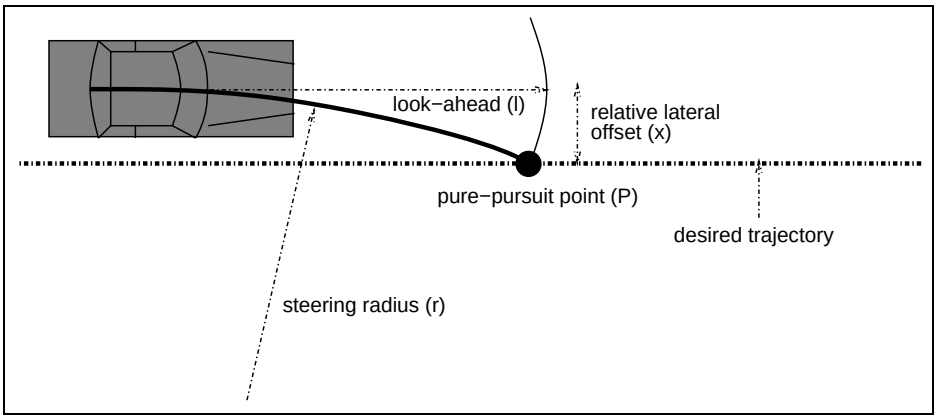

Figure 3.7: The lane tracker module provides information about the local road geometry in the form of a *pure-pursuit* point (P). This point on the road determines the steering curvature that should be used to steer the vehicle onto the desired trajectory at the specified *look-ahead* distance (l).

terface is based on the ALVINN [73] road follower and provides information about the local road geometry through a *pure-pursuit*[114, 4] point (See Figure 3.7).

A pure-pursuit point is defined to be the intersection of the desired vehicle trajectory and a circle of radius (l), centered at the vehicle's rear axle midpoint (assuming front wheel steer). Intuitively, this point describes the steering curvature which would bring the vehicle to the desired lateral offset after traveling a distance of approximately l. Thus the position of the pure-pursuit point maps directly onto a recommended steering curvature: $k = -2x/l^2$, where k is the curvature (reciprocal of steering radius), x is the relative lateral offset to the pure-pursuit point in vehicle coordinates, and l is a parameter known as the look-ahead distance. The look-ahead distance is an empirically determined pa-

rameter which corresponds to the gain of the steering controller. When l is too short, vehicle control may become unstable; when l is too long, the controller response is sluggish. Experiments on the Navlab vehicles [4, 73] have shown that good results are obtained when $l = 15$ meters in low speed situations and $l = 25$ meters at highway speeds. Human factors studies have also demonstrated that the pure-pursuit model approximates the human steering response [82].

Although lane tracking is a challenging robotics problem [23, 54, 73], most highway simulations [122, 69] still equate lane tracking and lane occupancy. Such simulations model lane changes as instantaneous actions, allowing their cognition modules to completely ignore the difficult decisions needed during lane changing — and fail to capture behaviors such as lane straddling and aborted lane changes.

SHIVA also supports lane trackers that actively steer the vehicle. By varying the desired lateral offset (represented by a pure-pursuit point) smoothly from the center of one lane to the center of the desired adjacent lane, these road followers are able to implement lane changes [48]. It is important to note that the actual lateral offset of the vehicle always lags the current position of its pure-pursuit point during the lane change.

3.5.3 Positioning

SHIVA simulates two types of positioning sensors: dead reckoning and global positioning systems (GPS). While both return the vehicle's current position in global coordinates, they differ in their noise characteristics: dead reckoning sensors return measurements that become increasingly inaccurate with distance traveled (since differential errors accumulate with vehicle motion), while GPS errors do not depend on the distance traveled. Positioning information, in conjunction with on-board digital maps, can allow cognition modules to initiate maneuvers (such as changing lanes into the exit lane) well in advance of the desired exit. In an actual implementation, such a positioning system could be augmented by road-sign or landmark recognition.

3.6 Cognition

Cognition modules must select appropriate tactical-level actions based upon information provided by the perception subsystems. The operational level aspects of the driving task are handled by either the actuation subsystem (See Section 3.7) or by modules such as lane trackers which accept tactical commands (such as the lane tracker described in Section 3.5.2).

Different cognition strategies need different perceptual inputs. SHIVA

enables researchers to create customized configurations of sensors for each cognition module. For example, purely reactive cognition modules can directly accept raw data from the realistic sensors. By contrast, the rule-based system (See Chapter 4) reasons about higher-level concepts such as gap sizes, velocities and acceleration constraints. The latter configuration requires significant processing of the sensor outputs — such as segmentation, differencing, and obstacle-to-lane mapping.

SHIVA places no constraints on the reasoning of individual cognition modules except that the outputs be compatible with the controller configuration of the vehicle. This also ensures that algorithms implemented in simulation can be later tested on the Navlab with minimal changes.

3.7 Actuation

While kinematics are realistically modeled in SHIVA, dynamics are largely ignored. In the absence of engine, transmission and suspension models, simulating actuation is relatively straightforward. The control interface is compatible with the Carnegie Mellon Navlab controller: cognition modules provide a control tuple, $C = (k, v)$ where k is the desired curvature and v is the desired velocity. Cognition modules provide C to the controller at regular time intervals; in the absence of new information, the controller will servo on the last commanded tuple.

As in the Navlab, steering and velocity control are decoupled. This simple model does not allow vehicles to follow precise trajectories, but is sufficient for specifying coarse behavior. The controllers generate steering wheel position and throttle/brake commands to bring current vehicle curvature and velocity to the desired positions as quickly as possible. The controller also enforces limits on acceleration and rate of change of curvature. Experiments on the Navlab [73, 47] indicate that a bandwidth of 10 Hz is required for highway driving using this controller. Thus, SHIVA uses a fixed time-step of 100ms and expects that cognition modules will provide a C at each time-step.

As discussed in Section 3.5.2, SHIVA's controller also supports low-level steering control through active road-following. In this paradigm, cognition modules specify a desired lateral position by manipulating the pure-pursuit point. The lane tracker then automatically computes the steering curvature required to bring the vehicle to the desired lateral offset. This allows cognition modules to remain ignorant of vehicle kinematics.

3.8 Simulation and Design Tools

Most existing simulators are designed to model only a few vehicle configurations and reasoning agents. By contrast, SHIVA's architecture is open-ended — enabling researchers to integrate new sensors, con-

trollers and intelligences into existing vehicle specifications. Users can study interactions between different intelligent algorithms by creating scenarios with heterogeneous vehicle configurations. An example of this is mixed-mode traffic where "human operated" and "autonomous" vehicles use very different sensors and driving strategies. SHIVA allows researchers to examine the repercussions of changing the proportions of different types of vehicles (e.g., manual versus automated[5]) on the highway.

Since reasoning in tactical situations is complex, algorithms are best developed in an iterative manner. To support rapid refinement of reasoning systems, SHIVA provides a set of tools that fall into three main categories. The work described in the following sections was done in collaboration with John Hancock [102, 103].

3.8.1 Visualization and Validation

The first step to validating algorithms is observing them in action. Visualization tools allow the designer to qualitatively evaluate his/her algorithms from different points of view. To verify whether vehicles are behaving reasonably with a given design, the researcher needs to be able to see how the vehicles behave both as individuals and as aggregates.

[5]The value of such studies is critically dependent on the validity of the manual traffic driver models.

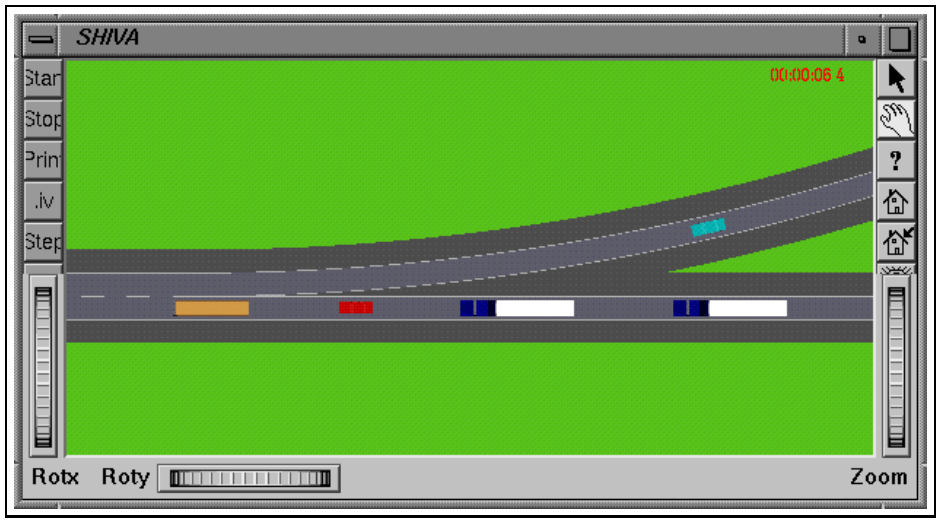

Figure 3.8: SHIVA's primary view is an interactive camera which shows the simulation from overhead at an aggregate level. Vehicles can be selected from this view for detailed inspection.

SHIVA provides a flexible suite of visualization tools using Open Inventor (a 3-D graphics library developed by Silicon Graphics)[6]. SHIVA's primary view (See Figure 3.8) is an interactive camera which shows the simulation from overhead at an aggregate level. Multiple views that track vehicles may be created by selecting the desired vehicle(s) in the primary view. These secondary views can display driver's eye (or other arbitrary) perspectives (See Figure 3.9). Since humans make tactical decisions from behind the wheel, these views are helpful in judging the quality of decisions made by the AI algorithms. A vehicle selected for monitoring can be ordered to change color or begin dropping markers — "virtual bread-crumbs" — allowing researchers to observe not only the vehicle's current position, but also its previous trajectory.

[6]These tools are only currently supported in the SGI implementation of SHIVA.

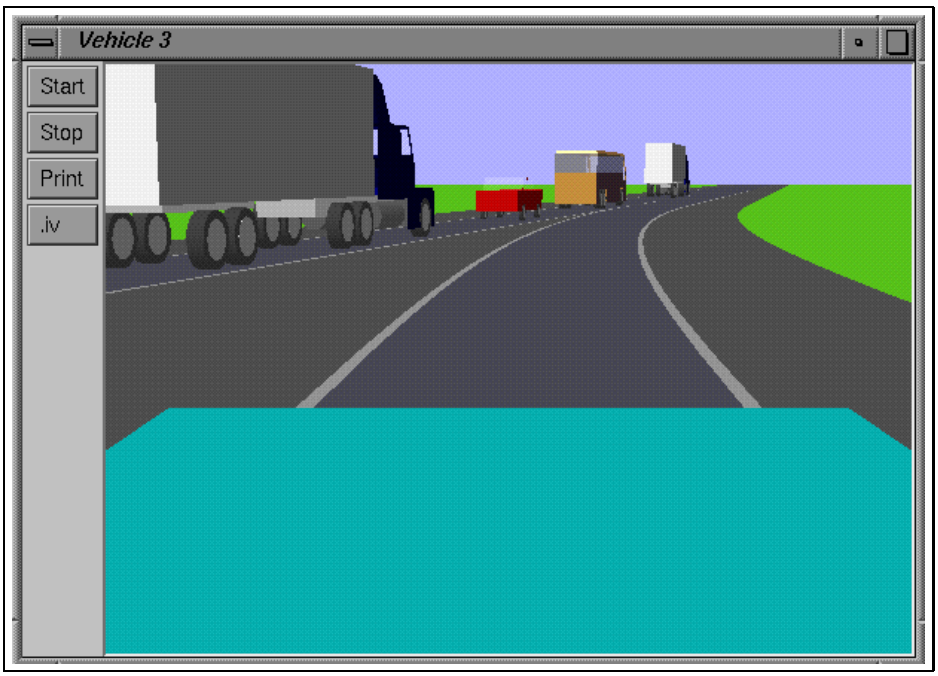

Figure 3.9: SHIVA allows users to select arbitrary secondary views for detailed examination of the scenario. This figure shows a driver's eye perspective of the scene.

SHIVA's visualization tools also support a host of features such as printing views to PostScript (for printing), 3-D dumps of situations as VRML (Virtual Reality Modeling Language) files, and animations of tactical scenarios as MPEG or animated GIF movies. This allows researchers to demonstrate their results to a wide audience (particularly when such files are made available over the World Wide Web)[7].

3.8.2 Measurement and Analysis

To improve on existing algorithms, users need the ability to analyze what the vehicles are doing correctly, and to identify what they are doing incorrectly. Although visualization tools allow the designer to see if something is wrong with the algorithms, this information is generally insufficient to diagnose the problem. To perform this quantitative analysis on-the-fly, researchers require access to reasoning object internals during the simulation. Current debugging tools are inadequate for this task since they only display a few variables at a time.

SHIVA displays qualitative and quantitative information through *Info Boxes* which update continually during the simulation. Researchers may request this information on a per-agent basis and focus only on the relevant details. Since simulation and animation are fully integrated, problems spotted using the visualization tools may be immediately investigated. Most importantly, these *Info Boxes* rapidly customize to the

[7] See <http://www.cs.cmu.edu/~rahuls/shiva.html>.

vehicle configuration so that only (and all of) the relevant information is displayed. Without customization, *Info Boxes* automatically adjust themselves to display whatever information is available for that vehicle configuration through inheritance. For example, an *Info Box* for a MonoSAPIENT vehicle (See Chapter 4) displays information about its cognitive state, while one for a PolySAPIENT vehicle (See Chapter 5) shows the accumulated votes for each action (See Figure 3.10).

In addition to local information about each vehicle, SHIVA also collects statistics about global performance. These include numbers of collisions, throughput, average velocities and exit success rates. Researchers can customize the appropriate aggregate measures and monitor them during the simulation.

3.8.3 Interactive Exploration and Modification

The final important requirement for a simulation and design system is the ability to interact with and modify objects on-the-fly. This provides several benefits: 1) supports rapid iterative development; 2) enables better exploration of the parameter space; 3) allows fine-tuned scenario generation.

Iterative development without recompiling is vital for incremental algorithm design. Interactive parameter modification can be used to expose algorithms to sudden changes in environment, encouraging ro-

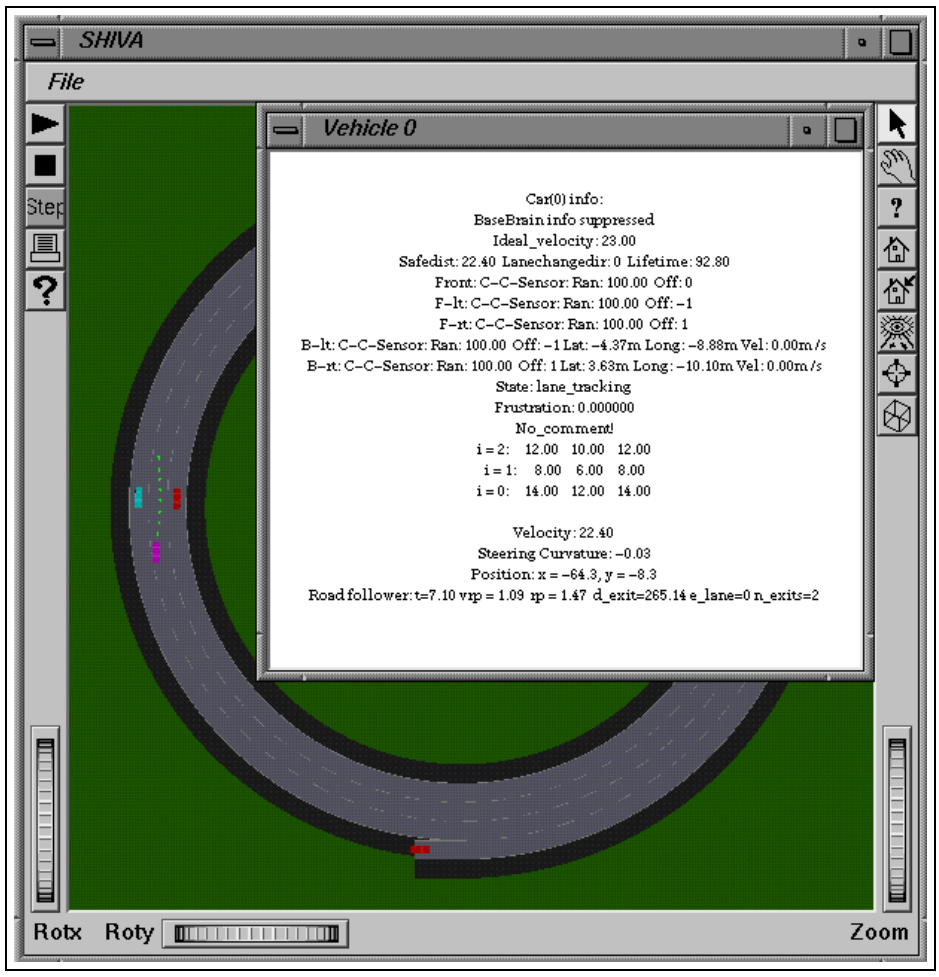

Figure 3.10: This figure, shows the *InfoBox* corresponding to a PolySAPIENT vehicle. Note that the *InfoBox* has configured itself to display the vehicle's sensors (showing the two stopped cars on the track) and the accumulated action votes. See Chapter 5 for more details on PolySAPIENT.

bustness. This is done primarily through SHIVA's *Control Boxes* — hierarchical interface tools which can be customized for particular vehicle configurations (See Figure 3.11). By enabling researchers to interactively create traffic situations on the fly, ControlBoxes also provide a means for scenario generation.

3.9 Scenario Generation

Many interesting traffic situations occur very rarely, both in real life and in simulation. However these are also the scenarios where tactical level reasoning systems are most challenged. Rather than forcing researchers to wait patiently for these events to arise, SHIVA provides users with several interactive scenario generation features. There are two aspects to scenario control: creation and manipulation. We examine each in turn.

3.9.1 Scenario Creation

The primary tool for scenario creation is the *Factory*. Each factory produces vehicle configurations (e.g., trucks with a PolySAPIENT cognition module and some specified sensor suite) from a specified set and injects them at the selected places on the highway network, with the desired probabilities, and at the appropriate times. Since factories are

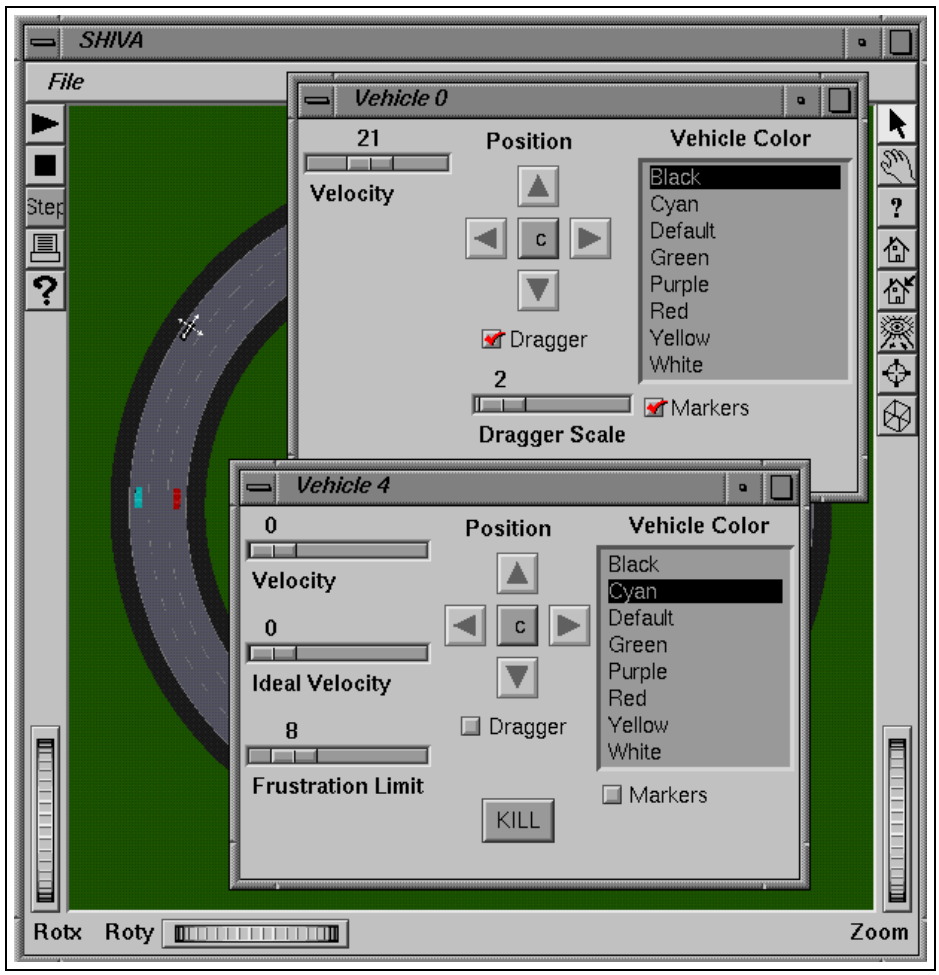

Figure 3.11: *Control Boxes*, such as the ones shown here, allow users to interact with vehicles during the course of the simulation. Each interface tool automatically configures itself to the appropriate vehicle configuration. For example, the *Control Box* in the foreground provides sliders to allow users to change MonoSAPIENT parameters.

specified using text files, they can easily be customized at run-time.

Once vehicles appear on the roadway, users can interactively move them into their desired positions. In addition to an intelligent *pick-and-place* interface, SHIVA supports a *tactical driving interface* which maps user commands into tactical level actions (e.g., move along a road segment, change lateral position or select velocity).

Users may select initial conditions for the scenario, such as starting velocities and vehicle parameters using *Control Boxes*. Once completed, the scenario can be saved to disk for future use. This is especially valuable when different cognition modules need to be (repeatably) tested on the same set of scenarios.

3.9.2 Scenario Manipulation

The integrated nature of SHIVA's simulation and animation environment enables users to interactively adjust the behavior of vehicles during a scenario. By pausing the run, changes can be made in parallel using different *Control Boxes*. The tactical control interface can be used to "drive" the selected vehicle around the track. Furthermore, users can ignore the (operational level) task of lane tracking and focus on the tactical issues, even on complicated road structures.

Slider bars enable run-time adjustments of key parameters in the var-

ious cognition modules. SHIVA also supports a number of ways to (indirectly) influence the behaviors of vehicles at run time — for example, the user may command a *breakdown* and force the selected vehicle to slow down and stop. Other functions allow users to simulate various malfunctions in the sensors or actuators.

3.10 Saving the Simulation State

As seen in Section 3.9, scenario generation requires the ability to recreate simulation states. One way to accomplish this is to build the simulator on top of a persistent database (e.g., SmartAHS [41] is built over Versant). The biggest advantage of this is that state is implicitly saved (allowing rewinding of the simulation on demand). Unfortunately, the price for this convenience is that each timestep of the simulation incurs a substantial overhead — preventing such a system from providing an integrated simulation/animation environment using current computer hardware.

SHIVA compromises on convenience by offering only a subset of this feature — researchers may interactively save the current simulation state to file, and restore it (exactly) as required. The data is saved as a commented text file, allowing users to examine (and edit) any of the elements in the saved state as needed. The primary purpose of this feature is to enable researchers to interactively create traffic scenarios

and then efficiently simulate them off-line without graphics. SHIVA can also dump state at regular intervals for later visualization or analysis. This feature also allows researchers to observe the evolution of the system from the same initial conditions given different reasoning object parameters.

3.11 Conclusion

SHIVA was developed as a safe and economical alternative to testing cognition modules in real traffic. Its extensive scenario generation tools enable researchers to test different control strategies on a series of stored traffic situations. SHIVA also collects statistics such as the number of collisions, average velocity and exit success rate, providing quantitative measures of a vehicle's performance over the scenario. Offline simulation with stored scenarios supports the development of learning strategies for tactical reasoning. Online integrated simulation and visualization features allow the flexible exploration of intelligent vehicle behavior in dynamic settings.

SHIVA is used as a development platform for the cognition modules discussed in the following chapters. Its scenario creation features are used in Chapter 7 to create tactical scenarios for intelligent vehicles.

Chapter 4

MonoSAPIENT: Rule-Based Situation Awareness

The function of the expert is not to be more right than other people, but to be wrong for more sophisticated reasons.
—Dr. David Butler [31]

Chapter 2 presented the idea of a situation awareness (SA) based tactical reasoning system, termed SAPIENT (Situation Awareness Planner Implementing Effective Navigation in Traffic). Such a system would monitor traffic and road features around the intelligent vehicle using on-board sensors, and suggest suitable actions to the robot's operational-level controllers.

In this chapter, I describe the first implementation of such a system, named MonoSAPIENT, designed in the SHIVA simulation environment (See Chapter 3). MonoSAPIENT is based upon a traditional mobile robot architecture, and explicitly represents driving knowledge in the form of decision tree rules.

4.1 Motivation and Related Work

MonoSAPIENT's design was motivated by material developed to improve human driver performance. In defensive driving literature, knowledge is often expressed in the form of high-level rules [117, 66, 118, 92, 68]. For example:

> Before lane changing, look for traffic approaching from the rear in the new lane. Also, check cars that are about to enter the new lane from the far lanes [66].

Such rules are useful in driver education because their triggers are clear (e.g., "before lane changing") and the recommended actions are also explicit (e.g., "check cars that are about to enter the new lane"). While these rules are too vague to fully specify the behavior of an intelligent vehicle, they indicate that explicit hand-coded driving knowledge may be a good methodology for a tactical driving system.

Previous work in tactical-level driving has concentrated on expressing driving knowledge in the form of rules [69, 57], decision trees [80] or finite state machines [21]. While these reasoning systems have only explored greatly simplified aspects of the driving task (e.g., Niehaus & Stengel assume curve free roads while Kotchetkov assumes single-lane highways), expert systems have been used in other fields with promising results [78, 85].

4.2 System Overview

At the tactical level, intelligent vehicle behavior falls under three broad categories [66]. MonoSAPIENT represents each category explicitly as a global *mode*.

Lane tracking: The vehicle is in a relatively clear lane and is able to travel at its desired velocity. Lane tracking systems are responsible for lateral control.

Car following: The driver attempts to maintain a constant headway behind the current lead vehicle. As in the *lane tracking* case, lateral control is limited to lane-tracking.

Lane changing: Regardless of motivation (passing, taking an exit etc.) the vehicle behavior always consists of moving from one travel lane to another over the span of a few seconds. During this time,

vehicle speed is constrained by vehicles in both initial and target lanes.

The behavior of the vehicle in each mode follows well-defined control laws, while the transitions between states are triggered by a decision-tree. Before examining the details of the system, I present a brief look at mobile robot architectures and indicate the forces which motivated MonoSAPIENT's design.

4.3 Mobile Robot Architectures

The traditional approach to robotics [30] involves a three-stage processing cycle as shown in Figure 4.1. In the first stage, sensors gather information about the world and convert it to a symbolic internal representation, known as a *world model*. In the second stage, the world model is processed by AI algorithms (typically involving planning and search) to find a course of action for the robot to achieve its goals, known as a *plan*. In the final stage, this plan is executed by the robot as a series of *actions* (actuator commands).

This process, as described above, suffers from several serious drawbacks. First, the approach implicitly underestimates the role played by perception [80, 43]: due to sensing constraints and uncertainties, the world model is likely to be both incomplete and partially incor-

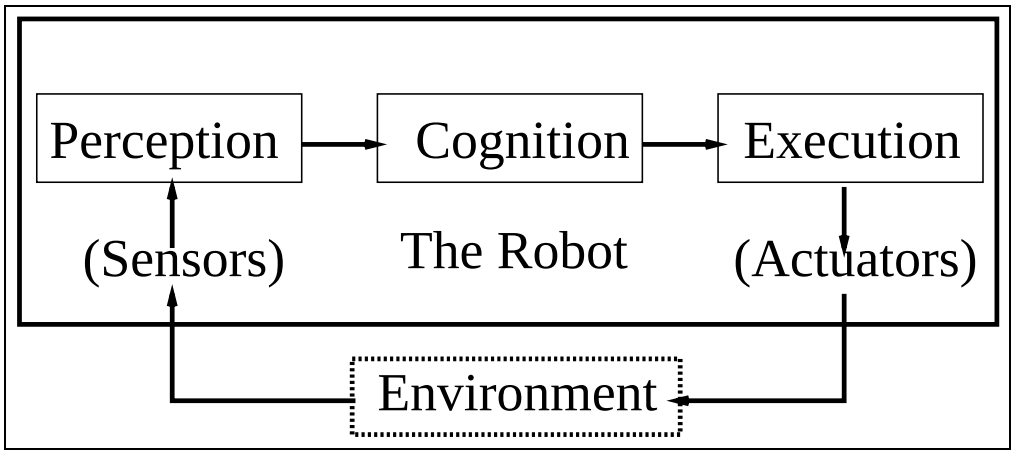

Figure 4.1: Traditional robot architectures can be viewed as a three-stage processing loop. The robot obtains information about the world in the first stage, processes it in the second stage to generate a plan, and executes this plan in the third stage.

rect. Second, the process assumes that it is possible to plan a complete path from the robot's initial state to its goal state(s). This is infeasible (particularly in real-time) in most complex domains due to an explosion in the number of searchable states, and the inability to perfectly predict the outcomes of actions. Third, the chosen plan cannot be guaranteed to execute perfectly — the robot may be forced to react immediately to unforeseen problems (very likely, given an incomplete world model). Consequently, current mobile robot architectures recognize that planning-heavy approaches to real-time problems in dynamic environments (such as driving in traffic) are infeasible [18, 84].

MonoSAPIENT recognizes the critical role of perception in tactical driving and makes reasonable assumptions about available information (See Section 4.4). By abandoning the hope of generating complete plans

from its world model, the chosen architecture interleaves planning and execution (as described in Section 4.7). Short-term planning enables real-time decision making based on incomplete world models.

4.4 Perceptual Processing

In an intelligent vehicle, most of the problems associated with sensing are addressed at the operational level. For example, vision-based lane trackers automatically deal with variations in lighting conditions [73] and positioning systems combine noisy inputs from several sources [105]. At the tactical level, it is reasonable to expect that perception systems can reliably track *traffic entities* such as vehicles, lanes and exits — along with information about the entity such as the speed of a vehicle or the distance to an exit. Thus, previous work in tactical driving [80, 21, 57] has generally assumed that the intelligent vehicle has a perfect model of its surroundings. However, the following perceptual effects are important, even at the tactical level:

Blindspots: The intelligent vehicle's sensors may not be able to scan all around the vehicle, or may only provide limited information (such as presence/absence of objects). To tackle blindspots, the vehicle must make some assumptions about the contents of the unknown region (either optimistic or pessimistic).

Occlusion: Even if all-round sensor coverage is available, on-board sensors cannot see through opaque objects. This problem is especially severe when the intelligent vehicle is surrounded by large vehicles such as trucks or buses. Assuming that occluded regions are free of obstacles is risky.

Sensor Noise: Measurements from real sensors are noisy for a variety of reasons. Physical phenomena (e.g., specular reflections), software limitations (e.g., range buckets) and unreliable tracking all introduce uncertainty into the observed world attributes.

Sensor Limits: Current sensing technology is able to provide sufficiently accurate measurements of range and relative velocity of other vehicles (using stereo, radar or optical flow) within a reasonable sensor range. At the far ranges, errors in bearing may make object-to-lane mapping unreliable. Furthermore, higher-order derivatives of position such as acceleration or jerk are very noisy, and cannot be assumed to be available[1].

In both MonoSAPIENT and PolySAPIENT (See Chapter 5), the sensor configuration shown in Figure 4.2 is assumed. The perception modules which process the sensor outputs are summarized in Table 4.1. The vehicle sensors are assumed to be reliable within 75 meters (sufficient to cover the stopping distance of the intelligent vehicle).

[1]One method of obtaining these higher-order derivatives is to use communication between vehicles; this is feasible only in fully-automated AHS concepts such as [110, 91].

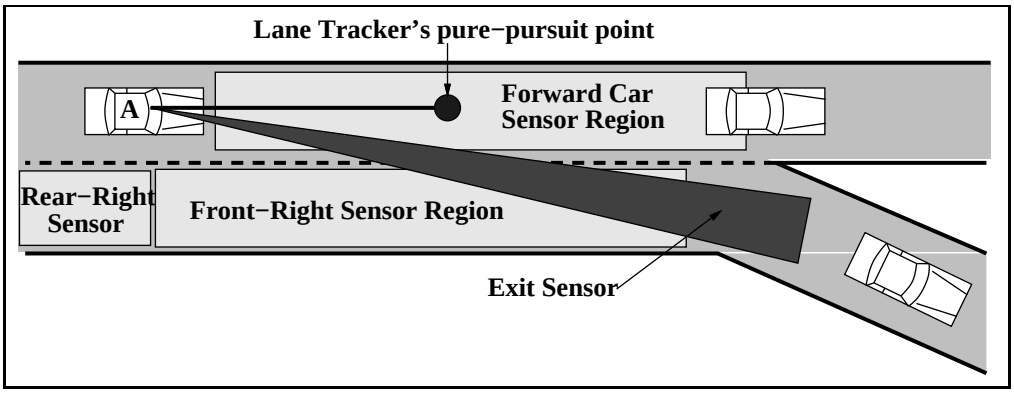

Figure 4.2: Functional sensor configuration used for driving in traffic.

Sensor Type	Sensed attributes
Lane Tracker	lane width, current position of pure-pursuit point, current lateral displacement, current road curvature, lane type (travel, on-ramp, etc.)
Exit Finder (range = 500m)	distance to chosen exit, lane delta to chosen exit, current exit number
Odometer	total distance traveled
Speedometer	current velocity
Positioning (e.g., GPS)	current global position (for strategic-level map)
Vehicle Sensor (range = 100m)	longitudinal distance to vehicle, lateral offset to vehicle, current velocity of vehicle, vehicle length and width, vehicle class: car, truck, etc. (optional)

Table 4.1: A summary of the perception modules used for driving in traffic, and the functions that they support.

4.5 The World Model

Information extracted from by the perception modules is assimilated into a global representation of the world. MonoSAPIENT's representation of the world has three aspects relevant to tactical-level driving:

1. Robot self-state, including physical and cognitive components
2. Road state, including road geometry and exit information
3. Traffic: speeds, relative positions and hypothesized intentions

Each is briefly discussed in this section.

4.5.1 Robot Self-state

While not always considered part of a traditional world model, the robot's self-state is an important influence in selecting the next action. Its two components are:

Physical State: consists of information sensed from the speedometer, odometer and other encoders. At the tactical level, the important elements include: current speed, current steering curvature, current lateral displacement (distance from center of current lane), and distance traveled.

Cognitive State: consists of two types of information: high-level preferences such as the desired speed and the preferred lane of travel; and, internal states which reflect the progress of currently executing plans such as lane changing or passing maneuvers. In this chapter, I will only discuss the following cognitive states (each maps to observable tactical maneuvers in progress):

lane tracking: The intelligent vehicle drives at its desired speed in the current lane, as steered by the road follower. The vehicle may transition to either of the two *lane changing* states on the basis of a short-term plan, or to the *car following* state if a vehicle appears in the forward sensor.

car following: The intelligent vehicle maintains a constant, safe headway (based on higher-level preferences) to the vehicle in front. A *lane change* may be initiated by a short-term plan, or the vehicle may move to the *lane tracking* state if the lead vehicle changes lanes or accelerates beyond the preferred velocity.

changing left/right: These states are an indication that the intelligent vehicle is involved in a lane transition (typically initiated by a short-term plan). At the conclusion of the lane change, the vehicle will move into either a *lane tracking* or *car following* state, as dictated by vehicle sensors. In response to unforeseen events (such as a pinch condition), a lane change in progress may need to be *aborted*.

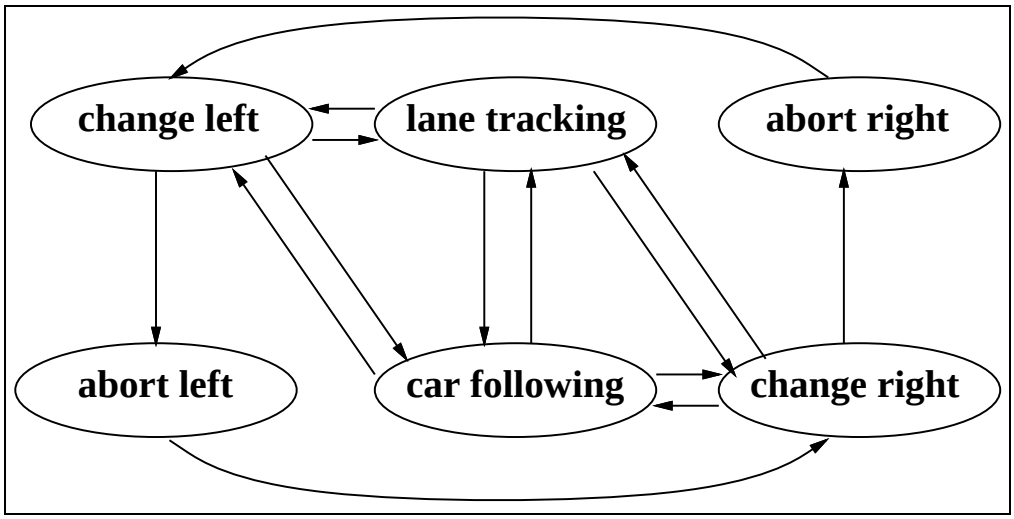

Figure 4.3: A summary of MonoSAPIENT's cognitive states, and their connections.

aborting change left/right: These intermediate states indicate that the previously executing lane change became unsafe, and that a new lane change (recovery) should be initiated. The cognitive state transitions to one of the *lane changing* states once the recovery maneuver has been initiated.

These states and their connections are summarized in Figure 4.3.

4.5.2 Road State

Road state refers primarily to information received from the lane tracking modules, potentially supplemented by on-board digital maps. Aspects such as lane markings and the number of lanes constrain lane-

changing maneuvers while posted speed limits and sharp curves constrain speed choices. The model must also record whether the intelligent vehicle is approaching its desired exit, if it is in the wrong exit lane, and the status of any known obstacles on the roadway (since they constrain lane options).

The road state is also used by perception modules to perform object-to-lane mapping. For example, when an obstacle is detected "dead-ahead" while the vehicle is on a curve, the cognition module must know whether the obstacle is actually on the roadway (and more importantly, whether the obstacle is in the current lane).

4.5.3 Traffic State

The modeling of other vehicles is the most important aspect of tactical driving. While the current speeds and relative positions of other vehicles can be ascertained in a relatively straightforward manner, their future behavior cannot. Thus, the intelligent vehicle is forced to make some hypotheses about the vehicles' intentions. Experienced human drivers can often predict the behavior of other drivers with surprising accuracy [66]. However, this involves both a large database of world knowledge (presumably unavailable to the intelligent vehicle) as well as perception well beyond the current state of the art in computer vision. For example, a driver can look in the rear-view mirror and notice a brightly colored sports car weaving through traffic and infer that this

car is likely to pass by in the left lane. Such sophisticated reasoning requires: 1) tracking a small, partially occluded, moving object through a cluttered scene; 2) recognizing the object as a sports car; 3) hypothesizing that people who drive such cars are likely to drive in a particular manner; 4) predicting the possible future positions of the sports car based upon this high-level understanding. Clearly this is well beyond MonoSAPIENT's current scope.

Instead, MonoSAPIENT makes the assumption that vehicles will continue to drive in their current lanes at their current velocities over the extrapolation time-interval. If the extrapolation results in vehicles which violate MonoSAPIENT's model of their headway, MonoSAPIENT predicts that the vehicles will adjust their speed to maintain the minimum headway. The primary drawback of this driver model is that lane changes (which a human driver may have been able to predict) cannot be anticipated. Future implementations of SAPIENT should incorporate the more accurate driver models that are anticipated in the Automated Highway System research community.

4.6 The Decision Tree

As described above, MonoSAPIENT's cognitive states are analogous to global *modes* of operation. A hand-crafted decision tree organizes the transitions between modes and the operations that are performed in

each state. This section illustrates the structure of the decision tree by describing one of its most important components: the lane change logic. A lane change consists of two sub-tasks: the *trigger*, and the *execution*. The first sub-task is responsible for determining whether a lane-change is needed, and if so, whether is it feasible. Once a suitable gap has been found, the second sub-task monitors the gap to ensure that it remains safe throughout the maneuver — aborting the maneuver if conditions change.

The logic for triggering and executing lane changes is presented in Figures 4.4 and 4.5, respectively. The complexity of the decision criteria prevents them from being visualized as trees, so they are summarized in pseudocode, with some of the important aspects described below:

frustration: A quantity which reflects the desirability of passing the current lead vehicle. Discussed at greater length in Section 4.7.1.

safe gap exists: Indicates that the gap evaluation process has found a suitable gap in the desired target lane. This can either be passive (where the gap must be adjacent to the vehicle), or active (where the intelligent vehicle attempts to pursue a gap by changing velocity).

lane exists to the left/right: Refers to checking lane-tracking systems to see if the target lane exists, and is legal for travel (can be replaced by a local map query, if available).

constrain velocity: MonoSAPIENT is conservative during lane changing, constraining vehicle speed so that it is safely following any cars in *both* original and target lanes. This is so that an aborted lane change does not create an unsafe configuration. The drawback of this is that MonoSAPIENT cars drive very timidly and are unable to accelerate and merge into dense traffic (as seen in Chapter 7).

perform gap reasoning: indicates that the target gap should be evaluated on the criteria described in Section 4.7.2. When a gap becomes unsafe, a lane change in progress may have to be aborted.

shift pure-pursuit point: At the tactical level, lane-changing is encapsulated as a shift in the lane-tracker's pure-pursuit point. This is detailed in Section 4.7.3. During a lane change, the pure-pursuit point is incrementally moved from the original lane into the target lane. MonoSAPIENT relies on the lane-tracker to generate a smooth trajectory.

Complexity results from the fact that certain lane changes need to be suppressed. For example, the intelligent vehicle should not initiate a lane change when it is already in the desired exit lane (doing so may cause it to miss its exit), nor should it use a left-exit lane as the passing lane (this might force it to exit the highway). Politeness criteria, such as driving in the right lane when possible, also need to be presented explicitly. These features require modifying the lane change logic in a

```
// Mode is "lane_tracking" or "car_following"
//
if (mode is car following) |
    update frustration
    servo velocity to do safe car following
|
if (desired exit within range &&
    lane type allows change &&
    safe gap exists in target lane) |
    execute(initiate change to target lane)
|
if (desired exit is in this lane && nearby) |
    execute(stay in current lane)
|
if (frustration exceeds threshold) |
    // Prefer passing on the left side
    if (lane exists to the left &&
        markings allow change && safe gap exists) |
        execute(pass blocker on the left)
    | else if (lane exists to the right &&
        markings allow change && safe gap exists) |
        execute(pass blocker on the right)
    | else |
        execute(stay in current lane)
    |
| else |
    if (ego-vehicle in the wrong exit lane &&
        lane to the left/right exists && safe gap exists) |
        execute(initiate change to left/right lane)
    | else if (lane exists to the right && safe gap exists &&
        // Prefer to travel in the right lane
        lane to the right is not an exit lane) |
        execute(initiate change to right lane)
    |
|
```

Figure 4.4: A simplified view of MonoSAPIENT's decision tree logic for initiating lane changes.

non-local manner, resulting in a tangled web of conditions — a factor discussed in greater detail in Section 4.8.

4.7 Interleaved Planning and Execution

At every time step (typically 0.1 seconds), the perception systems update the components of MonoSAPIENT's world model. Various algorithms then process the world model in order to generate a short-term plan. The plan can then be translated into operational-level commands

```
// Mode is "Executing Lane Change"
//
if (blocker_o exists in original lane) |
    constrain velocity based on blocker_o
|
if (blocker_t exists in target lane) |
    constrain velocity based on blocker_t
|

perform gap reasoning on target gap
if (target gap is not safe) |
    mode = |appropriate type of abort_lane_change|
    perform gap reasoning on original lane gap

    new target is now the safer of the two available gaps
|

query lane tracker for current lateral position
if (pure-pursuit point centered in target lane) |
    // lane change has ended
    if (blocker exists in target lane) |
        mode = car_following
    | else |
        mode = lane_tracking
    |
| else |
    // lane change is continuing
    shift pure-pursuit point towards target lane
|
```

Figure 4.5: A simplified view of MonoSAPIENT's decision tree logic for monitoring lane changing.

for the intelligent vehicle controllers. This section details this process by focusing on the scenario shown in Figure 4.6, where the intelligent vehicle (denoted by Car A), is blocked by the slower moving vehicle (Car B).

In a deliberative system, the world model derived from the given situation would be used to generate a comprehensive plan (e.g., decelerate for t_1 seconds, execute a left lane change for the next t_2 seconds, accelerate to pass Car B during the subsequent t_3 seconds, and end the passing maneuver with a right lane change in t_4 seconds). Unfortunately, this is not possible in the tactical driving domain. For example, Car A's location may prevent it from seeing potential vehicles ahead of Car B — particularly when the blocker is a large truck. In such a

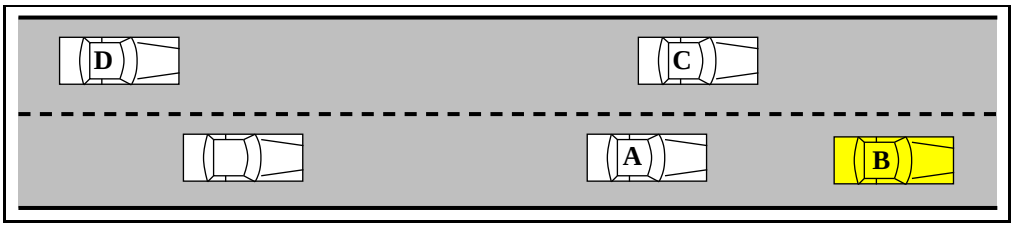

Figure 4.6: Lane changing scenario: the intelligent vehicle (A) decides to pass the blocker (B). A suitable gap is chosen, and a short-term plan initiated.

case, the detailed plan would have been generated from a world model that bore little resemblance to reality. Similarly, if Car B were to speed up during the passing maneuver (in an attempt to prevent overtaking), the final right lane change would have to be delayed. For these reasons, the mobile robot community has moved towards architectures that emphasize *interleaved* planning and execution [30, 29] — where, rather than searching completely to the goal state, the robot generates a partial plan and begins execution, possibly re-planning from the current state at every new time-step [84]. Not only does this make real-time planning tractable, but it also allows the reasoning system to integrate the latest information, without committing the robot to potentially incorrect plans. The following sections detail important aspects in the triggering and execution of the lane change.

4.7.1 Deciding to Pass

While car following, MonoSAPIENT computes a *frustration* function, $\Phi(t)$, at every time step, t. Intuitively, the frustration is a measure of the desirability of passing the current lead vehicle. When the frustration exceeds a *frustration threshold* (a strategic-level parameter), the intelligent vehicle moves into the gap reasoning phase (See Section 4.7.2). Until then, the frustration continues accumulating over time as given in the following equation:

$$\Phi(t) \quad = \quad \kappa\Phi(t-1) + \phi(t) \tag{4.1}$$

where: κ is a parameter ($0 < \kappa < 1$) corresponding to impatience, and $\phi(t)$ refers to the instantaneous frustration collected in the current time step. Note that, if the blocker speeds up before the pass is initiated, $\Phi(t)$ will decay over time.

$$\phi(t) \quad = \quad \begin{cases} 0 & \text{(if } v_b > v_d\text{)} \\ \lambda\alpha_b + (1-\lambda)\alpha_d & \text{(otherwise)} \end{cases} \tag{4.2}$$

where: v_b is the blocker's velocity; v_d is the desired velocity (goal set by strategic-level); λ is a strategic-level parameter; and, α_b and α_d are the decelerations required to reduce speed (over the headway span of time) to v_b and v_d respectively. α_b and α_d correspond to acceleration constraints [80], thus leading to:

$$\alpha_b \quad = \quad \frac{(v - v_b)^2}{2t_h} \tag{4.3}$$

$$\alpha_d \quad = \quad \frac{(v - v_d)^2}{2t_h} \tag{4.4}$$

where: v is the current velocity, and t_h is the current headway to the blocking vehicle[2]. Once the decision to pass has been made, MonoSAPIENT actively hunts for suitable gaps, as described in the next section.

4.7.2 Gap Reasoning

During lane-changing, MonoSAPIENT reasons explicitly about *gaps*. This can be contrasted with PolySAPIENT's approach (See Chapter 5), which examines individual *vehicles*, and examines how they constrain available actions. The primary benefit of an approach which explicitly reasons about gaps is that the intelligent vehicle is not forced to wait passively for an appropriate gap to show up, but can instead pursue it actively[3].

Consider how this operates in the lane changing scenario given in Figure 4.6. Assuming that sufficient Φ has accumulated, the intelligent vehicle will seek gaps in traffic on either side. The road model indicates a right lane change is infeasible (since there is no lane to the right). Thus, MonoSAPIENT will begin examining transitions to the *changing-left* state.

[2]A simpler function, such as $\phi(t) = \max(v_d - v_b, 0)$ gives qualitatively similar results.

[3]As detailed in Chapter 5, PolySAPIENT's vehicle reasoning objects discourage it from driving beside vehicles in adjacent lanes. This space cushion allows PolySAPIENT to change lanes without active gap hunting.

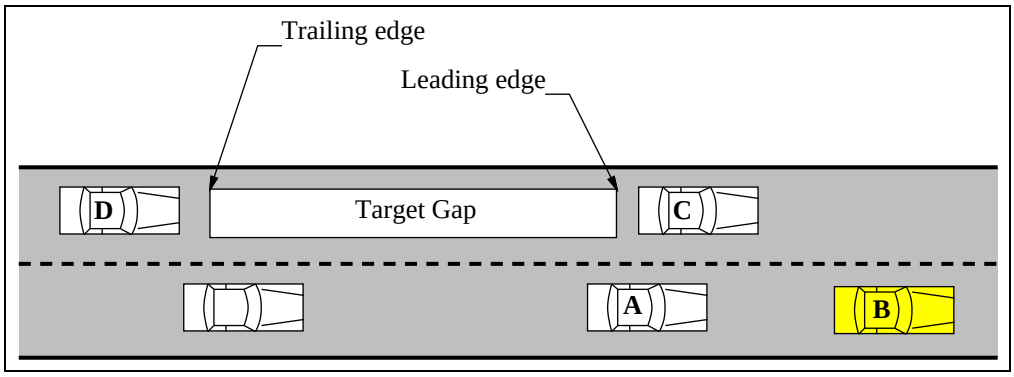

Figure 4.7: The lane changing scenario, with the gap between vehicles C and D emphasized. Gap attributes such as size, velocity, rate of growth and the time to arrival are used in the evaluation process to determine the gap's suitability.

MonoSAPIENT examines all visible gaps within sensor range (approximately 75 meters to the front and rear of the vehicle), and evaluates their desirability. Consider the marked gap between cars C and D (See Figure 4.7). Its measurable attributes are:

Size: Defined as the distance between the front edge of the trailing vehicle (C), and the rear edge of the leading vehicle (D).

Velocity: Defined as the velocity of the closest edge of the gap.

Rate of Growth: Defined as the difference between the velocities of the leading and trailing edges of the gap. An expanding gap is preferred.

By assuming constant velocities over the extrapolation time interval, the following derived gap attributes are obtained:

Inverse Time to Arrival: The time (t) until a possible merge point (where the headway both ahead and behind is safe) in the gap comes adjacent is used to compute the derived attribute: ($1/t$). Note that when $t < 0$, the gap is moving away from the robot. Also, when $|t|$ is small, $|1/t|$ can be very large. To avoid the asymptotes, the current implementation of MonoSAPIENT limits this attribute to the range $[-2, 2]$.

Expected Size: This is an estimate of the gap's size at the *time of arrival*, based on its current *size*, and its current *rate of growth*.

Leading Edge Velocity Differential: Defined as the difference between the speed of the leading edge (Car C) and the intelligent vehicle's current velocity. A negative value indicates that the new gap is inferior to the current situation, and should be avoided because it is moving too slowly — unless this jeopardizes a higher-level goal (e.g., if this is the only gap in the desired exit lane).

Trailing Edge Velocity Differential: Defined as the difference between the intelligent vehicle's velocity and the speed of the trailing edge (Car D). A positive value is preferred; a negative value indicates that MonoSAPIENT should check whether Car D can safely decelerate.

Leading Edge Desirability: Defined as the difference between the speed of the leading edge (Car C) and the robot's *desired* velocity. A high value indicates that the gap is opening up, and is thus more desirable.

Candidate gaps are pruned to eliminate obviously unsuitable targets (for example, those that are too small[4]). The attributes of each of the remaining gaps are combined in a weighted sum, and the highest scoring candidate gap is selected for the passing maneuver. In practice, very few gaps remain after pruning since MonoSAPIENT's limited sensor range (typically 100m ahead and back) does not leave much room for 4 second gaps at highway speeds (where each gap is $4 \times 30 = 120$ meters long).

4.7.3 Action Execution

The driving task requires multiple actions to be performed simultaneously at the control level (e.g., braking for a curve, while also steering). In MonoSAPIENT, these subtasks are encapsulated into individual higher-level *maneuvers*. This approach is consistent with the models for human information processing described in [38]:

> ... as we have discussed, two or more coordinated simultaneous actions are under the control of a single motor program, while multiple independent actions are under the control of separate motor programs operating in parallel.

[4]MonoSAPIENT is conservative driver, refusing to merge into gaps that will violate its headway thresholds, typically 2 seconds, on both front and back.

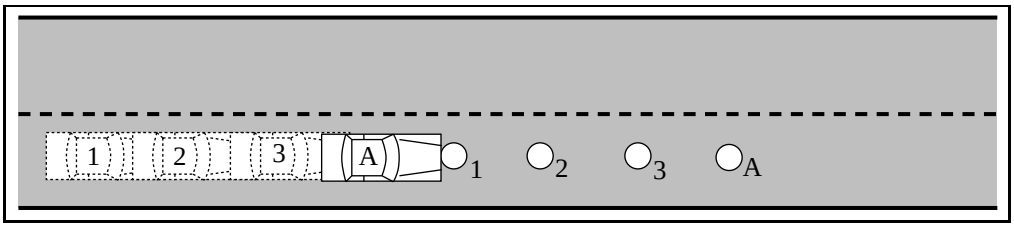

Figure 4.8: Under normal circumstances, the pure-pursuit point is centered in the current lane, at a constant distance ahead of the vehicle. Compare with Figure 4.9.

Thus, the complex operational-level task of changing lanes is captured, at the tactical level, by a maneuver which manipulates the pure-pursuit point. The pure-pursuit point was discussed earlier in connection with the lane tracker (See Section 3.5.2) and defined to be the intersection of the desired vehicle trajectory with an arc centered at the vehicle's rear axle. Under normal circumstances, the pure-pursuit point is centered in the current lane, at a constant distance[5] ahead of the vehicle, as shown in Figure 4.8. During lane changing, the desired lateral position is changed linearly from the origin to the destination lane. Since the operational-level controller automatically servos the steering angle to bear towards the pure-pursuit point, this strategy leads to a smooth lane change, as shown in Figure 4.9. A similar approach has been demonstrated on the Navlab robot [48].

Thus, for lane-changing, once a suitable gap has been chosen, MonoSAPIENT transitions into the appropriate *lane changing* mode. This commands the operational-level controller to match velocities with the

[5]The distance to the pure-pursuit point is constant in *time* (e.g., 1.5 seconds ahead of the intelligent vehicle).

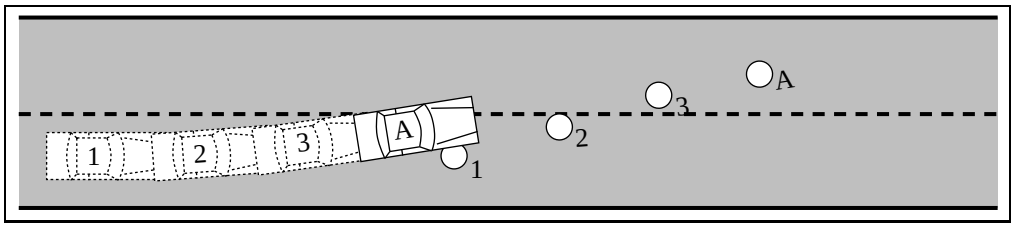

Figure 4.9: To execute the lane change, the pure-pursuit point is moved laterally from the origin to the target lane. This causes the operational-level controller to steer the vehicle smoothly into the next lane.

target gap[6], and initiate a lane change. The operational-level control for lane changing has been studied in depth [77, 123, 47], and is not discussed further in this thesis.

Unlike most tactical-level reasoning systems, which assume that the commanded lane change always succeeds, MonoSAPIENT actively monitors the progress of the maneuver to determine whether the lane change should be aborted. If the selected gap closes unexpectedly, MonoSAPIENT will transition to an *aborted lane change* state as discussed in Section 4.5.1.

4.7.4 Aborted Lane Changes

While rare, aborted lane changes are critical to the safety of the intelligent vehicle and need to be handled carefully. At this point, the intelligent vehicle's priority is to resume lane-tracking in any suitable

[6]MonoSAPIENT ensures that it is traveling at least as fast as the trailing edge of the gap and not faster than the leading edge.

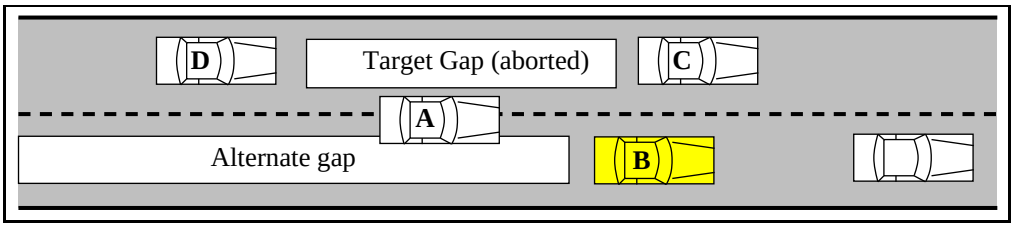

Figure 4.10: Vehicle A initiated a lane change maneuver to pass Car B. However, the desired gap is no longer safe. Car A must abort its lane change.

lane. Consider the situation shown in Figure 4.10 — a possible outcome from the scenario shown in Figure 4.6.

Here, the Car A's passing maneuver has become unsafe due to Car D's unexpected velocity change. Since the target gap is no longer desirable, Car A must abort its lane change. MonoSAPIENT checks to see that the intelligent vehicle's position in the original lane is still available before instructing the operational-level controllers to shift the vehicle back into the right lane, behind Car B. Note that it is possible to create pathological scenarios where neither lane is safe (due to simultaneous pinch conditions). In such cases, the intelligent vehicle has no good option and selects the better of the two (bad) gaps.

4.8 Discussion

Although defensive driving books and current situation awareness theory support MonoSAPIENT's monolithic, rule-based architecture, I show

that this standard approach is impractical for a realistic application such as driving in highway traffic. While nothing prevents such a system from making competent tactical driving decisions, the monolithic architecture makes it difficult to develop tactical reasoning in an incremental manner.

Consider the development of a rule for lane changing. The first version of the lane changing rule used in MonoSAPIENT was:

> "Change left if the vehicle ahead is moving slower than $f(v)$ m/s, is closer than $h(v)$, and the target lane is clear (no other vehicles within $g(v)$ meters)."

where: $f(v)$ is the desired car following velocity, $h(v)$ is the desired car following distance (headway), and $g(v)$ is the required gap size for entering an adjacent lane. This rule correctly handles many lane changing conditions but is unsatisfactory because the following elements are not considered:

Exits: The intelligent vehicle does not consider the implications of exits. This manifests itself as two major problems. First, the vehicle will change lanes out of an exit lane to pass a blocker, even if this maneuver will cause it to miss the exit. Second, the vehicle will pass a blocker (possibly on the right) by entering an exit lane, even if this will cause it to be forced to exit prematurely.

Aborted Changes: The rule implicitly assumes that lane changes are *atomic* (a similar criticism has been raised against [69, 64]). This prevents the intelligent vehicle from reacting to an unexpected change in the desired gap's status.

Clearly, the rule can be patched to partially handle the first condition by adding the following clause to the precondition:

"... and if the desired exit is further than $e(x, y, v)$ meters"

where $e(x, y, v)$ is a distance threshold to the exit based on current lane, distance to exit and velocity.

Correctly handling the second complication is much more difficult. The atomic lane-change assumption is generally coupled with an integral lane assumption (i.e., the cognition module reasons that vehicles are either fully in one lane or the other). Once this is extended, the concept of *change left* must be split into *initiate change left* and *continue executing change left* to account for lane changes that never complete.

Thus, it can be seen how MonoSAPIENT's monolithic structure prevents easy modification of the existing rules to add new functionality. The following disadvantages can be identified. First, as illustrated above, creating realistic rules requires the designer to account for many complex factors. Second, a small change in the desired system behavior generally requires many non-local modifications (also noted

in [21]). Third, the hand-coded rules perform poorly in unanticipated situations. And finally, implementing new features requires one to consider an large number of interactions with existing rules. While these are not problematic in a toy domain, they do not scale easily towards large applications such as the automated highway system.

In Chapters 5 and 6, I describe an alternative implementation of SAPIENT, known as PolySAPIENT, which addresses these concerns. By employing a distributed intelligence, PolySAPIENT decouples the different elements of the tactical driving task, solving each independently and combining the final results. The new structure allows situation awareness for tactical reasoning to be developed incrementally. Chapter 7 presents performance results for MonoSAPIENT and PolySAPIENT.

Chapter 5

PolySAPIENT: Distributed Situation Awareness

Made weak by time and fate, but strong in will,
To [drive], to seek, to find, but not to yield!
—Alfred Tennyson, Ulysses [conveniently misquoted]

As seen in Chapter 4, monolithic tactical driving modules (such as single-layer finite state machines), suffer from serious problems when they are scaled to complex traffic situations. The two main difficulties are: rules can interact in unexpected ways, and adding new functionality often requires non-local modifications to the system. Thus, large monolithic systems often contain subtle inconsistencies and are difficult to maintain. This motivates an alternative approach to imple-

menting situation awareness.

5.1 System Overview

PolySAPIENT is a distributed implementation of situation awareness that addresses these problems through the use of independent local experts, each specializing in a small aspect of the tactical driving task. Each expert, termed a *reasoning object* (See Section 5.2), recommends actions for the current situation based solely on its area of specialization. These recommendations, expressed in terms of votes and vetoes, are sent to a voting arbiter (See Section 5.5) which selects from these, the next action to be taken by the intelligent vehicle.

The PolySAPIENT architecture is shown in Figure 5.1. The *perception modules* (depicted as ellipses) are connected to the intelligent vehicle's sensors, and perform functions such as lane tracking or vehicle detection. Wherever possible, they correspond to existing systems available on the Carnegie Mellon Navlab (e.g., the lane tracker is based on ALVINN [73]). Each *reasoning object* (shown as a dark rectangle) obtains information about the situation from one or two perception modules, and independently calculates the utility of various courses of action. This information is then sent to the *voting arbiter*, which integrates these recommendations and selects the appropriate response. Finally, the tactical action is translated into steering and velocity com-

mands and executed by the operational-level controller. Each of these components is detailed in the remainder of this chapter.

5.2 Reasoning Objects

Conceptually, each reasoning object is associated with a single, relevant, *traffic entity* in the intelligent vehicle's tactical-level surroundings, and is responsible for assessing the impact of this entity on upcoming tactical actions. For example, the reasoning object associated with a vehicle ahead monitors the motion of that vehicle and determines whether to continue car following, initiate a lane change or begin braking. Similarly, a reasoning object associated with an upcoming exit is concerned with recommending the lane changes and speed reductions needed to successfully maneuver the intelligent vehicle to the off-ramp. This section discusses the decomposition of the tactical-level environment, and details the structure of reasoning objects.

5.2.1 Decomposing the Tactical Driving Task

As seen in Figure 5.1, there is no communication between reasoning objects — reasoning objects operate in parallel without sharing intermediate results. Thus, central to the PolySAPIENT architecture is an Assumption of Local Independence:

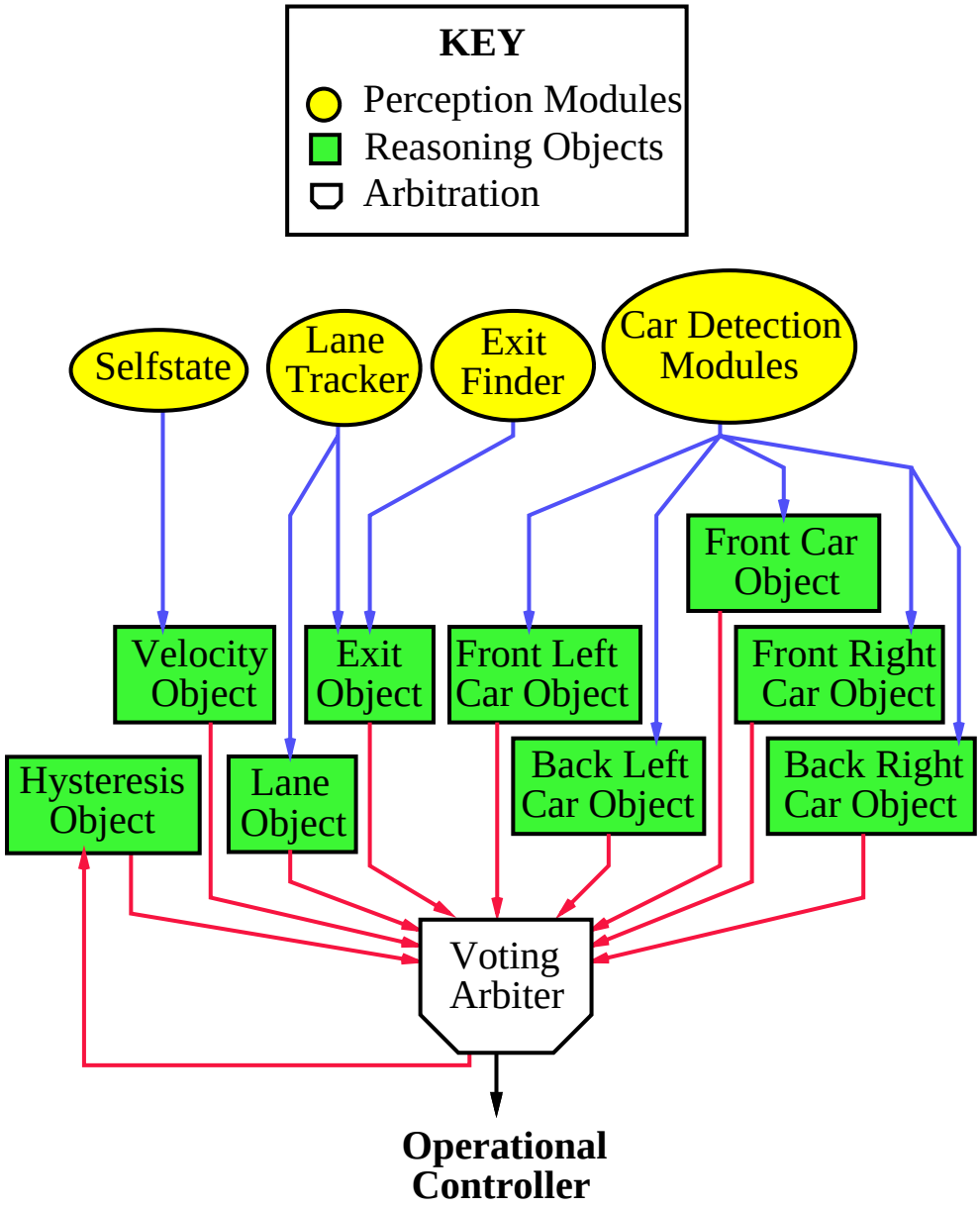

Figure 5.1: PolySAPIENT consists of a collection of reasoning objects which recommend actions based upon local considerations. Each reasoning object monitors a subset of the vehicle's sensors and votes upon a set of possible actions. Action fusion is performed by a domain-independent, voting arbiter.

> When determining the utility of a given tactical-level maneuver, each relevant traffic entity in the environment may be treated independently.

At first glance, this statement seems overly restrictive since it prevents reasoning objects from calculating the effects of potentially important interactions (e.g., an exit reasoning object, when determining the utility of a lane change, is unable to factor in the presence of a vehicle in the target lane). In practice however, the voting scheme presented in Section 5.5 correctly resolves most such interactions. The limits of the Local Independence Assumption are explored in Section 5.8.

By mapping each relevant traffic entity in the environment to a single reasoning object, PolySAPIENT gains a number of significant advantages. First, the tactical reasoning task is partitioned in a very clear manner, allowing designers to check at a glance whether PolySAPIENT addresses a particular aspect of driving. For example, by observing the collection of reasoning objects, it is clear that the current implementation cannot reason about traffic lights; verifying this hypothesis in a monolithic system, such as MonoSAPIENT, potentially requires the researcher to examine conditions in many scattered rules. Second, PolySAPIENT can be modified for new environments by adding appropriate reasoning objects. For instance, adding traffic light reasoning to the current implementation of MonoSAPIENT would be very difficult; by contrast, adding similar capabilities to PolySAPIENT would require very little beside designing an appropriate reasoning object (one that

needed to know only about traffic lights). Third, PolySAPIENT reasoning objects are not forced to share a single world representation, each reasoning object can be implemented by a different researcher, each selecting algorithms and representations appropriate to the specific task.

In addition to the inputs shown in Figure 5.1, the reasoning objects also require information about relevant strategic-level preferences. For instance, the exit reasoning object needs to know whether to take the upcoming exit.

Figure 5.2 shows how the reasoning objects presented in Figure 5.1 map to traffic entities in a particular scenario. Here, the intelligent vehicle (A), is approaching its desired exit while following a slow blocker. The four labels depicting the instantiated reasoning objects point to the objects' respective traffic entities. Note also that, where possible, each reasoning object is also assigned to monitor a single perception module. The vehicle on the exit ramp is too far away to be detected by the vehicle's sensors, and thus, it has no associated reasoning object.

In order to map PolySAPIENT's reasoning objects onto the functions required for tactical-level reasoning, it is instructive to comprehensively list the physical entities which can affect driving in traffic:

- Road properties: geometry, legal lanes, speed limits etc.
- Nearby vehicles: sizes, positions, and velocities
- Exits: distance, exit lane, speed constraints

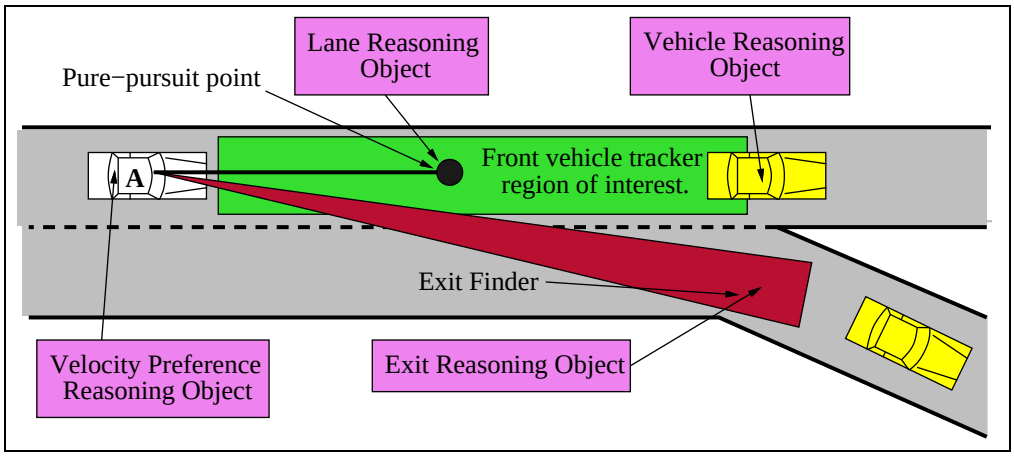

Figure 5.2: PolySAPIENT reasoning objects are associated with relevant physical entities in the environment. In this situation, the intelligent vehicle (A) is following a slow blocker as it approaches its desired exit.

- Self-state: current velocity, lateral position, explicit goals

Each of these entities can then be represented by one or more reasoning objects. Some of these reasoning objects (e.g., self-state) are permanent, while others (e.g., the vehicle ahead) are instantiated and destroyed in response to changing conditions or intentions. Details of PolySAPIENT's current reasoning object types may be found in Appendix A.

Each reasoning object tracks the associated physical entity's relevant attributes by monitoring relevant sensors. For example, a reasoning object associated with a nearby vehicle normally tracks longitudinal position and velocity, and lateral position (mapped into road coordinates). The tracking has two important implications. Firstly, it allows

the reasoning object to get a better estimate of the relevant attribute. Secondly, the reasoning object can accumulate statistics that can help influence decisions. For instance, based on the irregular lane-keeping performance of a nearby vehicle (an indication of an inexperienced or intoxicated driver), the reasoning object associated with that vehicle could favor actions that maintain a greater distance from that vehicle[1]. Thus, PolySAPIENT is not purely reactive; the local state associated with each reasoning object allows PolySAPIENT to make decisions based on past history which are not necessarily apparent in the observable current state of the environment.

5.2.2 The Structure of a Reasoning Object

Externally, all reasoning objects share a similar structure — each object accepts inputs from a subset of the intelligent vehicle's perception modules and sends outputs to the voting arbiter as a set of votes over the entire *action space* (See Section 5.5). Internally, however, PolySAPIENT's reasoning objects are heterogenous, maintaining local state and using whichever representations are most applicable to the assigned subtask. For example, the reasoning objects responsible for exit management are rule-based but the reasoning objects monitoring other vehicles use generalized potential fields.

[1]While current implementations of vehicle reasoning objects are not this sophisticated, PolySAPIENT provides this framework.

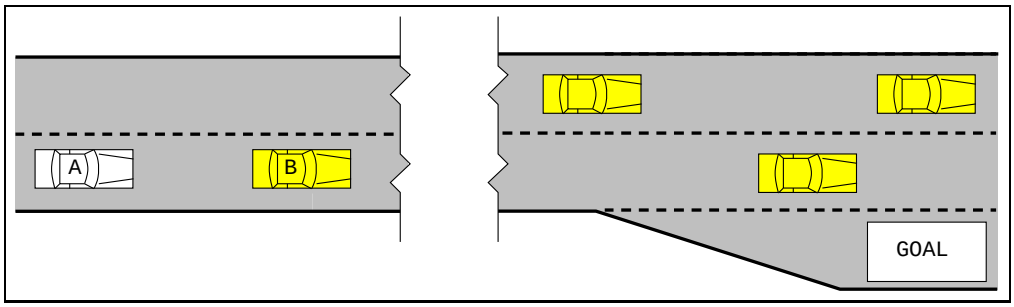

Figure 5.3: The intelligent vehicle (Car A), heading for the upcoming exit, is blocked by a slow-moving vehicle in its lane (Car B). Each entity, represented as an individual PolySAPIENT reasoning object, provides independent recommendations on the utility of the available actions. These votes are used by a knowledge-free arbiter to determine the best course of action.

Now that the decomposition of the driving task and overall structure of reasoning objects have been addressed, it is natural to ask, "What exactly does a reasoning object *do*?". An intuitive understanding of the answer can be gained from anthropomorphizing PolySAPIENT's components and examining a possible "dialogue" occurring in the scenario from Chapter 1, shown again in Figure 5.3. Recall that Car A, the intelligent vehicle, is hoping to take the upcoming right-lane exit. Car B is a slow-moving blocker in the same lane. The intelligent vehicle must determine whether passing Car B will lead to missing the exit. The relevant reasoning objects are:

- Self-state Reasoning Object (permanent, associated with Car A)
- Lane-tracking Reasoning Object (monitors road geometry)
- Vehicle Reasoning Object, associated with Car B

- Exit Reasoning Object, associated with the exit
- Knowledge-free, stateless arbiter

In this example, to simplify the description, the action space has been restricted to five discrete actions: {*accelerate*, *decelerate*, *shift-left*, *shift-right*, *coast*}.

Arbiter: *Based solely upon your perceptual inputs, how would you rate the utility of* each *of the intelligent vehicle's possible next actions? Please submit exactly one, real-valued, vote per action (positive for approval, negative for disapproval).*

Lane: *No suitable lane to the right, so* veto *the* right-shift *action. Slight disapproval of* left-shift *since we are already centered in a safe lane. Zero votes on other actions.*

Car B: *After generating a generalized potential field around Car B, involving our current closing speed and distance, I vote positively for* left-shift, right-shift *and* decelerate, *negatively for* accelerate *and mildly negatively for* coast.

Exit: *The distance to the upcoming exit is within a range threshold. We are already in the correct lane, so I vote negatively for both* left-shift *and* right-shift. *However the traffic density at the exit is not high enough to cause me to* veto *those actions. Our current speed is just within the speed restrictions imposed by the exit, so I will vote against* accelerate.

Self-state: *We are going slower than our target velocity. I strongly oppose* decelerate*, slightly oppose* coast *and favor* accelerate*. All of the available lane choices are acceptable.*

Arbiter: *After computing a scaled sum of votes for each action in the action space and eliminating vetoed actions (*shift-right*), the most popular action is* shift-left *(answer depends on the specific numerical values of the votes). Execute* shift-left*!*

From this dialogue, it is clear that the PolySAPIENT reasoning objects are myopic in their outlook. For example, the Exit Reasoning Object's votes are not influenced by the presence of the blocking vehicle; conversely, the reasoning object associated with the blocking vehicle is oblivious to the exit. Finally, the arbiter is completely ignorant of the driving task. Yet the combination of these local reasoning schemes leads to a distributed awareness of the tactical-level situation. Before discussing how a knowledge-free arbiter can combine these local views of the tactical driving task, a closer look at actions is warranted.

5.3 Actions and Action Spaces

Tactical-level actions can be defined in a road-coordinate system — as combinations of motion in lateral and longitudinal directions. These two dimensions are greatly different. In the lateral dimension: 1) positions are heavily constrained by the width of the highway, and further

by the structure imposed by lanes; 2) most vehicles remain in the same lane for long periods of time. Thus it makes sense to discuss the "utility of heading to a desired lateral position". By contrast, in the longitudinal dimension, since most cars travel at a high velocity, the "utility of being at a certain position" is an ephemeral quantity, while the "utility of moving at a certain speed" is more meaningful. These observations motivate PolySAPIENT's asymmetric representation of the two dimensions. Lateral actions map to desired positions (lateral offsets), while longitudinal actions map to desired velocities. Thus a null-action corresponds to maintaining the current lateral offset while continuing to move at the current velocity. Translating these tactical-level actions into operational-level commands is discussed in further detail in Section 5.6.

The *action space* defines the scope of tactical-level actions — unless a maneuver can be expressed by the action space, no reasoning object (or combination of reasoning objects) can express an opinion (vote) on the maneuver. While an action space could be an arbitrary set of tactical actions, both of the action spaces discussed in this thesis are discretizations of the 2-D lateral/longitudinal space described above.

PolySAPIENT's architecture imposes a number of constraints on the action space. First, since the action space is the common language between reasoning objects, its semantics must be consistent over all of the objects. For example, *shift-left* could mean either "Change one lane to the left" or "Adjust current lateral position by x meters", but its mean-

ing should be the same for all of the reasoning objects. Second, the action space semantics should not rely on hidden modes, unless those modes are observable through a sensor. Thus, if the action "shift-left" temporarily means something different *during* a lane change, all reasoning objects should be able to determine the state of this mode from some perception module. In the absence of this condition, the arbiter will blithely accumulate votes from two reasoning objects voting on the same action, when in fact they "mean" completely different things. The addition of new global modes should be avoided, since this can require changes to all existing reasoning objects. Third, the action should map clearly to operational-level commands. Since PolySAPIENT's arbiter cannot access the internals of reasoning objects, the information needed to *execute* this action cannot be hidden within the local state of a reasoning object. The following sections describe two action spaces implemented in PolySAPIENT and compare their strengths and weaknesses.

5.3.1 The Myopic Action Space

The Myopic Action Space (MAS) is a mode-less 3×3 discretization of the lateral/longitudinal space, centered on the *current* position of the intelligent vehicle (See Table 5.1). This simple action space has several clear benefits: 1) MAS' semantics are unambiguous (Compare with GAS in Section 5.3.2); 2) MAS is efficient, since reasoning objects

accelerate/shift-left	accelerate/straight	accelerate/shift-right
coast/shift-left	coast/straight	coast/shift-right
decelerate/shift-left	decelerate/straight	decelerate/shift-right

Table 5.1: The Myopic Action Space (MAS) is a 3×3 discretization of the lateral/longitudinal space. The labels are translated at the operational level into specific numbers. Thus, "left" and "right" map to lateral positions (e.g., 0.1 of a lane) while "accelerate" and "decelerate" map to changes in velocity (e.g., 0.1 m/s).

only have to consider nine possible actions. To alleviate the potential problem of jerky control (due to the coarse discretization of the action space), MAS could be extended to use a finer grid (e.g., 5×7), or the output from the tactical level could be sent through a low-pass, smoothing filter at the operational level.

5.3.2 The Global Action Space

The Global Action Space (GAS) provides a richer representation than the MAS, at the expense of additional complexity in the arbiter. The motivation is that MAS does not differentiate between small lateral motions (to adjust position within a lane) and larger motions corresponding to lane changes. In GAS, each lane is discretized into a number of sub-lanes, and reasoning objects are required to vote on the utility of the vehicle driving in that sub-lane (See Figure 5.4). While GAS is also vehicle-centric (only lateral positions in the neighborhood are considered), it is not vehicle *centered*: the position of the origin in GAS is always centered in the middle of the current lane. Whenever the

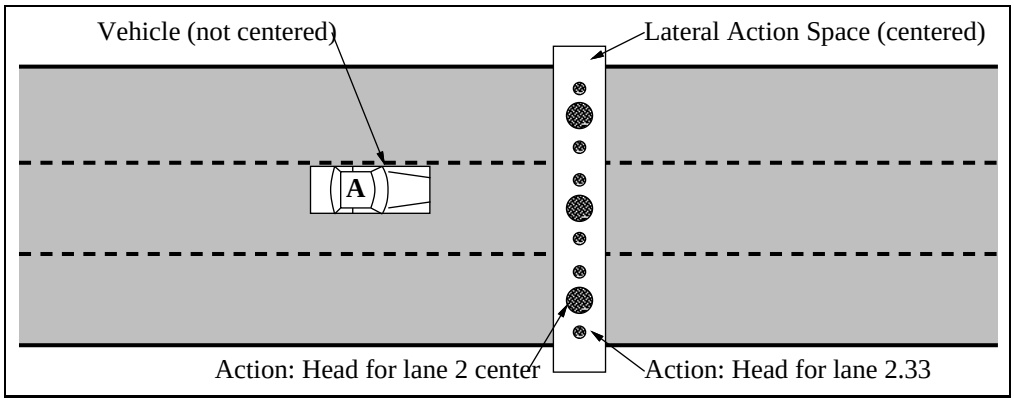

Figure 5.4: In the Global Action Space (GAS), a rich lateral representation is provided. Each lane is discretized into a number of sub-lanes, representing possible lateral positions for the intelligent vehicle. Reasoning objects are required to vote on the utility of driving in each of the sub-lanes.

vehicle changes lanes[2], GAS re-centers the coordinate frame appropriately.

The advantages of this representation include: 1) the lane centers are explicitly represented — allowing lane-trackers to vote without interpolation; 2) votes on semantically different actions are not combined — thus, the vote for "Swerve Left" (to avoid an obstacle) should not be added to "Change lanes to the left" (since driver prefers to drive in the fast lane). Besides the obvious drawbacks of requiring more processing power (since there are many more actions), the GAS scheme also introduces a serious complication.

[2]When vehicles are changing lanes, GAS is centered in the old lane until the center of the vehicle crosses the dividing line — at which point, the representation shifts to the new lane.

The primary difficulty with the GAS representation is that it is inherently ambiguous. Consider the situation shown in Figure 5.5 where the intelligent vehicle (Car A) in the left lane, is driving adjacent to another vehicle (Car B). Clearly, the reasoning object responsible for monitoring Car B will veto any attempts to change into the center lane. However, should this reasoning object also veto actions that involve the rightmost lane? One one hand, it could be argued that the utility of *moving* to the rightmost lane from the current position is bad given the current position of Car B. On the other hand, one could claim that this is not really the task of the vehicle reasoning object — after all, Car B is merely obstructing the direct route to the shadowed position: depending on the intelligence of the operational-level controller, a commanded change to the rightmost lane could be executed safely. Therefore, reasoning objects using GAS must follow consistent conventions regarding the semantics of such actions. The situation is further complicated by the fact that the different actions now take different amounts of time (especially given the option of multi-lane changes). This introduces the possibility that different reasoning objects may make significantly different assumptions about the time taken to execute an action — and this can result in voting problems, particularly when the utility of a given action is time-dependent (e.g., for vehicle reasoning objects).

Thus, to enjoy the benefits of GAS's expressiveness, a PolySAPIENT configuration must enforce stricter guidelines in the implementation of its reasoning object components. Such an implementation could also benefit from the reasoning object aggregates or the more sophisticated

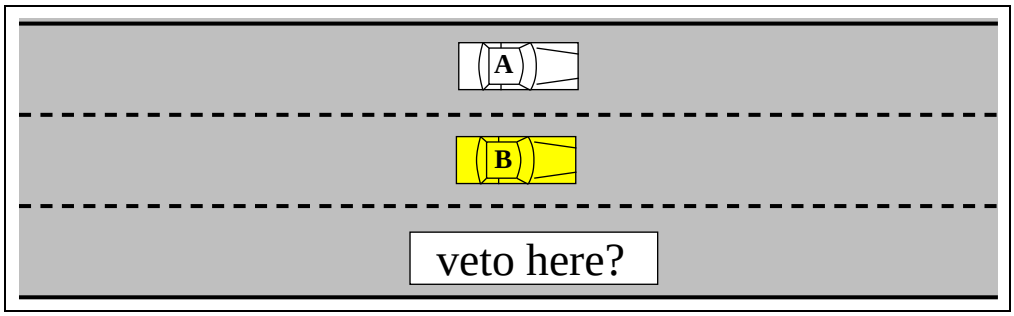

Figure 5.5: The GAS action space is semantically ambiguous. Although it is clear that the reasoning object responsible for Car B should veto lateral positions in the middle lane, it is less clear whether the same object should also veto positions in the rightmost lane (the areas in Car B's "shadow").

arbitration scheme proposed in Section 5.9. However, the implementation of PolySAPIENT whose results are reported in Chapter 7, uses the simpler (OAS) action space.

5.4 Action Selection

In any decentralized system, the issue of command fusion is critical. The recommendations from different modules are often contradictory, and these contradictions need to be resolved since the robot can only perform a single action at any given time. Consider the scenario shown in Figure 5.6. Here, the intelligent vehicle is driving in the center lane when Car B, a slow-moving vehicle abruptly begins changing lanes (possibly because Car A may have been in Car B's blindspot). Unless Car A reacts by braking or swerving, a collision is likely. Two modules

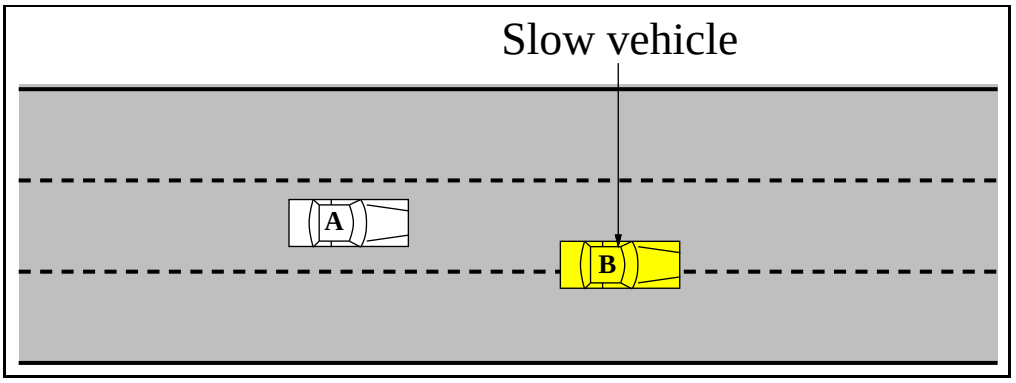

Figure 5.6: The intelligent vehicle (Car A) is driving in the center lane when a slow-moving vehicle (Car B) abruptly begins switching lanes. In a distributed reasoning scheme, there is a conflict between lane-tracking and obstacle-avoidance. Different command fusion schemes address this emergency in different ways.

are relevant to this scenario: a lane-tracking module, and an obstacle-avoidance module. In this situation, the two modules may recommend conflicting actions (e.g., stay in this lane vs swerve). The command arbiter must integrate these inputs into a coherent response. A number of different command fusion methods have been developed to address this issue.

5.4.1 Related Work in Command Fusion

In the subsumption architecture [18], all modules (known as behaviors) are active at all times and process the sensor data. Each behavior *individually* determines whether it is applicable to the current situation — at which point it sends control commands to the robot.

Conflicts between modules are resolved through a hard-wired priority scheme (inhibition links) leading to the *subsumption* of one behavior by another. Thus, arbitration is not explicitly represented as a module. In the scenario shown in Figure 5.6, the lane-tracking module would initially be in control, keeping the intelligent vehicle centered in its lane. The obstacle-avoidance module, would be passively monitoring the situation in its quiescent state. As Car B started changing lanes, the obstacle-avoidance module would activate (without any centralized decision) and override the lane-tracker's recommendations. Due to this, the intelligent vehicle would swerve or brake (as determined by the obstacle-avoidance behavior), and hopefully the emergency would be averted. At this point, the obstacle-avoidance module would return to its quiescent state, enabling the lane-tracking behavior to regain control. Note that during the emergency, unless the obstacle-avoidance module is also somewhat competent at lane-tracking, the intelligent vehicle runs the risk of running off the road. This example highlights subsumption's strengths (decentralized control) but also exposes some of its limitations as a command fusion scheme for tactical driving. First, the complete overriding of one behavior by another is not suitable for a highway's structured environment — since there is no guarantee that the response to one emergency condition (potential collision) will not result in a second emergency (running off the road). Second, the architecture requires the designer to create the inhibition links manually. While this layering of behaviors may be straightforward for an insect-robot exploring an unstructured environment, the

strategy scales poorly for an intelligent vehicle driving in traffic. Any time a new module is added to the system, the designer must carefully determine its interactions with all other behaviors, and create the necessary inhibition links. Since the arbitration is not localized (unlike PolySAPIENT's single voting arbiter), this modification process is non-trivial.

The potential field techniques [58, 53], fuzzy control [49, 126] and schema-based methods [6] use a different approach to command fusion: the arbiter does not select between the outputs of different modules, but rather *combines* them. In many of these approaches, commands are represented as vectors in some control space (e.g., velocity or acceleration vectors) and commands are combined by a straightforward, scaled vector sum in this space. Thus, in the example shown in Figure 5.6, the lane-tracking behavior would output a vector aligned with the road, while the obstacle-avoidance module's repulsive field would generate a vector pointing away from Car B. The arbiter would combine the two vectors by adding them — creating a vector that would simultaneously slow the intelligent vehicle and cause it to swerve to the left. Two important advantages of such methods over subsumption are that: 1) the arbitration need not be hand-crafted — provided that all modules are able to use the vector representation, combining the outputs is straightforward; 2) control of the robot does not abruptly move from one module to another. Furthermore, since the command fusion does not use any domain-specific information, adding new behaviors to such a system is relatively easy. Unfortunately, the naive combination of

commands using vector sums leads to some serious problems in the tactical driving domain. First, some of these methods (notably potential field approaches) are prone to local minima [84]. For example, a force motivating a vehicle to speed up (to finish an overtaking maneuver) may balance a decelerating force due to a speed limit — causing the intelligent vehicle to drive in the blocker's blind spot. Second, the semantics of vector addition do not always result in a valid solution. This point is best illustrated by the example of a mobile robot driving along a road which forks. Assume that one module prefers the right fork (+1, +1) while the other prefers the left fork (+1, -1). The simple addition of these two command choices would result in the vector (+1, 0) which corresponds to driving straight — between the two forks, and thus off the road.

PolySAPIENT's command fusion approach is largely motivated by the Distributed Architecture for Mobile Navigation (DAMN) [84], an arbiter used successfully to integrate a variety of cognition modules in several Navlab II [106] cross-country navigation projects. In DAMN, different modules express preferences for vehicle actions (commonly encoded either as raw steering curvatures, or as vehicle positions in a local map) with positive and negative votes. For example, DAMN can combine the outputs from a grid-based path planner [97] and a laser rangefinder-based obstacle-avoidance module [60] to navigate the Navlab II through an unstructured environment. The voting arbiter approach addresses the deficiencies with the other command fusion schemes discussed above. Furthermore, since all of the modules are

connected directly to the arbiter, the modules are independent. Thus, the voting arbiter encourages incremental development of reasoning modules. The main differences between DAMN and PolySAPIENT's arbiter are:

- PolySAPIENT relies on vetoes to allow one reasoning object to override positive votes for actions by a group of (less knowledgeable) objects.

- In DAMN, votes are generated by *behaviors*, which are abstract modules associated with driving. In PolySAPIENT, votes are generated by reasoning objects — which are always connected with a physical entity in the environment.

- DAMN is optimized to operate in a *stationary* environment (one without other moving objects). This allows DAMN to reason about the relative utilities of different vehicle *positions*, since these utilities are presumably constant over time. In tactical driving, this is obviously not possible.

The following sections explore the details of PolySAPIENT's command fusion arbiter.

5.5 Knowledge-Free Arbitration

The simplest voting strategy is for each reasoning object to select the action which it believes best suits the current situation. Given a large number of reasoning objects and a small number of possible actions, it is possible to create a knowledge-free arbitration scheme that would select the most popular action for execution. Unfortunately, this straightforward scheme has the drawback that reasoning modules are restricted to choosing only one favorite action. Neither does this scheme allow reasoning objects to express the strength with which they favor a specific candidate action.

Thus, most voting arbiters extend the idea in two ways. First, each module is required to provide a vote (either supporting or opposing) for *every* candidate action. Second, the magnitude of each vote reflects the strength with which the module favors the given action. The total number of votes available to a module is generally unrestricted — allowing a module to support multiple candidate actions, if desired. The incoming votes from different modules can also be scaled (to give preference to important modules). As observed above, the PolySAPIENT arbiter accepts vetoes in addition to votes. Thus, the vehicle reasoning object responsible for monitoring the car ahead can prevent the *accelerate* action even if other reasoning objects are in favor of the action.

Consider the scenario shown in Figure 5.2. The votes generated by the various reasoning objects in that situation is shown in Figure 5.7.

In this example, the reasoning object associated with monitoring the blocker strongly votes against speeding up in the current lane (since this would create an unsafe headway) while also voting against staying in the current (tailgating) position. The exit reasoning object, recognizing that the desired exit is approaching, votes for right lane changes while opposing left lane changes. And the desired velocity object, oblivious to the blocker ahead, votes to speed up. The votes submitted by a reasoning object for each action are scaled by weights corresponding to the importance of that object before being added together. Thus, the negative votes for speeding up are amplified because of the vehicle reasoning object's importance, while the positive votes for speeding up are diminished. Once all the votes associated with each action have been tallied, the action with the highest accumulated vote is selected.

Decentralized vote-based reasoning introduces several complications which must be considered. First, the semantics of a vote need to be consistent across all reasoning objects. Second, the amount of domain-specific knowledge to be included in the arbiter must be determined: too much domain knowledge in the arbiter resurrects the problems of a monolithic system (See Chapter 4) but too little leads to a lack of consistency. Third, the synchronization issue must be addressed: if various reasoning objects are operating asynchronously, how can one guarantee that all of the votes were generated for the same world state? Finally, the problem of silent reasoning objects must be handled: can one safely ignore a reasoning object in any given time-step? PolySAPIENT assumes: 1) that perception modules are responsible for guarantee-

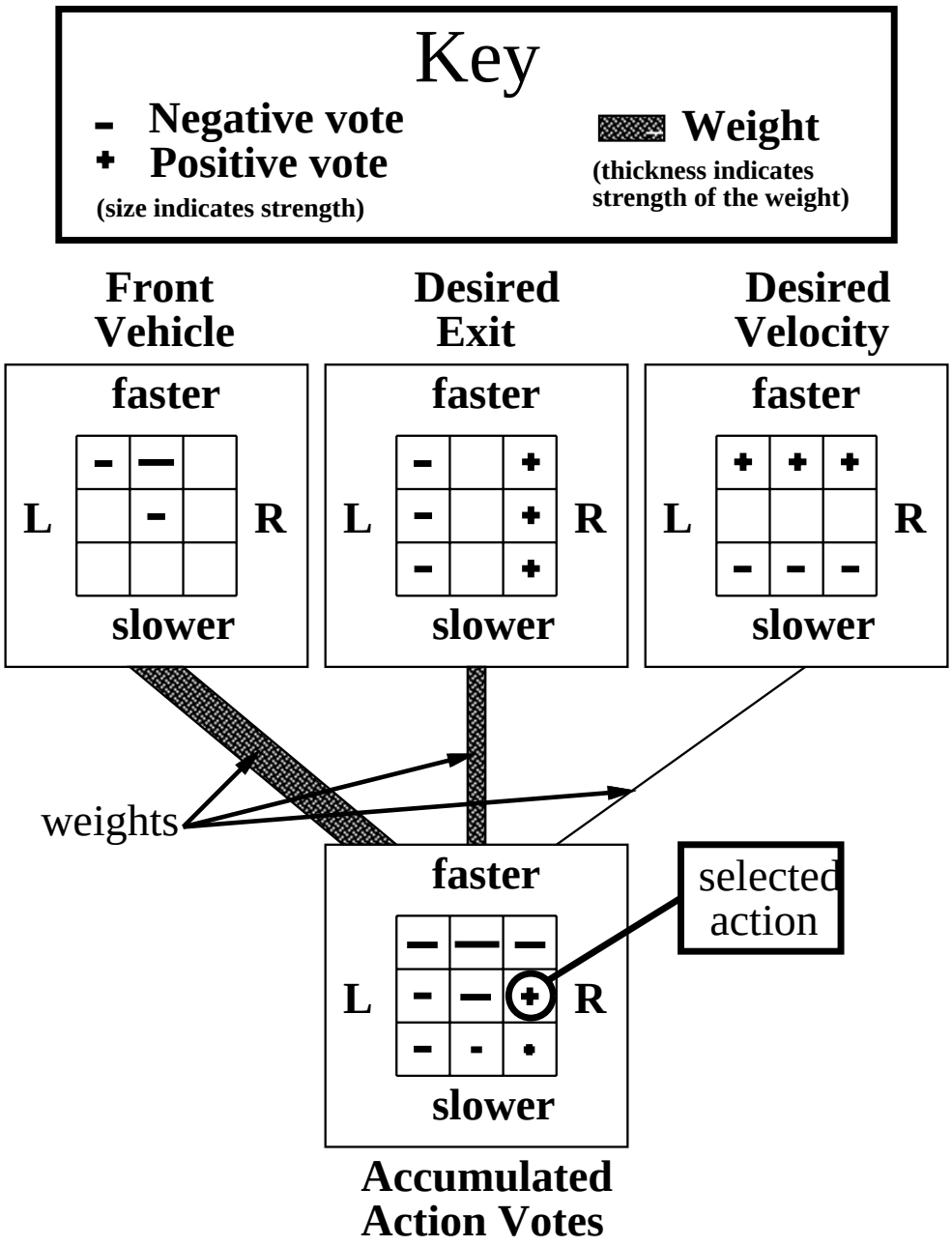

Figure 5.7: The PolySAPIENT arbiter selects an action by combining votes from various reasoning objects. The weight associated with each reasoning object is used to scale the votes, and the weighted votes for each action are added. The action with the highest accumulated vote is selected for execution.

ing consistency in the observed world state across reasoning objects; 2) that all reasoning objects will respond within the alloted time.

5.6 Action Execution

As discussed above, tactical-level maneuvers are expressed in a lateral/longitudinal action space (local road coordinates). Thus, they must be translated to the vehicle-centric representation used by operational-level controllers. This section describes how the current implementation of PolySAPIENT converts its actions into a Navlab-compatible form (desired steering curvature and velocity).

5.6.1 Lateral Control

The operational-level road-following in a Navlab vehicle is typically performed by a lane-tracker such as ALVINN [73] or RALPH [74]. Both of these vision-based systems rely on a low-latency loop to maintain the vehicle's desired lateral position. Feeding each curvature command through the tactical-level processing is impractical since any delays in reasoning would jeopardize basic vehicle safety.

PolySAPIENT's paradigm for lateral control is to *manipulate* the active lane-tracking system's steering behavior. Although the neural-net

based ALVINN system cannot accept direct requests for a lateral position, its implicit pure-pursuit steering model [114] provides a convenient interface for this purpose. By laterally moving the pure-pursuit point in the desired direction over time, the vehicle can be instructed to change lanes. RALPH uses a template-based approach with an explicit model of road geometry. In this case, PolySAPIENT's lateral preferences can be explicitly stated. The current (simulated) implementation in SHIVA uses a pure-pursuit manipulation interface.

5.6.2 Longitudinal Control

Speed control for intelligent vehicles is a non-trivial operational-level problem. Currently, the interface to the Navlab vehicles does not allow direct control over the vehicle's throttle and brake. These functions are abstracted through a *desired velocity* value. Given a speed, the Navlab's operational-level controller will attempt to smoothly control the vehicle to attain the desired value in an unspecified (but hopefully short) period of time. This controller motivates the model currently used for longitudinal control in PolySAPIENT. PolySAPIENT's tactical-level actions are mapped into desired velocities, and processed to ensure that they lie within the acceleration constraints of the intelligent vehicle.

5.7 Reasoning Object Complications

Since PolySAPIENT is a distributed system, consisting of a number of independent components, driving performance may suffer not only from either inadequate reasoning objects, but also from undesirable interactions between otherwise correct reasoning objects. Some, such as inconsistent vote semantics, or action space ambiguities, were discussed in Section 5.3. Others are described below, in the context of the following case study:

Symptom: PolySAPIENT vehicles weaved within lanes, exhibiting a *fish-tailing* behavior. These oscillations occurred most frequently during low-traffic conditions, and while these lateral oscillations did not seem to cause accidents, they did result in a bizarre driving style (See Figure 5.8).

Diagnosis: Examination of the individual reasoning objects, using the diagnostic tools in SHIVA, showed that they were all operating correctly. Due to the low-traffic conditions, most of the reasoning objects were passively monitoring the situation and making no recommendations (zero votes over the action space). The action selection arbiter would thus select actions based upon very little input, with the chosen action winning only by a small margin. This was not, in itself, a problem since a number of actions were equally good in the situation. Unfortunately, given the stateless nature of the arbiter, it was unlikely that the same action would

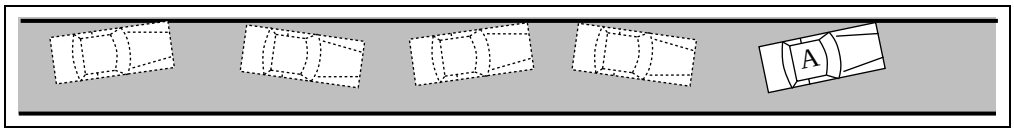

Figure 5.8: Initial implementations of PolySAPIENT vehicles displayed unexpected weaving behavior, as depicted in the diagram above. Architectures with stateless action selection schemes are sensitive to noise in the incoming votes. See the text for solutions to this problem.

be chosen during consecutive time-steps, due to small fluctuations in the voting space. This led the intelligent vehicle to wander aimlessly within its lane.

The lack of consistency in action selection, when combined with controller latency, is very dangerous. Consider the situation shown in Figure 5.9. Here, the intelligent vehicle (A), can swerve to avoid the blocker on either the left or right side. In the absence of other traffic, both actions seem equally good (assuming no intrinsic bias to overtake on the left). Assume that the arbiter opts to initially *change-left*. There is a delay before this action results in a perceivable change in lateral position (since the command is executed as a change in the operational-level lane-tracker's pure-pursuit point). Thus, in the next time-step, it is quite possible (due to noise), that the stateless arbiter may opt to *change-right*. The arbiter is unaware that this effectively *cancels* its decision to overtake on the left from the previous time-step. Rather than smoothly overtaking on either the left or the right side, this dithering prevents the intelligent vehicle from passing. In the worst case, Car A may be unable to decelerate in time, causing a

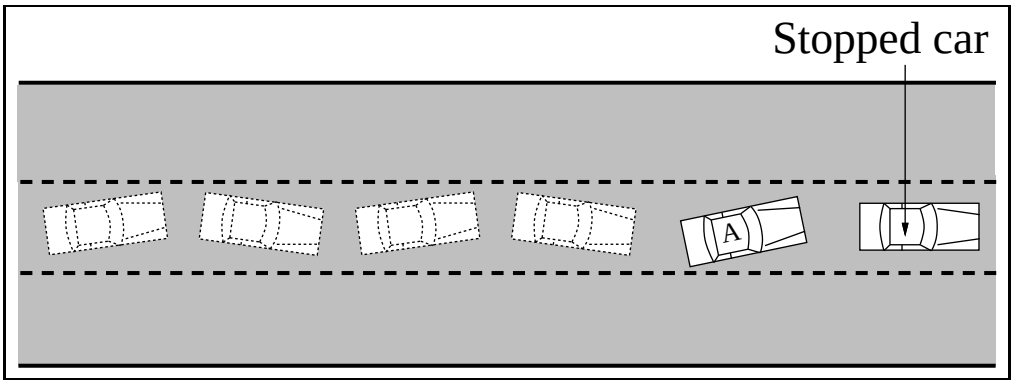

Figure 5.9: The combination of controller latency and inconsistent action selection can cause a collision in this scenario. Since lateral actions affect the lane-tracker's pure-pursuit point, there is a delay before the effects are noted by a change in the vehicle's actual lateral position. During this interval, the votes for *change-left* and *change-right* are both equally good. Unless the arbiter consistently recommends the same action, the vehicle will not successfully change lanes in time.

collision.

This problem cannot be addressed at the reasoning object level since the individual reasoning objects are not aware of the *outcome* of the previous time-step's voting. The solutions below examine how consistency was incorporated into PolySAPIENT without sacrificing the independence of the reasoning object paradigm.

Solutions: One solution to this problem is to introduce hysteresis into the system. Conceptually, the output from the previous time-step's action selection should influence the current time-step's voting. This can be done either through an additional reasoning object which votes for the previously executed action, or by modifying the arbiter to internally bias the voting in this manner. The

latter approach is used in the current implementation of PolySAPIENT.

Two methods of adding hysteresis were investigated. The first used the output vote matrix from the previous iteration, decayed by a constant:

$$O_t = kO_{t-1} + V_t \tag{5.1}$$

where, O_t is the output vote matrix at time-step t, k is the decay constant, and V_t is the incoming vote matrix (from the reasoning objects). This creates a leaky integrator for each action's votes, and favors actions that have been receiving positive votes in recent history.

The second method only used the *winning* vote from the previous iteration:

$$\begin{aligned} O_t(a) &= kO_{t-1}(a) + V_t(a) \quad \text{if } a = \alpha_{t-1} \\ O_t(a) &= V_t(a) \qquad\qquad\qquad\quad \text{otherwise} \end{aligned} \tag{5.2}$$

where, $O_t(a)$ is the output vote corresponding to action a at time t, $V_t(a)$ is the incoming vote corresponding to action a at time t, k is the weight decay constant, and α_t is the winning action at time t. This method favors the "incumbent action" (the action that is currently being executed by the vehicle) without rewarding any of the other candidates.

Experiments showed that the second method was more effective in curbing arbiter inconsistency. This is because the second approach clearly favors *consistency* by amplifying the gap between the best and next-best actions, ensuring that small fluctuations

in the votes do not cause the arbiter to vacillate between the top actions.

Discussion: The problem of action selection is vital to any distributed system, and the price for the convenience of a knowledge-free arbiter is that the system can behave poorly even with well-designed components. The problem of arbiter vacillation is not limited to tactical driving. Fortunately, the solution outlined above is general, and may easily be used in other domains.

Hysteresis is not a panacea for the problems of distributed system arbitration: it can introduce other undesirable behaviors. One potential problem is that the arbiter's response becomes more sluggish since old actions continue to influence the robot's current behavior even after the situation has changed. This can manifest itself as overshooting the desired goal. For example, in the MAS representation, the action of centering the vehicle in its lane is not explicitly represented: rather, the lane reasoning object votes for incremental right and left lateral adjustments (all measured relative to the vehicle's current position). Here, badly tuned hysteresis parameters can prevent the vehicle from locking onto the lane center since the adjustment action will come too late. Note that these problems would not be present in a richer representation such as GAS[3].

[3]However, as seen earlier, richer representations introduce new complications.

5.8 Independence Assumption Violations

Since the PolySAPIENT architecture is based on the assumption of local independence (See Section 5.2.1), it is natural to explore the conditions under which this assumption fails — and what impact this has on PolySAPIENT's behavior.

The best example of such a complication comes from outside the highway domain: negotiating a "Right turn on Red" at a city intersection [51] (See Figure 5.10). Here, the intelligent vehicle, Car A, is monitoring traffic on the cross street before turning right. The relevant instantiated reasoning objects are: 1) a vehicle reasoning object monitoring Car B; 2) a reasoning object watching the state of the light[4]; 3) lane-tracking objects for the lanes of interest. This example also obviously assumes that the action "turn right at the intersection" is representable in the action space.

In this situation, Car A's decision should be based on both the state of the light at the intersection, and on the presence of traffic on the intersecting road; more precisely, it should depend on the *relation* between two physical entities in the environment — something that is not currently represented explicitly. The best solution given the current architecture is as follows. First, the vehicle comes to a complete stop based solely on the state of the traffic light. Second, it makes the

[4]This type of reasoning object has not been implemented since traffic lights are not present in the highway domain.

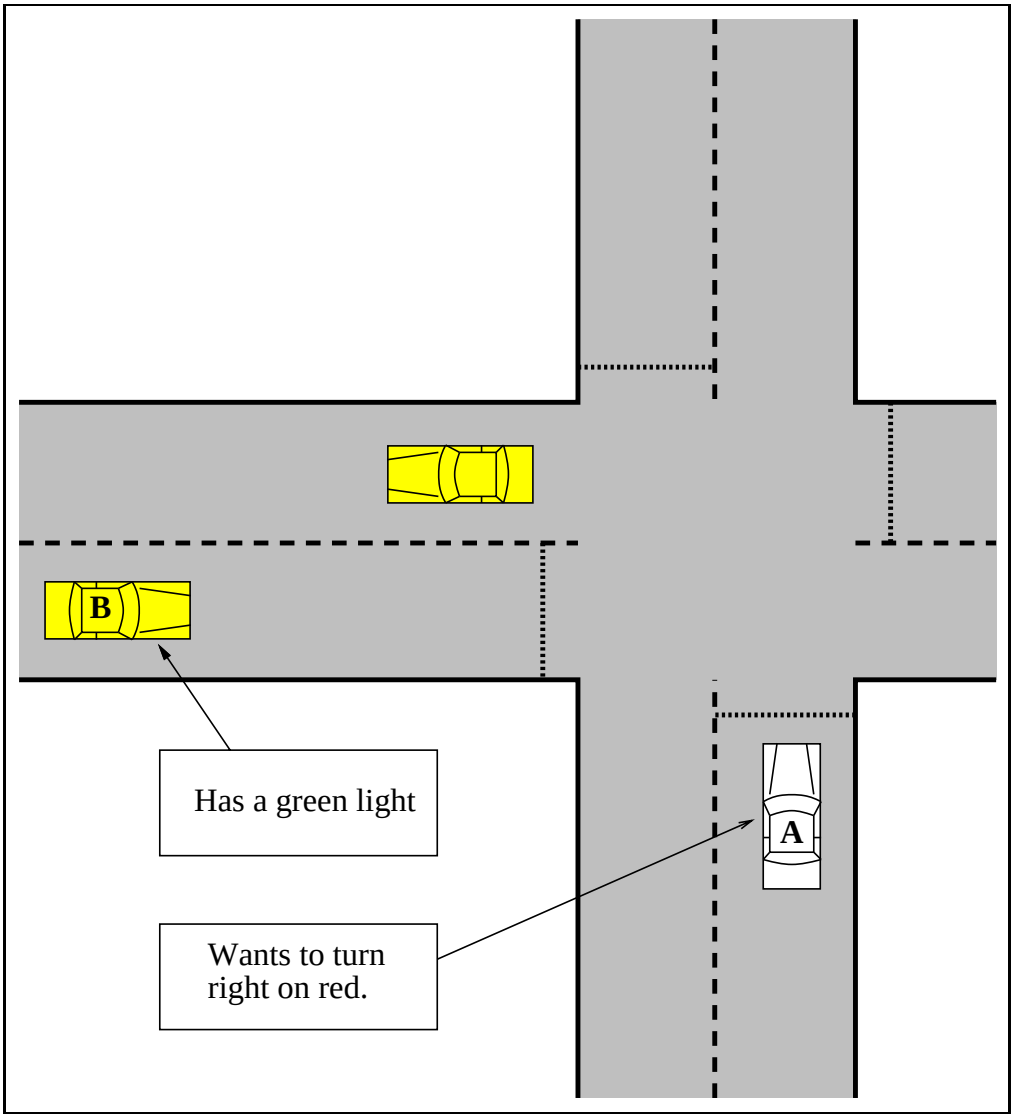

Figure 5.10: Situation showing a possible violation of the independence assumption: in this city-street scenario, Car A wants to make a right turn on red. See text for details.

decision to turn right based solely on Car B's reasoning object's recommendations. This is possible since the state of the light no longer constrains the vehicle once it has come to a complete stop. Conversely, the option of turning right is not considered before vehicle has stopped at the intersection. Capturing the relationship between the two entities requires extensions to the architecture, as discussed in the next section.

5.9 Extending the PolySAPIENT Paradigm

As mentioned above, the current architecture used by PolySAPIENT is unable to explicitly model relations between two physical entities in a clean manner. One easy solution is just to create a new reasoning object type for the relation, and have its votes influence the available actions. While this may be feasible in certain situations, such an approach has the potential of causing an explosion of new reasoning objects, with unclear interactions. Another approach is to create hierarchies of reasoning objects, known as *aggregates*. A reasoning object aggregate would be instantiated by perception modules, in much the same way that individual reasoning objects are instantiated on demand. The reasoning object aggregate would then accept inputs from its component reasoning objects performing an internal form of action selection (potentially using a different, internal, action space). The results of this arbitration would then be passed to the external arbiter, as if the reasoning object

aggregate were a single reasoning object.

A second place for possible work in PolySAPIENT is a more sophisticated arbitration mechanism. The current arbiter is both knowledge-free and stateless[5]. With a richer action space, such as GAS (See Section 5.3.2, it may be desirable to incorporate some amount of planning to the arbiter. Now, the action space would be interpreted as a set of short-term goals to be achieved by a sequence of operational-level commands. This thesis does not further explore the potential of these extensions.

5.10 Discussion

Although PolySAPIENT's independence assumption may seem to create a very disconnected implementation of situation awareness, this methodology is nevertheless quite consistent with the SA literature. For instance, in Endsley's model, the cognitive processing involved in SA is divided into three levels [25, 26]. At the *information level*, the agent detects the target cue or object in the environment. This corresponds to PolySAPIENT's perceptual processing where reasoning objects are instantiated in response to observed traffic entities. At the *interpretation level*, this information is "processed and integrated into an assessment of the situation". In PolySAPIENT, the "processing" is

[5] Some notion of state is provided by the hysteresis reasoning object.

performed by individual *reasoning objects* while "integration" occurs in the voting arbiter. At the *prediction level*, the agent projects how the objects in the environment will behave in the near future. In PolySAPIENT, the prediction of an entity's future interactions is performed locally by its appropriate reasoning object. Thus, while no central element in PolySAPIENT is aware of the complete traffic scenario, one can consider the *collection* of currently instantiated reasoning objects as possessing the distributed equivalent of Endsley's situation awareness.

The reasoning object paradigm is well suited for addressing a complex task such as driving in traffic. By associating each reasoning object with a physical entity in the environment, PolySAPIENT avoids creating a large group of interacting behaviors, building a fairly small set of largely independent modules. If the set of modules does not adequately span the problem space (e.g., if there is no reasoning object dealing with exits), then clearly PolySAPIENT will not perform well when such an entity occurs in the environment — since that entity will effectively be invisible. Worse, if two reasoning objects are incorrectly bound to the same entity, and the two are not in complete agreement, PolySAPIENT will develop a split personality. To avoid these problems, most PolySAPIENT configurations maintain a one-to-one mapping between visible entities and objects.

Since developers are free to implement different reasoning objects in different ways, it is likely that each type of reasoning object's behavior

will be somehow dependent on a number of internal magic numbers, or parameters. These parameters correspond to thresholds in a decision tree, weights in a neural network, or gains in a classical controller. Furthermore, as discussed in Section 5.5, the votes coming from different reasoning objects are scaled by different amounts. Unfortunately, this flexibility means that PolySAPIENT's behavior potentially depends on a large number of parameters. Manual tuning of these parameters can be a tedious and error-prone task since parameters can lie dormant in an infrequently used reasoning object – manifesting themselves as undesirable behavior only in rare situations. The next chapter describes an automated solution to this problem: learning.

Chapter 6

Learning Situation Awareness

> *By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest.*
> *—Confucius*

The fact that humans learn to drive, and improve with experience suggests that an intelligent vehicle could acquire situation awareness through repeated exposure to traffic situations. Although machine learning has been successfully applied to operational level tasks, such as lane-tracking [73, 74] and trailer-truck docking [119], very few systems have applied learning to tactical-level driving.

6.1 Applying Learning to Tactical Driving

Learning strategies can be broadly classified into two types: supervised and unsupervised. In the former, the system relies on a teaching signal provided by an external source. Examples include backprop neural networks [93, 119] which are given input and output mappings for the examples in a training set, and genetic algorithms [39] which are given feedback through a user-supplied evaluation function. By contrast, unsupervised techniques discover inherent patterns in data without outside help. In this chapter, I describe how learning can be used to improve the performance of an existing tactical driving system without learning the task from scratch. I also show how the same techniques can be used to identify bad configurations of reasoning agents.

6.2 PBIL: Evolving Intelligent Driving

Population-Based Incremental Learning (PBIL) is an evolutionary algorithm related to both competitive learning and genetic algorithms [14]. The object of the algorithm is to create a real-valued *probability vector* which, when sampled, reveals high quality candidate solutions with a high probability. Like a GA, PBIL optimizes a black-box evaluation function by iteratively sampling the function at a number of points in parallel. Unlike a GA, however, PBIL does not maintain a persistent population of candidates — at the start of each new generation, the old

population is entirely discarded and a new population is created from the statistics of previous high-scoring candidates (represented in the probability vector). At the end of each generation, the evaluation received by the best candidate in that generation is used to update the probability vector in such a way that, over time, the vector generates better candidate solutions. The remainder of this chapter, focuses on the details of the PBIL algorithm, and how it is used in PolySAPIENT to learn situation awareness (through simultaneous, distributed parameter optimization).

6.2.1 Internal Representation

A prerequisite to any discussion of a learning technique is a look at the method's problem encoding. In PBIL, candidate solutions are represented as bit-strings of constant length. Since the domain-specific semantics of the bit-string are ignored by the PBIL algorithm, users may select any encoding which maps points in the search space to a sequence of '0's and '1's. In many problems involving continuous domains, this involves a quantization of the problem space into discrete regions at an appropriate resolution. Section 6.3.2 presents the encoding scheme used in PolySAPIENT; the current section illustrates the operation of the PBIL algorithm on abstract bit-strings and assumes that a suitable mapping from the problem space to this internal representation is available.

6.2.2 The Probability Vector

As mentioned above, candidate strings in PBIL are generated from a probability vector which encodes the statistics of high-scoring bit-strings from previous generations.

In each generation, the l-bit candidate strings are created by sampling the l-length real-valued vector of the form:

$$P = (p_1, p_2, \ldots, p_i, \ldots, p_{n-1}, p_n) \tag{6.1}$$

where every p_i (a number between 0 and 1), is the probability that a candidate solution generated from the vector P will contain a '1' in its i-th place. For example, the vector

$$P = (0.5, 0.5, 0.5, 0.5) \tag{6.2}$$

will produce four-bit strings with an equal probability of a '0' or a '1' in each position. By contrast, the four-bit strings produced by the vector

$$P = (0.99, 0.99, 0.99, 0.01) \tag{6.3}$$

are likely to contain to contain a '1' in each of the first three positions and a '0' in the final position. Since each sampling of the probability vector generates one candidate string, n independent samples are needed to fill a generation with population size of n. The probability vector is initialized to $(0.5, \ldots, 0.5)$ at the start of the first generation — resulting in a uniform distribution of initial candidate strings. Over time, the incremental updates of the probability vector result in

a skewed distribution (like the one in Equation 6.3) which is likely to produce candidate strings that score highly on the evaluation function.

6.2.3 The Algorithm

The important steps in the PBIL algorithm can be summarized as follows:

- A population of candidate solutions is generated from the probability vector (as a set of bit-strings).
- Each candidate is evaluated by the black-box function. This represents the external feedback from the problem domain.
- The probability vector is slightly perturbed in the direction of the best candidate. Thus, if the candidate solution had a '1' in its i-th place, the element p_i in the probability vector is increased slightly; and conversely, if the best candidate had a '0' in the i-th place, p_i is decreased slightly (since p_i is a probability, it is always constrained to remain between 0 and 1).
- All of the candidate solutions are destroyed, and the algorithm returns to the first step with the updated probability vector. This iterative procedure is repeated until the researcher is satisfied with the performance of the candidate solutions.

```
****** Initialize Probability Vector ******
for i := 1 to LENGTH do P[i] = 0.5;

while (NOT termination condition)
      ****** Generate Samples ******
      for i := 1 to SAMPLES do
            sample_vectors[i] := generate_sample_vector_according_to_probabilities(P);
            evaluations[i] := Evaluate_Solution( sample_vectors[i]; );
      best_vector := find_vector_with_best_evaluation( sample_vectors, evaluations );

      ****** Update Probability Vector towards best solution ******
      for i := 1 to LENGTH do
            P[i] := P[i] * (1.0 − LR) + best_vector[i] * (LR);

      ****** Mutate Probability Vector ******
      for i := 1 to LENGTH do
            if (random (0,1) < MUT_PROBABILITY) then
                  if (random (0,1) > 0.5) then mutate_direction := 1;
                  else mutate_direction := 0;
                  P[i] := P[i] * (1.0 − MUT_SHIFT) + mutate_direction * (MUT_SHIFT);


USER DEFINED CONSTANTS (Values Used in this Study):
SAMPLES: the number of vectors generated before update of the probability vector (100)
LR: the learning rate, how fast to exploit the search performed (0.1).
LENGTH: the number of bits in a generated vector (3 * 20)
MUT_PROBABILITY: the probability of a mutation occuring in each position (0.02).
MUT_SHIFT: the amount a mutation alters the value in the bit position (0.05).
```

Figure 6.1: The basic PBIL algorithm.

The full algorithm is given in Figure 6.1. PBIL, as described here is characterized by two parameters: population size, and learning rate. The former refers to the number of candidates generated from the probability vector in each generation, and the latter determines how far the probability vector is perturbed in each generation. Intuitively, one can see that a larger population size results in a better exploration of the search space (at the expense of speed) and that a larger learning rate can lead to faster convergence (at the risk of overshooting). Note that at any point in the algorithm, the probability vector can be viewed as a prototype vector for generating solution vectors which return high evaluations given the available (current) knowledge of the search space. Over successive generations, as different candidates explore the prob-

lem space, this knowledge grows and the probability vector becomes increasingly accurate at representing the important characteristics of good solutions for the given domain.

The basic PBIL algorithm can be enhanced in a number of ways to improve its performance in specific domains. The two extensions used in PolySAPIENT are *elitist selection* and *mutation*:

elitist selection: Since the population is stochastically recreated in each generation, the basic PBIL algorithm cannot guarantee that the best candidate from the previous generation (used to perturb the probability vector) will reappear in the current generation. This problem, exacerbated in implementations using small population sizes, causes PBIL to unnecessarily explore unproductive areas of the search space. Elitist selection refers to the explicit preservation of the best bit-string from one generation to the next.

mutation: As PBIL learns, the probability vector becomes skewed and the population starts to stagnate — with the same bit-string appearing multiple times. While this redundancy is beneficial in stochastic domains (where the same bit-string may return different evaluations), the lack of diversity inhibits exploration of the search space. Mutation refers to the small chance that a particular member of the population will experience a bit-flip at a random position. While most mutated strings will perform poorly, it is hoped that a fortuitous mutation may result in improved per-

formance.

Both of these extensions have also been used successfully in conjunction with other learning techniques such as genetic algorithms. A discussion on the similarities and differences between PBIL and genetic algorithms, as well as an empirical comparison of performance on a suite of standard problems may be found in [12].

6.3 Applying PBIL to PolySAPIENT

PBIL's domain-independent qualities have enabled researchers to apply it to a variety of optimization problems including figure layout [11], registration in medical robotics [13] and neural network weight training [10]. Unlike many supervised learning methods, evolutionary algorithms like PBIL do not need explicit models (or examples) of correct behavior (i.e., competent tactical driving) since good solutions are automatically discovered through unguided, stochastic exploration. Furthermore, since PBIL treats the system as a black-box, incorporating PBIL into an existing system does not require major changes to the system's architecture. These reasons motivated the decision to use PBIL for parameter optimization in PolySAPIENT.

The tactical-driving domain challenges PBIL in a number of ways. First, the environment does not give immediate feedback to tactical-level ac-

tions (delayed rewards and penalties). Second, since a vehicle's decisions depend on the behavior of other vehicles which are not under its control, the same bit-string can receive different evaluations under different conditions. Third, the PBIL algorithm is never exposed to all possible traffic situations (thus making it impossible to estimate the "true" performance of a PBIL string). Finally, since each evaluation takes a considerable time to simulate, the PBIL algorithm must find good parameter values within a reasonably small number of evaluations. The solutions to these challenges are described in the following sections.

6.3.1 The Evaluation Function

One of the main attractions of using machine learning to solve difficult problems is that it is often easier to *recognize* good solutions than it is to manually develop an algorithm to address the task. In some domains, immediate feedback on the worth of an action is available. This is especially true if the system is receiving instruction from a human teacher. For example, in ALVINN [73], the difference between the lane-tracker's steering direction and the human driver's steering command is used to generate an error vector. By contrast, in the experiments described here, the absence of an instructor prevents the simulator from assigning an evaluation to individual tactical-level actions. However, the ac-

cumulated effects of actions, such as missed exits, near-collisions[1], and erratic lane-keeping are observable — and can be used to evaluate a given choice of parameters.

The following evaluation function combines these observable elements into a weighted sum to quantify the performance of a PolySAPIENT-driven vehicle:

$$E = -10000\nu - 1000\kappa - 500\beta - 0.02\chi - 0.02\lambda + \zeta \quad (6.4)$$

where:

- E is the evaluation score quantifying the performance of the vehicle.
- ν is a flag: zero under normal circumstances, and one if the arbiter ever receives a veto for *all* actions. This factor penalizes cognition modules which "freeze up" in difficult situations.
- κ is the number of near-collisions involving the vehicle.
- β is a flag: zero under normal circumstances, and one if the vehicle exited prematurely, or otherwise missed its designated exit.
- χ is the difference between the desired and actual velocities, integrated over the scenario run. It represents whether the vehicle was able to maintain the target velocity.

[1]A near-collision is defined to occur when the distance separating two vehicles is smaller than a certain threshold: e.g., 0.5m laterally and 2.0m longitudinally.

- λ is the vehicle's lane deviation[2], integrated over the scenario. This serves to penalize bad lane-keeping.

- ζ is the distance traveled by the vehicle in the scenario. This incremental reward for partial completion of scenarios helps learning. ζ will not increase once the vehicle misses its designated exit.

Some subtleties in the evaluation function are worth noting. First is the incremental reward term, ζ: the absence of which greatly hinders learning, as seen in the following example. Consider a candidate string which is almost optimal, deficient only in a critical parameter related to the exit reasoning object. When simulated, a vehicle with this candidate string may successfully negotiate traffic all the way to its desired exit — and then ignore the exit. Without the incremental reward term, such a vehicle would get an evaluation no better than a (very poor) vehicle which mistakenly took the first available exit. For PBIL to learn that these two candidate strings are qualitatively very different, despite their similar evaluations, is a challenging (and unnecessary) task. Incremental rewards allow the researcher to favor traits that are positively correlated with the task to be learned. For example, the distance covered by a vehicle is a measure of the vehicle's ability to negotiate traffic and avoid taking the wrong exit — certainly a trait to be encouraged. Second, the accumulated penalties for speed and lane deviation are divided by the distance traveled. Without this normalization, vehicles that did not travel very far (i.e. took an early exit) would

[2]Lane deviation is defined as the distance to the closest lane center.

be implicitly rewarded for accumulating less of a deviation. Finally, the evaluation function is noisy from PBIL's viewpoint since a given bit-string, which maps onto a set of PolySAPIENT parameters, can receive markedly different evaluations from different simulations. For example, in one case, congestion at the desired exit could prevent the learning vehicle from exiting; in another case, smoothly flowing traffic could allow the same bit-string to receive a very high score. To minimize the impact of these fluctuations, each bit-string is evaluated multiple times, and the individual evaluations averaged to create the final score. This robustness is gained at the cost of a corresponding increase in training time [40]. The coefficients in the evaluation function (Equation 6.4) were determined manually. However, as shown in 7.2.1, the training algorithm is robust to changes in these coefficients: changing the evaluation function within generous limits still produces qualitatively similar driving modules.

6.3.2 Parameter Encoding

As described in Section 6.2.1, PBIL requires that candidate solutions be represented as bit-strings. Since PolySAPIENT parameters are real-valued, an appropriate quantization scheme is needed. Furthermore, the tradeoff between accuracy (which increases with bit-string length) and learning speed (which decreases with bit-string length) suggests that each parameter be encoded in as few bits as possible. As de-

Bit Value	Internal Parameter Base	External Parameter Value
000	0	1
001	1	2
010	2	4
011	3	8
100	4	16
101	5	32
110	6	64
111	7	128

Table 6.1: Three bits are used to encode each parameter. For internal parameters, a linear coding was used; for external parameters, an exponential coding was used. The internal parameter base is then scaled to cover the range appropriate for each parameter.

scribed in Chapter 5, there are two types of parameters in PolySAPIENT: those internal to reasoning objects (thresholds or gains which affect the behavior of their algorithms) and those external to reasoning objects (weights which scale the objects' action votes). In the experiments described in this chapter, each parameter was discretized into eight values (represented by a three bit-string). Internal parameters, whose values are expected to remain within a certain small range, were linearly mapped into three bits; by contrast, external parameters, whose values could range widely, were converted to a bit-string using an exponential representation. The latter provides finer resolution at the low values of the parameter in a small number of bits. The encoding scheme is summarized in Table 6.1. The three bits corresponding to each parameter are concatenated (in a predetermined order) to create the bit-string corresponding to a particular PolySAPIENT configura-

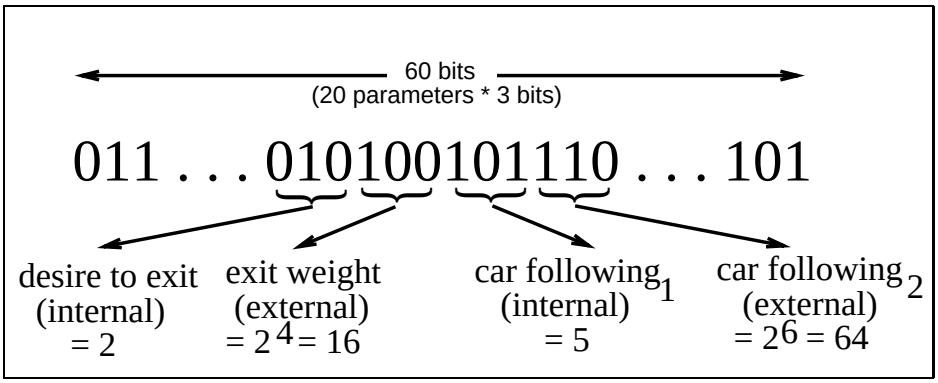

Figure 6.2: The three-bit encoding scheme used to represent parameters: internal parameters are linearly scaled while external ones are exponentially scaled.

tion. This is illustrated in Figure 6.2. For the experiments described in this chapter, a group of twenty PolySAPIENT parameters, both internal and external, were chosen (for a string length of 60 bits).

6.3.3 Training Scenarios

As mentioned in Section 6.3.1, the absence of immediate feedback due to the lack of a teacher requires that the evaluation of candidate strings be dependent on the *accumulated results* of individual tactical actions. Thus, each candidate string is evaluated over the lifetime of a simulated vehicle in a training scenario in the following manner.

The PBIL algorithm generates a candidate parameter string (by sampling the probability vector) which maps to a set of reasoning object parameters for the PolySAPIENT cognition module. A SHIVA factory

creates a *learning vehicle* with these parameters and injects it into the simulated environment at a suitable entry point. The learning vehicle has the following strategic-level goals:

- Leave the highway at a particular exit.
- Attempt to drive at a desired velocity, v_d.
- Avoid collisions (and near-collisions) with other vehicles.

The simulation ends when any of the following events is encountered: the learning vehicle takes an exit (hopefully the correct one); the learning vehicle is involved in ten or more near-collision incidents; the scenario times out; the learning vehicle is stationary for long periods of time (speeds training time by eliminating vehicles which sit still for the entire scenario). At this point, SHIVA computes an evaluation for the learning vehicle based upon its accumulated statistics. SHIVA then repeats the simulation for the same learning vehicle in other scenarios and averages the individual evaluations. The overall evaluation for the vehicle is then used by the PBIL algorithm to update its probability vector.

The procedure outlined above does not discuss a number of complications required in the implementation. The first issue relates to the behavior of *other* vehicles during the scenario — essentially a bootstrapping problem. Without a set of good parameters, it is difficult to

generate realistic ambient traffic. Without reasonable traffic providing a suitable training environment, it is impossible for PBIL to learn good parameters (especially those parameters which relate to reasoning about other vehicles). To escape this circular dilemma, the ambient traffic in each scenario was not controlled by PolySAPIENT; rather these vehicles were driven by the *rule-based* controllers described in Chapter 4. Some repercussions of this choice are discussed in Section 6.4. The second issue is that multiple scenarios were needed to expose each learning vehicle to the interesting (and difficult) aspects of driving in traffic. The scenarios explored different values for some of these factors:

- Ambient traffic density
- Number of stopped vehicles on the road (to force lane changing)
- Velocity distribution of ambient traffic

Naturally, additional training scenarios increase training time, particularly when candidate strings are evaluated multiple times on each scenario to compensate for the stochastic nature of the environment (see Section 6.3.1). In the experiments described below, each candidate vehicle was simulated four times per scenario, over four different scenarios (where each scenario involved driving the vehicle over approximately two laps of the training track described below).

The testbed for the learning vehicles was SHIVA's *cyclotron* test-track

(see Figure 7.5). The track consists of a multi-lane loop, with a single-lane on-ramp (at the bottom of the figure) and an exit (located at the top of the figure). Two factories are located at the on-ramp: one for rule-based vehicles, and the second for learning vehicles. Vehicles taking the exit, or vehicles which are involved in ten or more near-collisions were removed from the testbed.

Despite the track's odd geometry, the cyclotron has a number of advantages as a testbed for training and evaluating vehicles in simulation. First, it is topologically identical to an infinite highway with equally spaced exits and on-ramps. By equipping simulated vehicles with smart exit-finding sensors, it is possible to transform the strategic-level goal of "Take the n-th exit" to the equivalent goal of "Drive $n-1$ laps and take the next exit". Unlike the infinite highway, the cyclotron can be displayed in its entirety on the workstation screen, enabling researchers to observe the behavior of all of the vehicles at once. Most importantly, the cyclotron's topology allows one to create challenging traffic entry/exit interactions with only a small number of vehicles (since the vehicles are recycled from one situation to the next).

Each training scenario was initialized with eight rule-based vehicles, placed in the multi-lane loop, and one learning vehicle entering from the on-ramp. The rule-based vehicle factory ensured that the ambient traffic level was maintained, while the learning vehicle factory produced learning vehicles (one at a time) with the candidate strings generated by PBIL. Figure 6.3 shows a learning vehicle that has threaded

its way through an obstacle course composed of stopped and moving cars.

6.4 Results

A series of experiments using a variety of PBIL population sizes, evaluation functions and initial conditions were performed. Section 7.2.1 presents some of these in detail. The training method described here is tolerant of small changes in evaluation function and environmental conditions, and PBIL is reliably able to optimize parameter sets in this domain. Figure 6.4 shows the results of one such experiment with a population size of 100. Also shown are the results from a second experiment with a population size of 20, using the same evaluation function (see Figure 6.5). Experiments exploring this technique's sensitivity to the evaluation function are described next.

6.4.1 Evaluation Function Sensitivity

The evaluation function described in Section 6.3.1 is a linear combination of several measured aspects obtained over scenario runs. A series of experiments were performed where the coefficients associated with each aspect were varied.

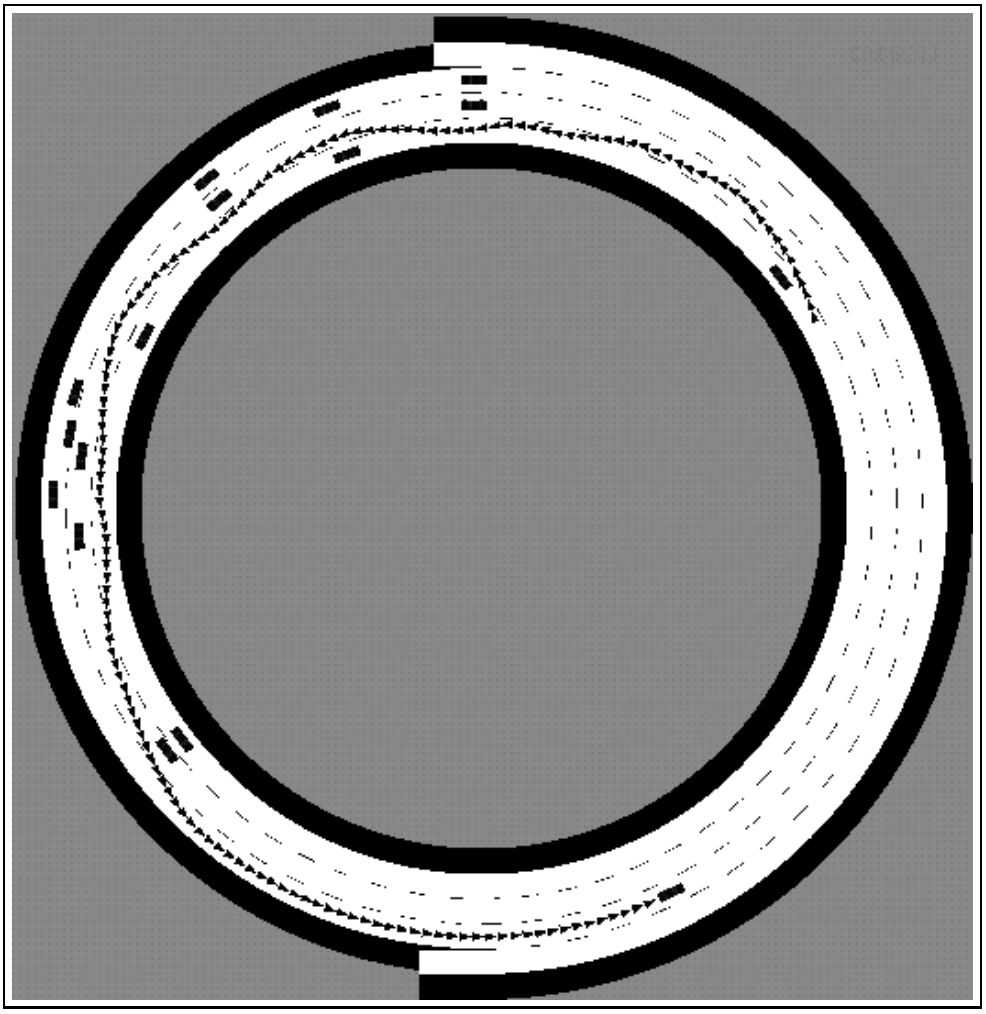

Figure 6.3: An example of a training scenario on the *cyclotron* track. The vehicle leaving a trail is the PolySAPIENT learning vehicle. The other vehicles are either stopped (stationary obstacles) or moving (ambient traffic).

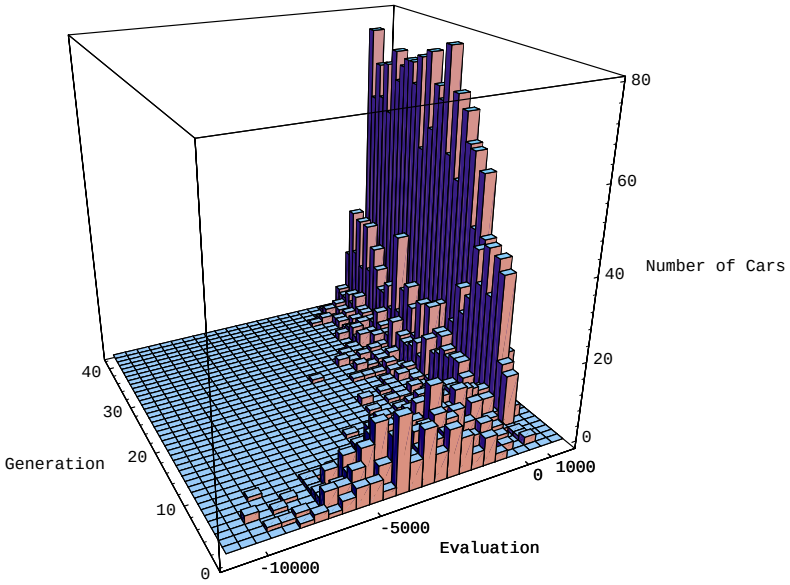

Figure 6.4: 3-D Histogram showing increase of high-scoring parameter strings over successive generations. Population size is 100 cars in each generation.

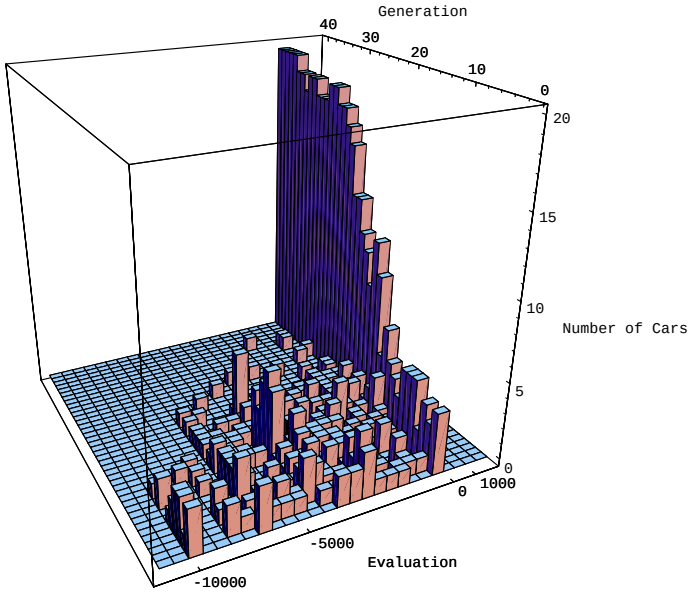

Figure 6.5: 3-D Histogram showing 20 cars in each generation.

In each experiment, a coefficient from the evaluation function was selected and changed by an order of magnitude (multiplied by 10). Then, PBIL was used to train PolySAPIENT vehicles on the scenarios described earlier. Each training run examined the exaggerated impact of one aspect of the evaluation function. The results showed that the performance of the vehicles was *not* sensitive to the evaluation function — all six tests converged to PolySAPIENT parameters that performed well on the set of test scenarios (See Chapter 7 for more details). From this, it can be hypothesized that the bulk of the "intelligence" in the PolySAPIENT vehicles comes from the structure and functions inside the reasoning objects, and that the learning is an automated way to get various reasoning objects to work well together. This is supported by PolySAPIENT's design: each reasoning object is specifically designed to tackle certain problems. For example, the vehicle reasoning objects do not need to learn how to dodge vehicles from scratch — they are already equipped with generalized potential fields. Since all vehicles possess the right *framework* for the task, the task of learning how to drive is transformed into the easier task of parameter optimization.

Chapter 7

Evaluating Tactical Driving

76.4% of all statistics are meaningless.
—Unknown

In this thesis, I have described a framework for developing tactical driving systems based on situation awareness theory and presented two implementations of such a system. The first approach, MonoSAPI-ENT, is a monolithic, rule-based system which explicitly models the domain knowledge needed for driving in traffic. The second, PolySAPI-ENT, consists of a collection of independent experts (reasoning objects), each of which focuses on a local aspect of the tactical driving task. To evaluate these systems, quantitative measures of performance are desired. This chapter discusses how tactical driving systems can be evaluated, using methods derived from situation awareness research

in other domains.

7.1 Assessing Situation Awareness

While situation awareness has long been anecdotally correlated with competence, attempts to quantitatively assess an agent's SA are relatively recent. Methods for evaluating situation awareness have been divided into three broad categories [33, 88]:

Explicit performance measures: (e.g., SAGAT: SA Global Assessment Technique [25]). Here, a simulated mission is stopped at arbitrary points during the run, and the agent is questioned about the current situation. The answers are later compared with what was actually happening, and the agreement between the two is used as a measure of the agent's situation awareness.

Implicit performance measures: This involves the design of experimental scenarios that include tasks and events that probe [111, 88] the agent's situation awareness. The method assumes a direct relationship between SA and performance.

Subjective ratings: (e.g., SART: Situation Awareness Rating Technique [112]). In this approach, agents are immersed in a challenging scenario requiring a high degree of situation awareness. At the completion of the scenario, during a debriefing process, the

> agents are expected to rate their own level of SA, based on their performance in the scenario.

Although the agent in all of these studies was a human operator or pilot, the methods can be extended to assess the performance of automated agents. For the first two methods, this is relatively straightforward. However, the third (subjective) assumes that the agent is capable of a high degree of introspection — currently beyond the scope of most AI systems. While TacAir-Soar [85] agents can be debriefed using a natural language interface, it is unclear whether they are able to accurately assess their own degree of situation awareness[1]

Nearly all methods used to evaluate SA in humans rely on the notion of a *scenario*. Pew defines a scenario as:

> . . . a set of environmental conditions and system states with which the participant is interacting that can be characterized uniquely by its priority goals and response options [72].

In the tactical driving domain, a scenario is a traffic situation involving a small number of vehicles, where the agent is given some high-level goals (such as "take the next exit"). To solve the scenario, the agent must demonstrate the ability to achieve its goals while successfully reacting to low-level concerns (e.g., other cars), as perceived by

[1] Some critics claim that no agent, human or automated, is capable of assessing its own degree of SA through introspection [88].

the agent's sensors.

I have explored both implicit and explicit performance methods in evaluating the tactical driving systems described in this thesis. The former requires the researcher to postulate suitable evaluation functions based on external observables, and examine how these functions vary over different situations. The latter involves testing driving systems on a number of *micro-scenarios* by stopping the simulations at critical points and examining the inputs, internals, and intermediate results of the algorithms. The implicit methods show how the intelligent vehicles behave under typical conditions, while the explicit methods are invaluable for debugging specific aspects of an agent's behavior. This chapter presents a selection of experiments of both types, and a discussion on the competence of the two driving systems.

7.2 Implicit Measures of SA

Chapter 6 introduced the idea of an evaluation function as a method of quantifying the competence of evolutionary PolySAPIENT vehicles. The learning technique assumed that the internal parameters of a given vehicle were correlated with the score its received (over a set of scenarios), and that this score was also measure of its tactical driving competence, or situation awareness. Thus, the evaluation function can be interpreted as an implicit measure of SA. Unfortunately, the

evaluation function, E, (See Section 6.3.1), is a *combination* of measured quantities, and does not give researchers an intuitive feel for a vehicle's actual behavior.

The experiments described in this section measure "real" quantities such as the number of near-collisions, or the percentage of missed exits, in order to gain a better understanding of the tactical reasoning modules' performance. I detail two such experiments: the first examines how these quantities vary as PolySAPIENT vehicles are trained using PBIL; the second compares the performance of MonoSAPIENT vehicles with trained PolySAPIENT cars.

7.2.1 Insights into PolySAPIENT Training

Three observable quantities play a significant role in the PolySAPIENT training evaluation function, E (See Section 6.3.1): κ, the number of near-collisions; β, a flag for missed exits; and, ζ, the distance traveled by the intelligent vehicle in the scenario. Thus, for a given population of PolySAPIENT vehicles, the quantities: $K = \sum_{\forall v} \kappa$, $B = \sum_{\forall v} \beta$, $Z = \sum_{\forall v} \zeta$ (over all vehicles, v, in the population), reflect the "goodness" of the population. The three graphs in Figure 7.1 show how K, B, and Z change over successive generations. Each population contains 40 vehicles, and each vehicle is evaluated on four different scenarios, all on the cyclotron track. The training scenarios were similar to the testing scenario showed in Figure 6.3, except that stationary obstacles and

ambient traffic factories were placed in different locations . The graphs show that:

- The number of near-collisions, K, drops steadily as PBIL tunes the PolySAPIENT parameters. In the final generation, *none* of the vehicles in the population are involved in *any* near-collisions over the set of four scenarios.
- The fraction of vehicles in the population which missed their exit also decreases steadily over time as the PolySAPIENT vehicles learn. This too is zero in the final generation.
- The third quantity, Z, reflects the incremental improvement in performance of the vehicles during training. It can be seen that the early vehicles are eliminated from the scenario (either through time-out, taking the wrong exit, or crashing) before they travel very far. Over time, the vehicles become more situationally aware and are able to travel greater distances. Note that Z has an upper bound (since ζ cannot be greater than the distance to the desired exit).

The same experiments were then repeated for six sets of training runs, each with the different evaluation functions, as described in Section 6.4.1. A subset of the results are shown in Figures 7.2 and 7.3. The first figure shows the number of near-crashes observed during training for various evaluation functions; the second shows the number of missed-exits observed during training for the same evaluation functions. Note that all

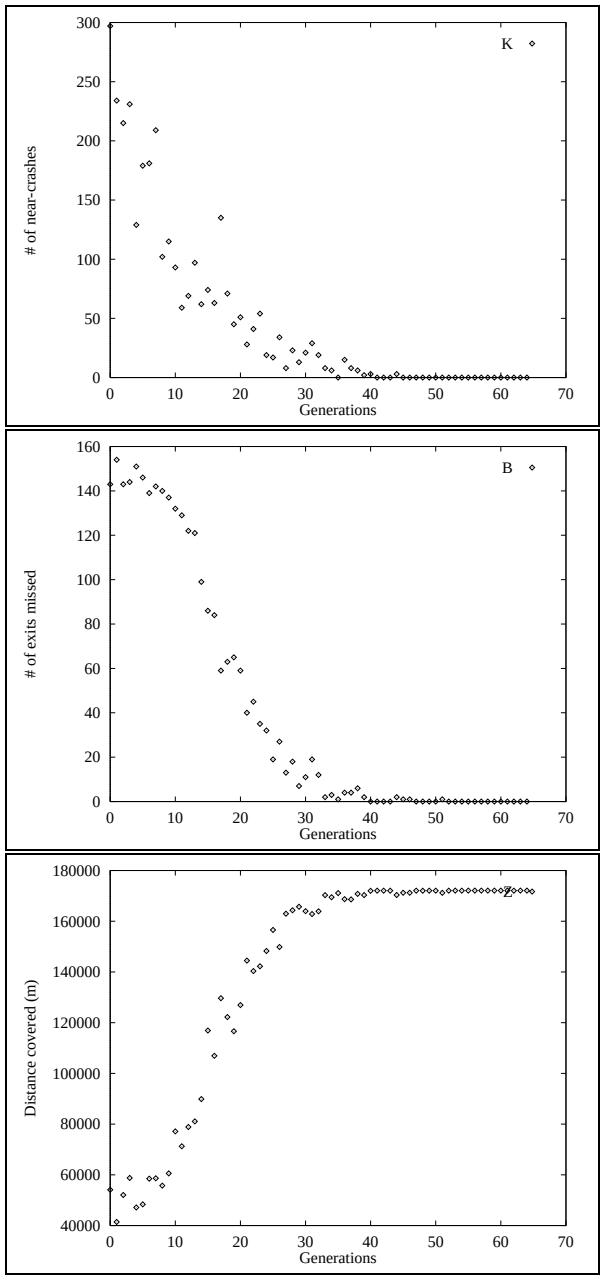

Figure 7.1: This graph shows how the number of near-collisions (K), number of missed-exits (B), and distance traveled (Z) in a population of learning PolySAPIENT vehicles varies with successive generations. In these tests, the population size was set to 40, and each vehicle was evaluated on four scenarios. Note that both K and Z decrease to zero, while Z, the incremental reward, rises.

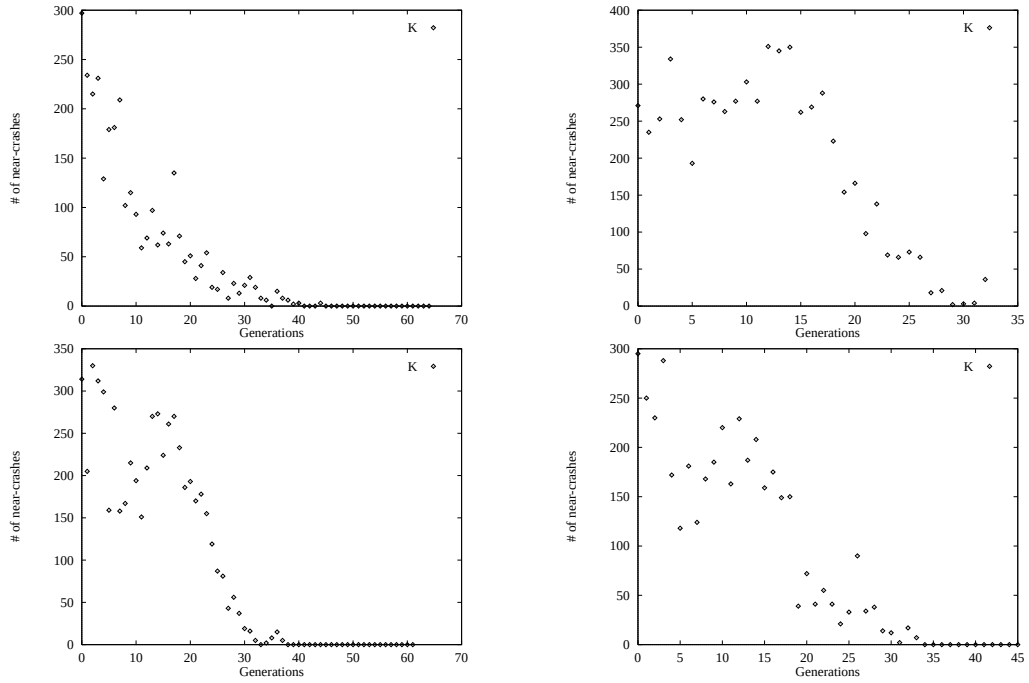

Figure 7.2: This graph shows that the PolySAPIENT parameters learned by PBIL converge to competent vehicles despite variations in the evaluation function used. Each of these graphs shows the total number of near-collisions (K), in a 40 element population of cars, evaluated on four scenarios. In each experiment, a coefficient weighting one observable was multiplied by 10. These are, (clockwise from top left): κ, β, χ, and ζ.

of the graphs follow the trends discussed above (K and B both dropping to zero over time), and support the hypothesis that the PolySAPIENT training process converges to competent vehicles despite variations in the evaluation function.

To explore the limits of the effects of the evaluation function on training, further sets of experiments was performed, where particular coefficients in the evaluation function (See Equation 6.4) were drastically

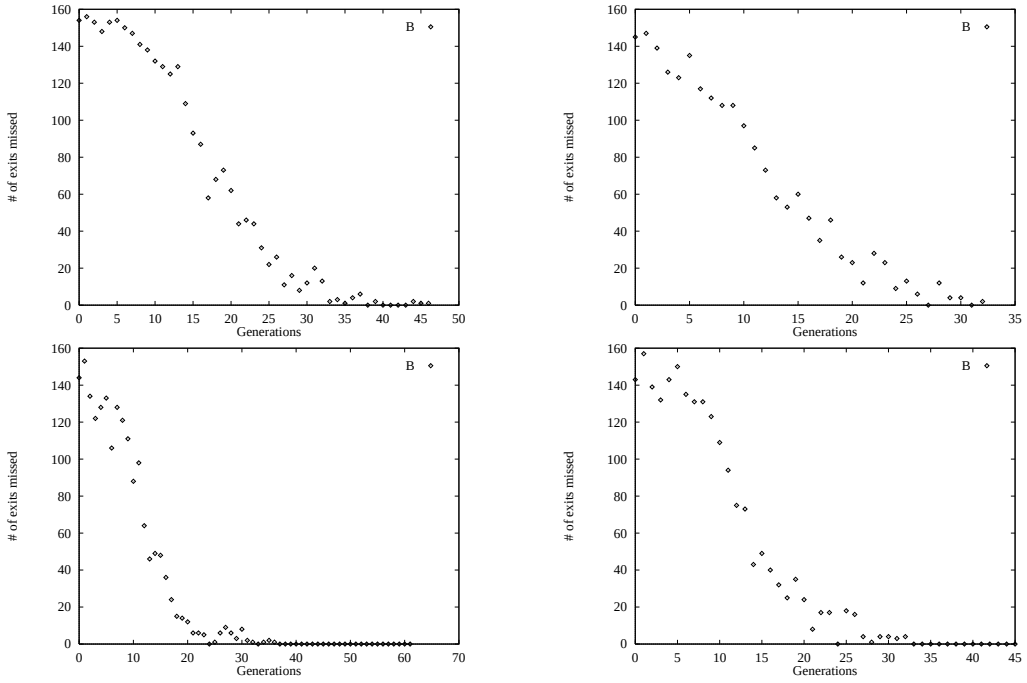

Figure 7.3: This graph shows that the PolySAPIENT parameters learned by PBIL converge to competent vehicles despite variations in the evaluation function used. Each of these graphs shows the total number of missed exits (B), in a 40 element population of cars, evaluated on four scenarios. In each experiment, a coefficient weighting one observable was multiplied by 10. These are, (clockwise from top left): κ, β, χ, and ζ.

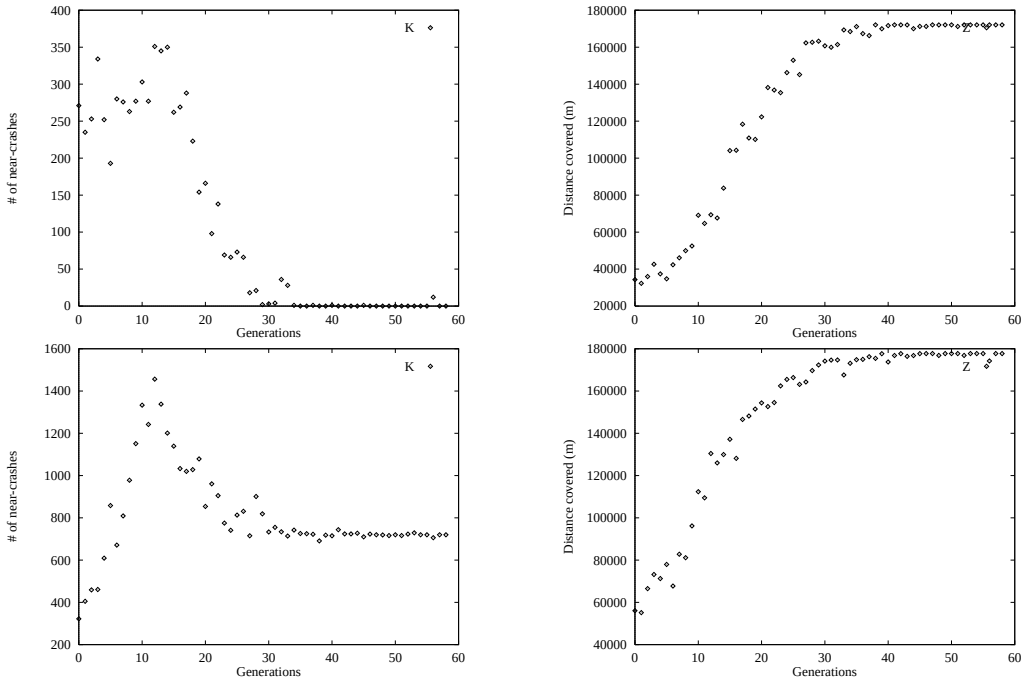

Figure 7.4: These graphs present a subset of results obtained by varying the evaluation function by a factor of 1000 (see text). The left column shows the total number of collisions (K) for evaluation functions where the coefficients weighting κ and ζ were varied. The right column shows the sum of vehicle distance traveled (Z) for the same two evaluation functions. Note that while increasing κ's coefficient by 1000 still results in competent vehicles, when ζ's coefficient is increased, K does not drop to zero. These results are discussed in the text.

perturbed. In each set, one coefficient was selected and multiplied by 1000. The observables, K, B, and Z were collected during the training period. A subset of the results is shown in Figure 7.4. A number of interesting points may be noted.

First, perturbing different coefficients has different impacts on the observables. The two graphs in the top row of Figure 7.4 show K (the

total number of collisions) and Z (the total distance traveled) in the set of runs where the coefficient associated with collisions (κ) was multiplied by 1000. By comparing these graphs with their counterparts from Figure 7.1, one can see that the behavior is remarkably similar. By contrast, perturbing the incremental reward factor (ζ's coefficient) results in strikingly different behavior: the PolySAPIENT vehicles never learn how to avoid obstacles, even at the end of training. Why does ζ's coefficient have such little impact on the qualitative behavior? One answer is that collisions are strongly penalized in training: a vehicle is eliminated after ten crashes. This means that collision avoidance is already critical, and the actual value of the coefficient is not that important. On the other hand, by increasing the incremental reward (ζ's coefficient) by a factor of 1000, vehicles that are not good at avoiding obstacles still get high scores (provided that they crash less than 10 times in one run). The final vehicles in this set crash an average of four times during the scenario: normally a significant penalty, but negligible given the warped evaluation function.

Second, the different parts of the evaluation function shown in Equation 6.4 are closely tied. Thus, a vehicle which learns to avoid obstacles is also more likely to be able to make its desired exit (since vehicles which crash more than 10 times in training are eliminated). Similarly, a vehicle which learns to take its desired exit is guaranteed to get a very high incremental reward. This explains why the vehicles still perform surprisingly well even when the evaluation function contains one factor which is off by several orders of magnitude. As long as the evalu-

ation function does not contain too many competing aspects, the PBIL algorithm learns a satisfactory solution.

Third, the learning task is considerably simplified because the natural tendency of untrained reasoning objects is still qualitatively correct. For example, a vehicle reasoning object implemented with generalized potential fields will attempt to avoid obstacles (rather than move towards them) in all cases. The main task of the evolutionary algorithm is to *glue* the various components of the system together so that they can solve the task. The fact that this can be accomplished robustly reflects positively on PolySAPIENT's architecture.

7.2.2 Heavy Traffic on the Cyclotron

In these experiments, the *cyclotron* track, shown in Figure 7.5 was used. Vehicles were injected into the scenario from the on-ramp at regular intervals, τ. Each vehicle was given two strategic-level goals: 1) make exactly one circuit of the track before exiting; 2) maintain the speed at which it was injected whenever possible. The aim of the experiment was to see how the two tactical driving systems, MonoSAPIENT, and PolySAPIENT, would behave as the roadway became more congested. Three sets of experiments with different traffic configurations were tried: all-MonoSAPIENT cars, all-PolySAPIENT cars, and a random mix of MonoSAPIENT and PolySAPIENT cars.

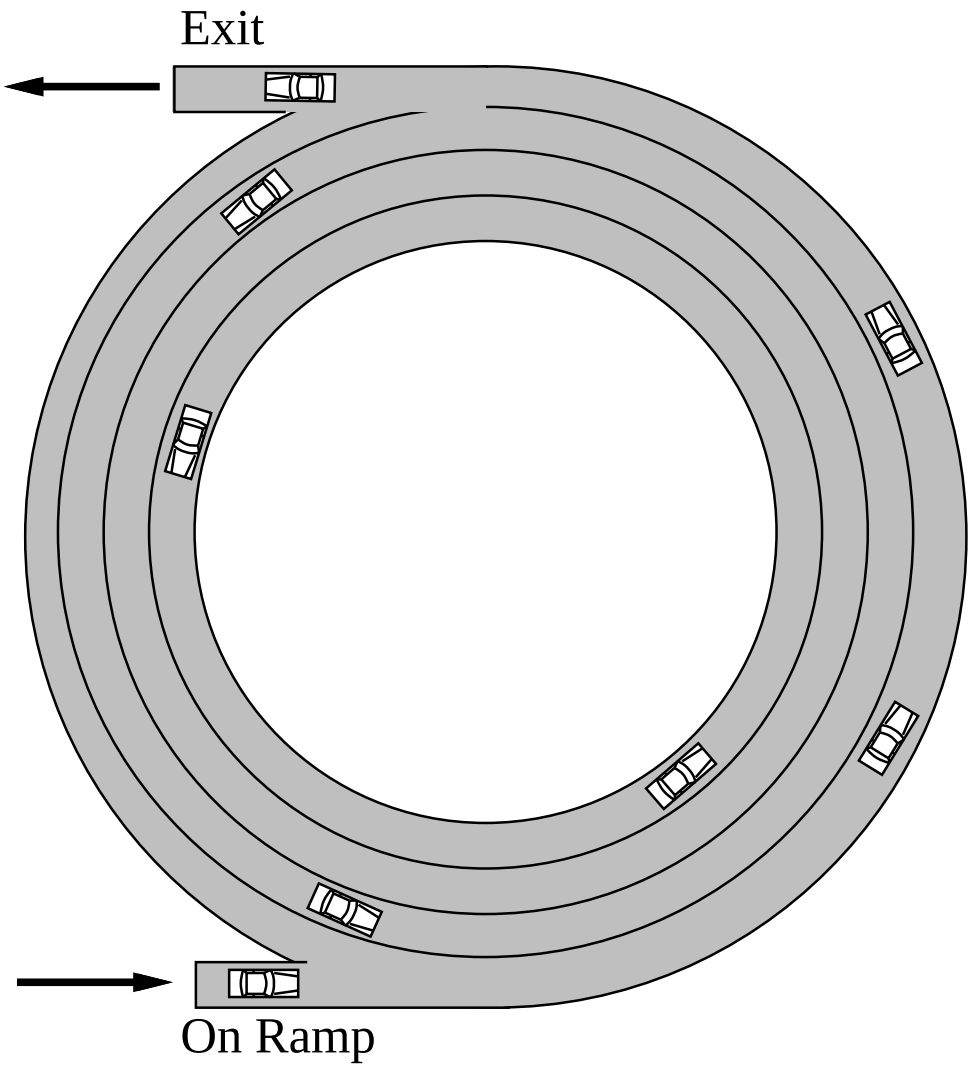

Figure 7.5: The cyclotron track: a test-bed for the Busy Highway scenario. Vehicles were injected at the on-ramp every τ secs and instructed to make a lap of the course before exiting.

As expected, the number of vehicles on the roadway increased until the rate of vehicles entering the scenario was equal to the rate of vehicles leaving (either because the vehicles had successfully completed their circuit, or because the vehicles were unable to merge into the traffic stream). At low rates of traffic flow (e.g., $\tau > 6$ seconds), all of the three traffic configurations safely negotiated the scenario (no missed exits). Once the traffic flow was increased, the behavior of the three traffic configurations diverged.

The graph in Figure 7.6 shows how the number of vehicles varies as a function of time when $\tau = 3$ seconds for the all-MonoSAPIENT and all-PolySAPIENT cases. Even in heavy traffic, neither of the pure traffic types have any collisions. Although both types of vehicles perform well initially (when the roadway is clear), once the number of vehicles on the track increases to about 6, the conservative MonoSAPIENT drivers are unable to merge into the traffic stream (since they require safe headways on *both* sides of the gap). Thus are unable to change lanes, and exit the scenario prematurely. To make matters worse, the MonoSAPIENT vehicles that were already on the roadway become trapped in the inner loops of the cyclotron (due to the high rate of traffic in the entry/exit lane).

The all-PolySAPIENT traffic, on the other hand, is able to drive successfully. This can probably be attributed to two factors: 1) the aggressive driving style, relying on time-to-impact reasoning objects, is willing to merge into smaller gaps; 2) the distributed reasoning system

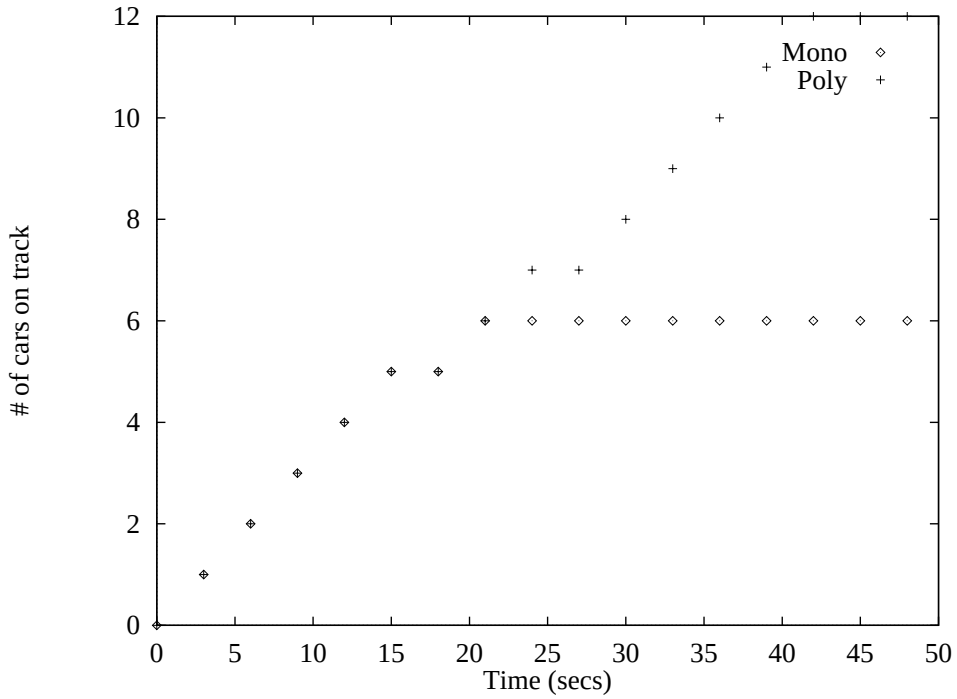

Figure 7.6: This graph shows how the number of vehicles on the cyclotron varies as a function of time in heavy traffic (traffic injection rate $\tau = 3$ secs between cars). Note that only six MonoSAPIENT vehicles were able to merge onto the cyclotron loop; by contrast, all of the PolySAPIENT vehicles were able to merge and complete the scenario.

is better at making tradeoffs — the negative votes for merging into a potentially unsafe gap are tolerated since the alternative (missing the exit) is seen to be worse. MonoSAPIENT's decision-tree is brittle, and rejects the gaps outright.

Interleaving MonoSAPIENT and PolySAPIENT cars in the heavy traffic scenario leads to a stable heterogenous behavior with no collisions While the more aggressive PolySAPIENT vehicles miss fewer exits, the MonoSAPIENT vehicles perform better than they did in the pure-MonoSAPIENT case because of the reduced congestion.

7.3 Explicit Measures of SA

Micro-scenarios are traffic situations such as the one described in Chapter 1. For the purposes of this discussion, each scenario focuses on the behavior of one vehicle (Car A in the diagrams) which attempts to achieve some clearly specified high-level goals. Furthermore, each scenario is designed to assess whether the intelligent vehicle has "understood" certain tactical driving concepts. It is important to note that the PolySAPIENT vehicles were *not* trained on these scenarios — they were only trained on obstacle courses in the cyclotron (See Figure 6.3).

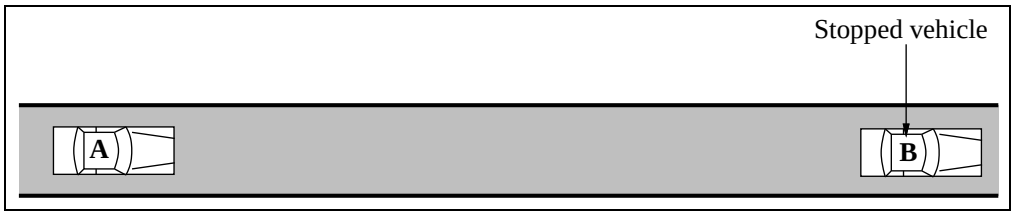

Figure 7.7: The emergency stop scenario is trivial but important. It determines whether the intelligent vehicle is overdriving its sensors and whether the vehicle realizes the inevitability of coming to a complete stop.

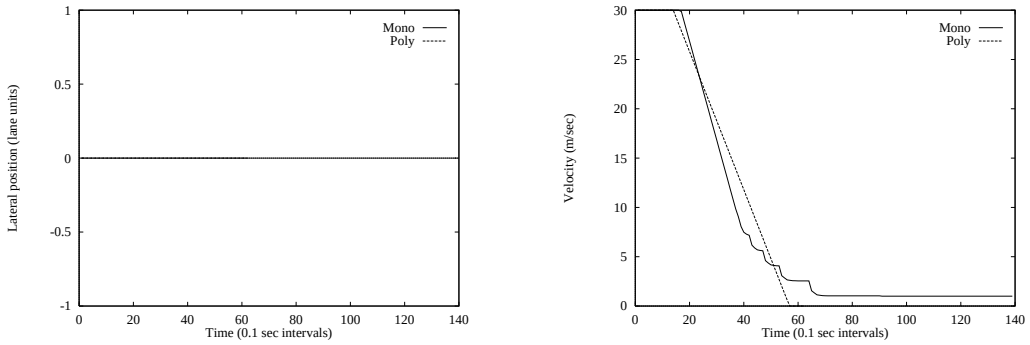

Figure 7.8: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the emergency braking scenario (See Figure 7.7). Both reasoning systems recognize that swerving is not possible, but PolySAPIENT's deceleration profile is smoother.

7.3.1 Stationary Obstacle: Emergency Braking

The first scenario (See Figure 7.7) tests whether the intelligent vehicle will brake in time to avoid colliding with a stationary obstacle in its lane when swerving is not feasible. While this scenario is trivial, it tests concepts that are critical for safety in any Automated Highway System.

Description: One lane road with no shoulder on either side, with a stopped vehicle on the roadway. The intelligent vehicle is initially travelling at highway speed (30 m/s), 200 meters behind the obstacle. See Figure 7.7).

Strategic Goal: Maintain desired velocity (30 m/s) — impossible.

Key Concepts: 1) Determine whether the intelligent vehicle is overdriving its sensor range; 2) See if the intelligent vehicle realizes the inevitability of the situation and begins decelerating in time.

Measures of Success: The primary indicator of success is avoiding the imminent collision. Secondary factors include smoothness of the deceleration profile and seeing whether the vehicle initiates any futile lane-changing maneuvers.

Results: Both MonoSAPIENT and PolySAPIENT vehicles solved this scenario successfully (See Figure 7.8). While both realize that lane-changing is impossible, PolySAPIENT's deceleration profile is smoother (potential fields vs rules). Unless PolySAPIENT vehicles encountered similar situations (stopped cars) during training, they were unable to solve the scenario, because PolySAPIENT relies on a time-to-impact potential field for obstacle avoidance. If there are no stationary objects in the training scenarios, PBIL tunes the potential field for objects that are moving at approximately the same speed as the ego-vehicle — leaving insufficient braking distance for such emergencies.

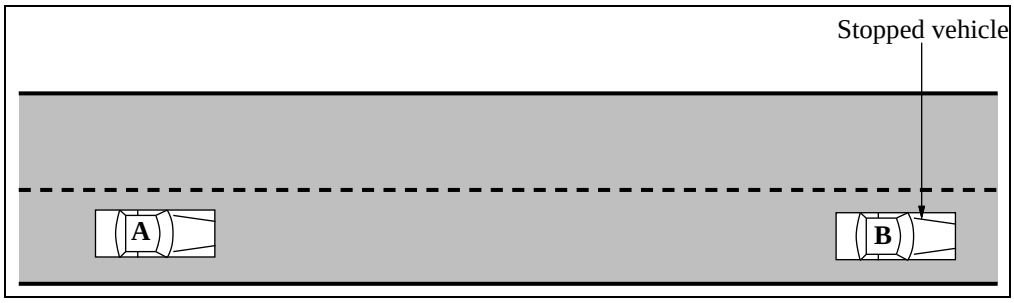

Figure 7.9: This scenario tests whether the intelligent vehicle is able to swerve to avoid the obstacle and explores the tradeoffs between strategic goals and guaranteed safety.

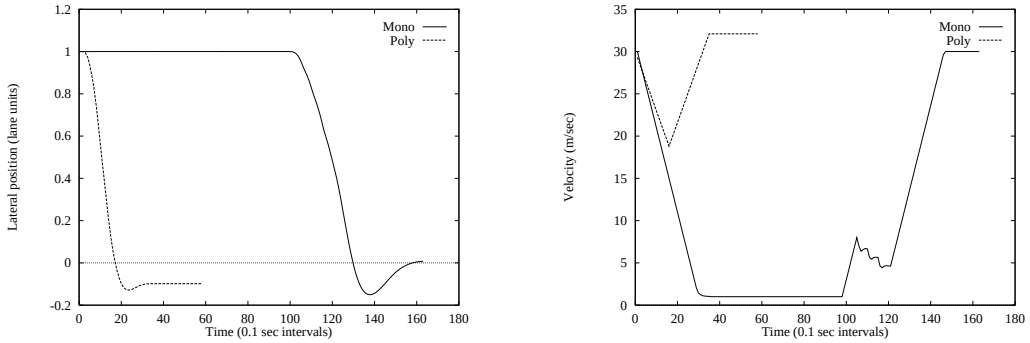

Figure 7.10: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the emergency swerving scenario (See Figure 7.9). The left lane is *lane 0*, the right lane is *lane 1*. See the text for a discussion of these graphs.

7.3.2 Swerving a Stationary Obstacle

The second scenario is also straightforward, since there are no other vehicles on the road (See Figure 7.9.

Description: Two lane road, with a stopped vehicle in the right lane. The intelligent vehicle is initially travelling at highway speed (30

m/s), also in the right lane, 200 meters behind the obstacle. See Figure 7.9.

Strategic Goal: Maintain desired velocity (30 m/s).

Key Concepts: 1) While a collision can be avoided either by braking or swerving, the higher level goals are furthered by swerving. 2) Exploring tradeoffs: safety vs strategic goals.

Measures of Success: The primary indicators of success are avoiding the collision. A point of interest is to see which tradeoff (velocity vs safety) the agent elects to make.

Results: Both MonoSAPIENT and PolySAPIENT avoid collisions, but the differences in their driving styles are clearly seen. MonoSAPIENT's rules require the vehicle to maintain a safe headway in *both* lanes during a lane change, forcing the vehicle to decelerate greatly (and the lane change happens at a low velocity). The second deceleration in the MonoSAPIENT's velocity profile (at 11 seconds) occurs because the vehicle is yawed sharply across the road and must brake to avoid running off the left side of the road. By contrast, PolySAPIENT begins changing lanes immediately (while braking slightly), and then accelerates once it has entered the clear lane. Note that PolySAPIENT does not center itself completely in the target lane. This is because the lane-tracker is allowed to deviate by a small amount (0.1 of a lane unit corresponds to 40 cm), and the chosen configuration leaves more distance between the two vehicles during the swerve.

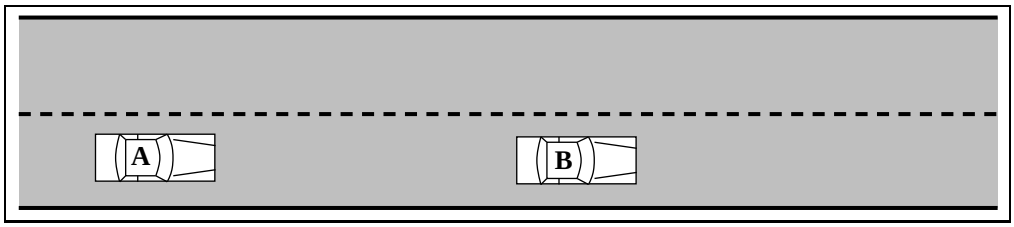

Figure 7.11: This scenario tests if the tactical reasoning system can overtake a slower-moving vehicle.

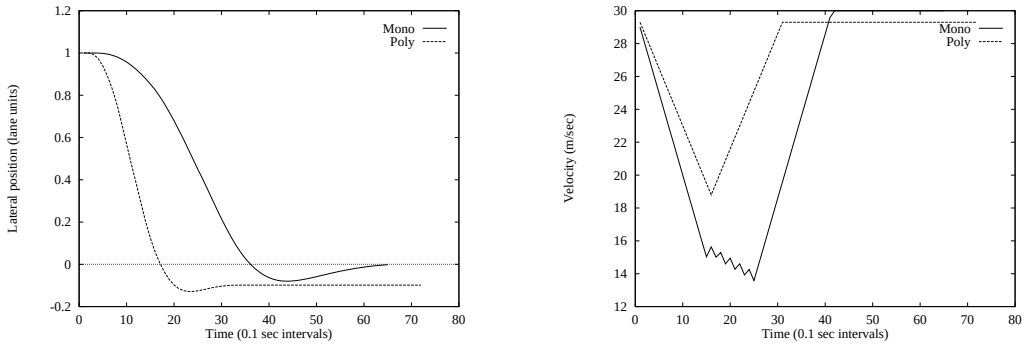

Figure 7.12: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the overtaking scenario (See Figure 7.11). The left lane is *lane 0*, the right lane is *lane 1*. See the text for a discussion of these graphs.

7.3.3 Overtaking a Slow Vehicle

The third scenario (See Figure 7.11) is a common occurrence on the roadway, and is included mainly as a sanity check. It is expected that all tactical reasoning systems should perform well in this situation.

Description: Two lane road, with a car following scenario in the right lane. Initially both vehicles are moving at 30 m/s but the lead vehicle starts braking (as it approaches an exit, for example). The

intelligent vehicle should overtake.

Strategic Goal: Maintain desired velocity.

Key Concepts: Check that the driving system is capable of overtaking.

Measures of Success: Only the ubiquitous requirement of a collision-free roadway[2]. A smooth lane-changing trajectory is preferred.

Results: As expected, both MonoSAPIENT and PolySAPIENT vehicles handle this scenario competently (See Figure 7.12). Both decelerate initially, while initiating a lane change, and then accelerate once their lane is clear. As expected, MonoSAPIENT's conservative rules force it to continue decelerating even when the intelligent vehicle is at the halfway point across lanes.

A similar scenario was then created, but this time with light traffic in the passing lane (See Figure 7.13) and a more gradual deceleration from the lead vehicle. The graphs in Figure 7.14 show the trajectories. Note that PolySAPIENT changes lanes with very little velocity fluctuation (aggressively entering the gap). MonoSAPIENT is more cautious, and misses this gap; this forces the intelligent vehicle to slow down — making the lane more difficult (because of the high velocity difference between the two lanes). However, MonoSAPIENT is able to find a later gap, pursue it, and change lanes. The velocity graph for the MonoSAPIENT vehi-

[2] In the absence of vehicle malfunction, of course...

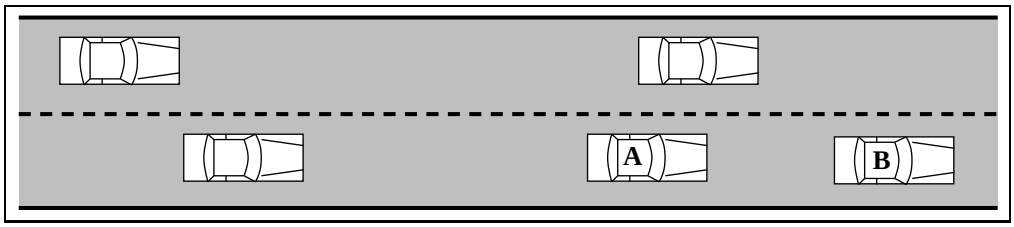

Figure 7.13: This scenario tests if the tactical reasoning system can overtake a slower-moving vehicle, in traffic.

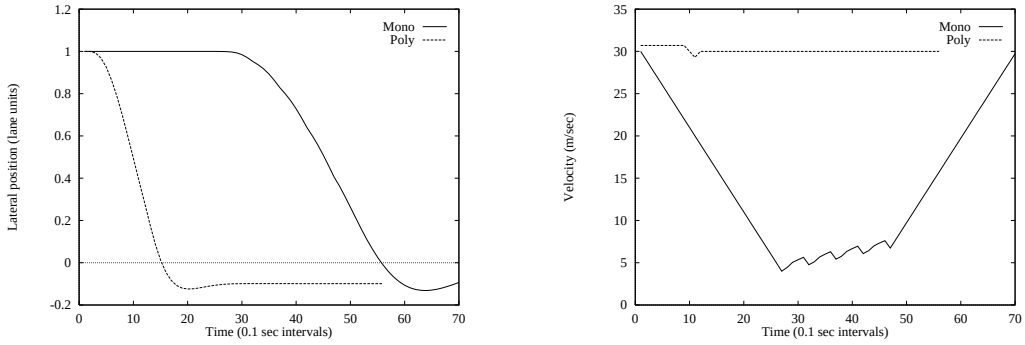

Figure 7.14: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the more difficult overtaking scenario (See Figure 7.13). The left lane is *lane 0*, the right lane is *lane 1*. See the text for a discussion of these graphs.

cle displays some undesirable tremors caused because of a brittle threshold in the algorithm: when the headway drops below a certain point, the MonoSAPIENT vehicle brakes; when the headway is clear, it attempts to accelerate to its desired speed. This problem can be addressed through a more sophisticated car following model.

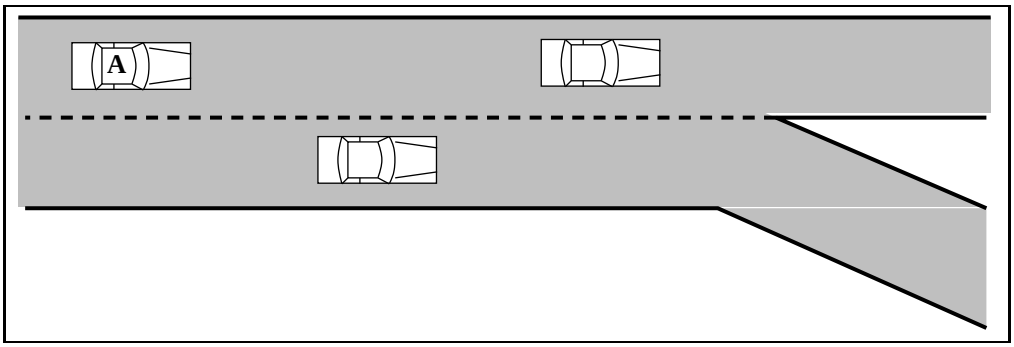

Figure 7.15: Exit scenarios add complexity to the tactical driving domain because they introduce additional strategic-level goals. The conflicts between two strategic-level goals leads to interesting tactical behavior.

7.3.4 Exit Scenarios

These scenarios introduce a second (possibly conflicting) strategic goal. The scenario of taking an exit on a clear roadway is assumed to be trivial, so only the exit scenario with traffic is shown (See Figure 7.15). Note that this is easier than the scenario shown in Figure 1.1 since the lane-change is *required* in order to satisfy strategic goals — the only challenge is to successfully merge into the target lane traffic.

Description: Two lane road, with a right-lane exit 200 meters away. The intelligent vehicle is in the left lane, moving at 30 m/s. There is light traffic in both lanes.

Strategic Goal: 1) Maintain desired velocity; 2) Take the next exit.

Key Concepts: Mandatory lane change into traffic stream.

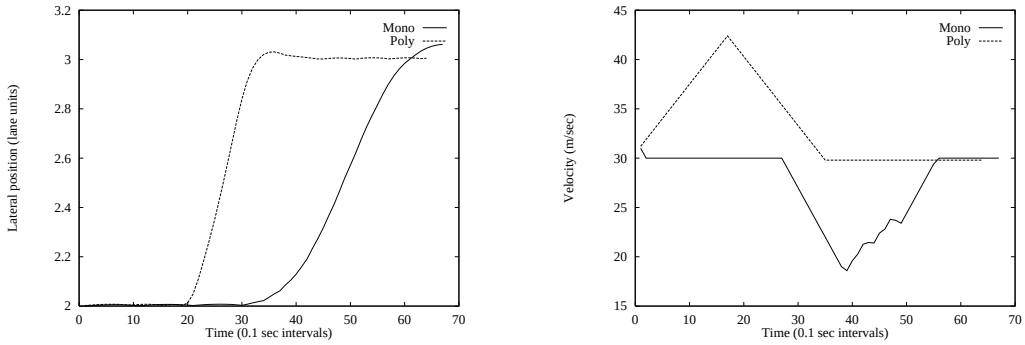

Figure 7.16: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the exit scenario (See Figure 7.15). In this scenario, the left lane is *lane 2* and the right lane is *lane 3*. See the text for a discussion of these graphs.

Measures of Success: 1) No collisions; 2) Successful exit; 3) Maintaining a reasonable velocity (i.e., not stopping in the left-lane on the highway).

Results: This scenario becomes difficult when there are few gaps in the target lane. Figure 7.16 shows PolySAPIENT accelerating to pass the vehicle in the exit lane, and changing lanes in time to make the exit. By contrast, MonoSAPIENT pursues the gap behind the vehicle in the exit lane, and makes a more cautious lane change. Neither MonoSAPIENT nor PolySAPIENT currently reason about *other* vehicles' intentions. Thus, computer-controlled cars in the target lane will not make way for a vehicle which needs to take the upcoming exit.

The final exit scenario is the one that was first introduced in Chapter 1 (and has since been used in numerous examples). It is shown, for the

last time, in Figure 7.17.

Description: Two lane road, with a right-lane exit 200 meters away. The intelligent vehicle is in the left lane, moving at 30 m/s. There is light traffic in both lanes.

Strategic Goal: 1) Maintain desired velocity; 2) Take the next exit.

Key Concepts: Deciding *whether* a lane change is feasible. A difficult test of the agent's ability to predict the future.

Measures of Success: 1) No collisions; 2) Successful exit; 3) Maintaining a reasonable velocity.

Results: The third facet of situation awareness, according to Endsley's definition [25] is the "projection of ... status in the near future". This is extremely difficult in the dynamic, incomplete, noisy domain of tactical driving. MonoSAPIENT's search tree resolves the conflict explicitly: lane-changing is prohibited when the intelligent vehicle is in the correct lane for a desired exit, once the exit gets within a certain range. PolySAPIENT, on the other hand, is less predictable. Based on the interactions between the blocker reasoning object, the exit reasoning object, the velocity reasoning object (and reasoning objects associated with the other vehicles), it will elect either to pass or to stay. Figure 7.18 shows the two reasoning systems selecting different strategies in the same scenario. MonoSAPIENT waits behind the slow moving vehicle while PolySAPIENT gambles successfully.

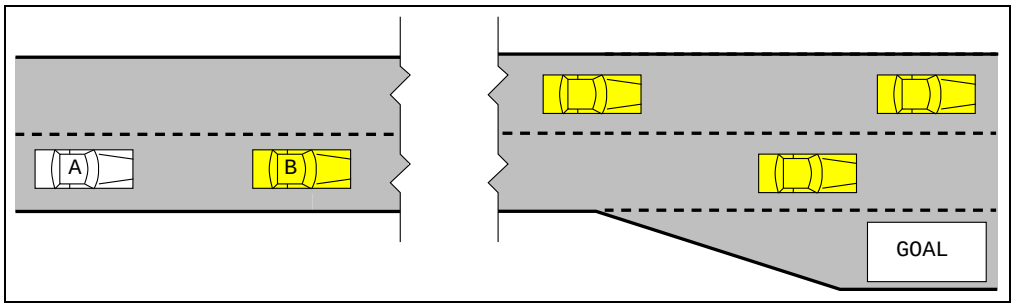

Figure 7.17: This exit scenario is difficult because the lane changes are optional. To address the strategic-level goal of maintaining speed, the intelligent vehicle must decide whether or not to gamble.

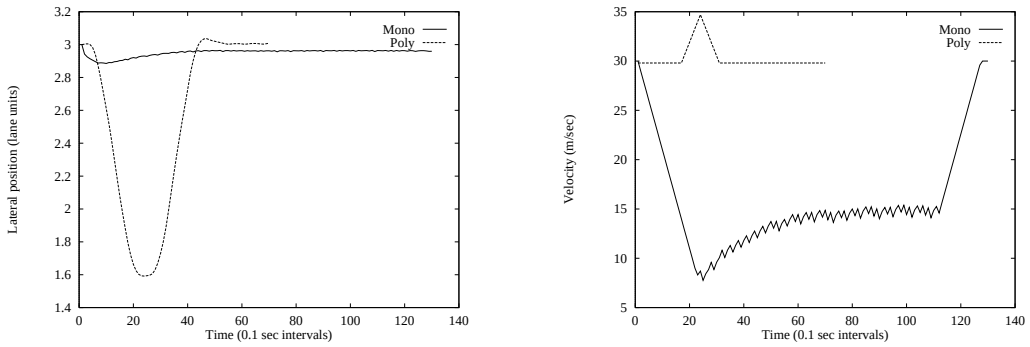

Figure 7.18: Lateral displacement (left) and velocity (right) as a function of time, for MonoSAPIENT and PolySAPIENT vehicles on the more difficult exit scenario (See Figure 7.17). In this scenario, the left lane is *lane 2* and the right lane is *lane 3*. See the text for a discussion of these graphs.

This scenario falls near the limit of both MonoSAPIENT and PolySAPIENT's reasoning capabilities. It should be noted that human drivers occasionally predict the outcome of such situations incorrectly and are forced to miss the exit. In the next section, I discuss scenarios which are currently beyond the scope of my systems.

Finally, this chapter examines some rare and challenging situations which currently fall outside SAPIENT's scope. Human drivers find these tactical scenarios difficult because the right actions cannot be determined solely on current information.

7.3.5 Discovered Check

Some scenarios are almost unsolvable by the time they are encountered, and must therefore be anticipated in advance and prevented. Consider the situation shown in Figure 7.3.5.

Description: Two lane road, with a stopped car (C) in the right lane. The intelligent vehicle is in the right lane, moving at 30 m/s, safely following another vehicle (B). There is moderate traffic in the left lane.

Strategic Goal: Maintain desired velocity

Key Concepts: Anticipating danger; defensive driving.

Measures of Success: No collisions

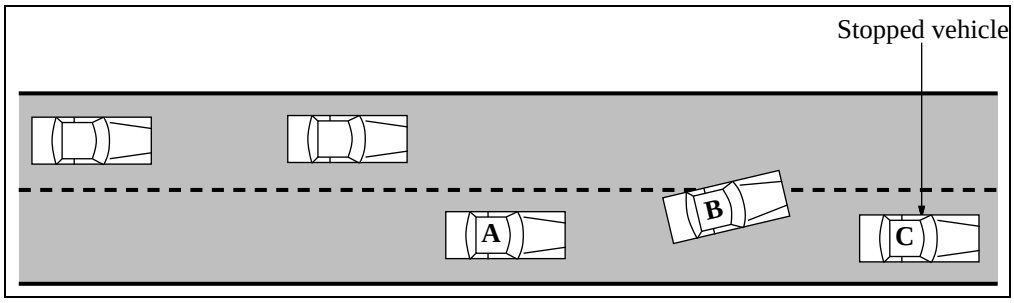

Figure 7.19: Discovered check: in this scenario, the intelligent vehicle (A) is safely following another car (B) at highway speeds. Car B, seeing the stopped car (C), blocking the lane ahead is able to change lanes. Unfortunately, traffic in the adjacent lane prevents A from changing lanes, and A has insufficient headway to stop.

Results: I call this dangerous configuration *discovered check* because the veiled threat of the stopped vehicle C is revealed only when the blocker B moves out of the line of fire. The main difficulty is that even a safe headway provides insufficient stopping distance: 2 second headway at 30 m/s is 60 meters; stopping distance assuming -7 m/s^2 deceleration is 64 meters (before any reaction delays are factored in). Thus, unless A is able to anticipate the danger and increase headway *before seeing* C, a collision is unavoidable. Humans have two defenses against such conditions: 1) they see several cars ahead, through windshields [8]; 2) they use domain knowledge to predict likely places for such incidents, such as entrances to shopping malls, or driveways.

Given these harsh conditions, I was very surprised to find that PolySAPIENT is able to successfully negotiates these scenarios reliably, even under challenging circumstances. In these experi-

ments, the headway between vehicles in the fast lane was reduced to 1 second, and all vehicles in the scenario were under PolySAPIENT control. PolySAPIENT solves the scenario by starting to brake as soon as Car C becomes visible. This has two effects: 1) it reduces the impact velocity with Car C in the event of a collision (the primary motivation for the vehicle reasoning object); 2) a clearing in the fast lane is more likely to become adjacent. As soon as the 1 second space is adjacent, the threat of a collision with the stationary vehicle overpowers votes from the reasoning objects monitoring the vehicles on either side of the gap, and the vehicle begins changing lanes into the (currently unsafe) gap. Fortunately, the vehicle on the trailing side of the gap begins braking to increase its headway, and Car A successfully escapes the trap.

MonoSAPIENT vehicles are currently unable to solve such scenarios because it never merges into an unsafe gap — because it cannot understand that the alternative is worse.

Intelligent vehicles, despite their lack of general domain knowledge, may still be able to solve these challenging scenarios. For example, a communication system, either vehicle-to-vehicle or infrastructure-to-vehicle that enabled other cars (or even Car C itself) to broadcast the potential danger would prevent Car A from following Car B too closely. Also, since the discovered check scenario tends to happen in certain locations (e.g., near shopping malls), incorporating this information through a digital map would allow an intelligent vehicle to anticipate

such situations in advance and increase its headway accordingly.

7.4 Discussion

The evaluation methods described in this chapter have been motivated by situation awareness research directed towards measuring SA in human experts. Fortunately, two of the three approaches are general, and were extended to evaluate intelligent vehicle agents. The implicit metrics were used in a series of experiments to show the large-scale competency of the two driving systems discussed in this thesis (MonoSAPIENT and PolySAPIENT). The explicit metrics, in conjunction with micro-scenarios, showed that the tactical driving systems display competence in critical driving situations. In particular, the scenarios also highlighted the tradeoffs between the two approaches, and indicated situations where one method was more successful than the other. Finally, the discovered check scenario was an example of an intelligent vehicle solving a complex scenario without explicit planning; by assiduously maintaining a space cushion, PolySAPIENT was able to merge into an unsafe gap and avoid a collision with a stationary vehicle.

It is worth noting that the comparisons presented in this chapter between MonoSAPIENT and PolySAPIENT are not intended to be interpreted as comparisons between two architectures, but rather as comparisons between two implementations; it is obvious that the behavior

of a PolySAPIENT behavior can be emulated by a rule-based system. The difference is that *creating* a PolySAPIENT vehicle to solve such scenarios is far easier than creating an equivalent MonoSAPIENT vehicle. This is best illustrated in the form of the following anecdote.

7.4.1 Escaping the Discovered Check

Until two days before my thesis defense, PolySAPIENT vehicles would consistently crash in the Discovered Check scenario (See Section 7.3.5). This did not particularly surprise me since the scenario was known to be challenging for humans. However, Dean Pomerleau [75] suggested that I try changing the behavior of the PolySAPIENT vehicles to aggressively maintain space cushions. Such a change would have been a major undertaking in a MonoSAPIENT vehicle; however, PolySAPIENT's decoupled architecture gave me the confidence that the change would be straightforward. I modified the structure of a single reasoning object type (vehicle reasoning objects) to accelerate or decelerate based on vehicles in the adjacent lane. This change required a trivially small amount of code, and was completely local to the reasoning object. As a result, I was able to generate results and produce videos showing the PolySAPIENT vehicles successfully surviving the discovered check. Thus, while MonoSAPIENT and PolySAPIENT may not differ computational expressiveness, the latter is clearly superior in practice.

Chapter 8

Conclusion

Every problem has a solution that is simple,
easily explained, and wrong.
—Unknown

8.1 Thesis Summary and Contributions

Cognitive psychologists believe that the tactical driving task, often taken for granted by humans, is a challenging information processing task:

> . . . in driving a car, you must divide attention between the gas pedal, steering wheel, speedometer, windshield, side-

> view mirror, rear-view mirror, and possibly even the radio. Also in this situation you are not in control of the rate of environmental change that must be dealt with — cars go whizzing by, traffic lights change, the road curves, and unwary pedestrians step into the street. When ... viewed from the perspective created by this type of situation, the critical question concerns how much information you can consciously become aware of in a limited amount of time [38].

My thesis presents two tactical-level reasoning systems, MonoSAPIENT and PolySAPIENT, that drive competently in traffic. These approaches are motivated by situation awareness theory, a body of research which has been applied to describe human competence in domains such as air-to-air combat. Since testing experimental tactical driving systems in real traffic is dangerous, I developed a simulation and design environment, SHIVA, specialized towards modeling tactical-level concerns. SHIVA's realistic sensor models force cognition modules to reason with incomplete, noisy world models. Furthermore, since SHIVA's perception and control modules are largely compatible with the Carnegie Mellon Navlab robot vehicles, it is expected that the algorithms developed in simulation can be easily ported to real-life. The inherent difficulty of the tactical driving problem results in a very complex decision tree. This drawback in MonoSAPIENT is addressed in PolySAPIENT by a novel decomposition of the problem: processing is distributed over a set of local experts, known as reasoning objects. Each

reasoning object is associated with physical entities in the world, and different types of reasoning objects implement their processing using a variety of algorithms. All reasoning objects speak a common language: votes and vetoes over a shared action space. A knowledge-free arbiter performs action selection by selecting the most popular (non-vetoed) action. The loosely coupled structure of the reasoning objects allows different reasoning objects to be implemented by different people, using different methods. While PolySAPIENT's reasoning objects are a powerful paradigm for designing tactical-level systems, the chore of manually tuning parameters makes integrating new reasoning objects into an existing configuration difficult. Fortunately, this parameter tuning can be automated using an evolutionary algorithm, PBIL, essentially enabling PolySAPIENT vehicles to learn how to drive — using only a user-defined fitness function, and without any external teaching signals. As an added benefit, PBIL can be used to rapidly prototype configurations of proposed reasoning objects, quickly allowing researchers to discard those configurations that are too limited for the task. Central to both the training and evaluation process is the notion of a tactical scenario. Observing the performance of a cognition module on a series of such scenarios allows one to gain an understanding about the effectiveness of the intelligent vehicle. Such explicit methods for measuring situation awareness are supplemented by testing the driving systems on larger-scale scenarios.

The primary contribution of this thesis is the PolySAPIENT framework, a method for solving tactical driving situations using a collection

of distributed modules, each tied to relevant physical entities in the driving environment. Additionally, the thesis provides a simulation and design tool for intelligent vehicle research, and applies learning to manage the unforeseen interactions between a group of independent agents.

8.2 Future Work

Research in tactical driving is still largely unexplored. My thesis work can be extended in a number of areas.

8.2.1 Tactical Driving in Real Traffic

The most exciting extension of this thesis research is porting SAPIENT to the Carnegie Mellon Navlab with the goal of driving an intelligent vehicle in real traffic situations. For this to happen, the existing, stable, operational-level systems such as lane-tracking need to be supplemented with similarly reliable obstacle detection and vehicle sensing. While the obstacle-detection systems primarily need to be forward-looking, vehicle sensing should operate with a wide (preferably all-round) field of view. The current hardware on the Navlab is inadequate for this task, but proposed upgrades such as the helical scanning laser range-finder should make this feasible. Car detection using

range sensors should also be supplemented by vision-based methods: for example, a perception module that recognized turn signals or brake lights (perhaps an extension of RACCOON [101]) could provide tactical reasoning systems with useful information about driver intentions.

In parallel with developing these perception systems on the robot testbed, functionally identical systems should be implemented in SHIVA[1]. The important requirement is that the interface to the cognition modules be identical for both the simulated and real systems. Reasoning objects should be associated with the perception system so that a stimulus in the real world (e.g., seeing a vehicle approaching from behind) will trigger the creation of the appropriate reasoning object.

The reasoning object parameters should be tuned (as described in Chapter 6) in a set of simulated runs, and tested in SHIVA on micro-level scenarios (as shown in Chapter 7) to ensure that they respond correctly to common situations such as braking, passing and swerving. Once this is completed, a simple scenario should be created in both SHIVA and on the test-track at low speeds. Alternately, a driver-warning interface (where the computer issues recommendations without controlling the vehicle) can be added to SAPIENT, allowing the vehicle to be tested in uncontrolled traffic situations. Only once the designers are confident of the system's robustness, should it be allowed to drive in real traffic.

[1] SHIVA currently provides a Navlab-compatible lane-tracker and controller.

8.2.2 Mixed Traffic Studies for the AHS

As discussed in Chapter 1, the proposals for an Automated Highway System with both human- and computer-controlled vehicles operating on an unconstrained roadway (known as mixed-traffic concepts) demand a high degree of competence from intelligent vehicles. In this thesis, I have shown that intelligent vehicles can drive safely on simulated roadways without rigid protocols enforced through vehicle-to-vehicle or vehicle-to-infrastructure communications. However, the consequences of introducing intelligent vehicles on existing highways remains to be studied.

The tactical-level consequences of mixed traffic demand a simulator capable of heterogenous vehicle simulation, such as SHIVA. Once detailed predictive human driver models are available, a number of important scenarios should be investigated in SHIVA. Three topics are worth mentioning briefly:

Impact of Aggressive Intelligent Vehicles: How will drivers react to intelligent vehicles that drive with reduced headways and superhuman reflexes? It is possible that aggressive automated vehicles will cause human drivers to become nervous (with negative effects on roadway safety). Conversely, it can be hypothesized that even a few intelligent vehicles could improve highway throughput by damping unwanted traffic oscillations.

Intelligent Vehicle Cliques: Many of the fully-automated AHS concepts employ communication to improve tactical driving performance. A weaker form of communication is also worth investigating: with the aim of creating automated vehicle *cliques*. Computer-controlled vehicles in an area could broadcast their intentions to a local group. Specialized reasoning objects would then process these intentions and use the information to influence their own tactical decisions. This approach may have several advantages: 1) incremental deployment since the automated vehicles do not *rely* on communication; 2) by cooperating, intelligent vehicles could have a relative advantage over manual traffic, (e.g., communicating vehicles could open up gaps for vehicles which intended to merge), providing an incentive for people to automate their vehicles. The benefits of forming such cliques could be quantified by measuring throughput figures for automated vehicles and manual traffic in the same macro-scenario.

Adapting to Existing Conditions: Chapter 6 discussed how SAPIENT reasoning objects can be tuned to optimize an evaluation function. It would be interesting to pursue this idea in the context of mixed-traffic applications for intelligent vehicles. Thus, SAPIENT vehicles could adapt to optimize a shared throughput as they drive in a mixed-traffic stream. It is likely that the optimal parameter settings will change as the percentage of human-controlled vehicles in the traffic mix is lowered, and the methods described in Chapter 6 will allow vehicles to evolve appropriately.

8.2.3 Reasoning Objects in Other Domains

The idea of a loosely coupled, heterogenous architecture for distributed intelligence can be applied to a broad class of problems beyond driving in traffic. As Rosenblatt [84] discussed, decentralized systems connected using voting arbiters have wide applicability in the field of mobile robots. However, SAPIENT-like architectures need not be restricted to physical domains. An interesting potential application for these systems is in rich, collaborative, virtual environments containing avatars [5, 42]. Avatars are virtual puppets representing users or programs in the system, that interact with the shared environment (and other avatars) in a quasi-realistic manner. Computer-controlled avatars need to respond to a variety of stimuli in real-time, while furthering their own internal goals. The problem of acting in such an environment is thus similar to the tactical driving task. In a SAPIENT avatar, different reasoning objects (potentially implemented by different people) would be associated with the relevant entities in the scene. The parameters of the various reasoning objects could be tuned by allowing learning-avatars to interact with trained avatars and human users of the system. By enabling users to modify an avatar's evaluation function, avatars could be personalized to reflect the users' preferences. In response to increasing complexity in the agent's action space, the PolySAPIENT extensions proposed in Section 5.9 may become more worthwhile.

Complex problems like driving in traffic rarely have easy solutions.

However, the ideas outlined in this thesis demonstrate that the right decomposition of a task may transform something that seems intractable into something that is merely difficult.

Appendix A

Reasoning Objects

This appendix details the internals of the different reasoning object types in the PolySAPIENT configuration shown in Figure 5.1. One of the architecture's main strengths is that the reasoning object configuration can very easily be changed without requiring any changes in the other objects. Thus, the objects presented below can be extended (and new reasoning objects added) to reflect the changing requirements of an intelligent vehicle designer.

A.1 Desired Velocity Reasoning Object

This stateless reasoning object is aware of the preferred velocity (as set by strategic-level goals) and seeks to bring the intelligent vehicle's cur-

rent velocity closer to the preferred velocity. Its only perceptual inputs are the vehicle's current velocity. The implementation is straightforward: votes are determined purely on the longitudinal component of an action, and actions which move towards the preferred velocity receive positive votes (while those which move away receive negative votes). The magnitude of these votes is a parameter. If the vehicle is moving at the preferred velocity, all actions which maintain the current speed receive a small positive vote. The magnitudes of all these votes are parameters which may be set by the user or learned over simulated trials using the methods described in Chapter 6.

A.2 Lane Reasoning Object

The lane reasoning object has two internal states: lane-keeping and lane-changing. It has a single perceptual input: the vehicle's lane-tracking module. When the vehicle is correctly centered in the lane, this object votes positively for all actions that maintain the current lateral position. When the vehicle moves out of its lane (during lane changing), the vehicle reasoning object monitors the motion of the lane-tracker's *pure-pursuit* point [114, 73] and votes positively for actions that continue moving the pure-pursuit point towards the new lane. Once the pure-pursuit point is close to the new lane, the lane reasoning object changes its local state to lane-keeping. This reasoning object is also responsible for vetoing any lateral motions that would cause the

vehicle to run off the road (e.g., right lane changes when the vehicle is in the right lane). The object is currently implemented as a decision tree.

A.3 Vehicle and Obstacle Reasoning Objects

These reasoning objects are currently stateless (future work involves creating more sophisticated vehicle models that can infer driver intentions). The perceptual input is a obstacle/vehicle tracker which provides the relative position and velocity of the target. Internally, these reasoning objects construct a time-to-impact potential field around the relevant obstacle (similar to the generalized potential fields used in [59]). The vote against a particular action is inversely proportional to the estimated time-to-impact. Thus, such a reasoning object will prefer braking to accelerating when it sees a stationary obstacle ahead in its lane since decelerating will increase the time-to-impact. Naturally, when an action avoids the collision entirely (time-to-impact is infinite), the reasoning object will not vote against it at all. Intuitively, this means that the intelligent vehicle will try to swerve when going at high velocities. Naturally, the constants in the potential field are all parameters which can be tuned in the usual manner.

Vehicle reasoning objects are also responsible for maintaining a *space cushion* around the vehicle (as recommended by defensive driving the-

ory). A space cushion provides several benefits: 1) it increases the number of available actions in a given situation; 2) it reduces the chance that a bad action from a neighboring vehicle will cause the intelligent vehicle to crash; 3) it partially addresses the current lack of explicit planning in PolySAPIENT by breaking vehicle configuration deadlocks. For example, the vehicle reasoning object will recommend slowing down slightly when a vehicle is passing on the left, and speeding up slightly when passing a vehicle on the right. This strategy discourages vehicles from driving side-by-side for long periods of time.

A.4 Exit Reasoning Object

The exit reasoning object is stateless, but is aware of the relevant strategic-level goal (which exit is desired). Its perceptual inputs are the exit-finder and the lane-tracker. The former provides the distance and lane of the desired exit (e.g., through a digital map and GPS) while the latter provides information about local lane markings (such as the locations of exit-only lanes). This reasoning object has two main goals: 1) to help the intelligent make the correct exit; and, 2) to prevent the vehicle from taking the wrong exit. As the vehicle approaches its desired exit, the reasoning object generates increasingly negative votes for actions whose lateral component would take the vehicle away from the exit lane, and positive votes for actions which move the vehicle towards the exit. Simultaneously, actions which reduce the vehicle's

velocity to the recommended exit speed are favored. If the vehicle is in danger of missing its exit, it will decelerate to allow gaps in the target lanes to catch up to its position.

The second goal is addressed by monitoring lane-markings and avoiding driving in exit lanes and on-ramps (until the desired exit is reached). Thus, when the vehicle is driving in an undesirable lane, the reasoning object votes against maintaining the current lateral position. The actual values of the votes are dependent on several parameters: the rate at which votes decay with distance to the desired exit, the penalty for being in an undesirable lane, etc. As described above, these parameters can either be set manually or tuned automatically.

A.5 Hysteresis Reasoning Object

This reasoning object is responsible for maintaining consistency from one time-step to the next. Since PolySAPIENT lacks a global notion of a plan, it is possible for the system to oscillate between two actions that have equally high votes. The hysteresis object takes as input, the accumulated votes from the arbiter and merely casts a positive vote for last round's most popular action. By favoring the incumbent action, the hysteresis object eliminates much of the dithering that a purely stateless system can exhibit.

Bibliography

[1] J. Aasman. Implementations of car-driver behaviour and psychological models. In J. Rothengatter and de Bruin R., editors, *Road User Behavior: Theory and Practice*. Van Gorcum, Assen, 1988.

[2] K. Ahmed, E. Moshe, H. Koutsopoulos, and R. Mishalani. Models of freeway lane changing and gap acceptance behavior. In *Proceedings of 13th International Symposium on Transportation and Traffic Theory*, 1996.

[3] R. Allen and T. Rosenthal. A computer simulation analysis of safety critical maneuvers for assessing ground vehicle dynamic stability. In *Vehicle Dynamics and Simulation, SAE SP-950*, 1993.

[4] O. Amidi. Integrated mobile robot control. Technical Report CMU-RI-TR-90-17, Carnegie Mellon University, May 1990.

[5] D. Anderson, D. Greening, M. Moses, M. Marvit, and R. Waters. Open community overview. Technical report, Mitsubishi Electric Research Laboratory, November 1996. `http://www.merl.com/opencom/opencom-overview.htm`.

[6] R. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, 1989.

[7] M. Aschenbrenner and B. Biehl. Improved safety through improved technical measures? empirical studies regarding risk compensation processes in relation to anti-lock braking systems.

In R. Trimpop and G. Wilde, editors, *Challenges to Accident Prevention: The Issue of Risk Compensation Behavior*. Styx Publications, Groningen, 1994.

[8] H. Asher and B. Galler. Collision warning using neighboring vehicle information. In *Proceedings of ITS America*, 1996.

[9] D. Ballard and L. Rippy. A knowledge-based decision aid for enhanced situational awareness. In *Proceedings of IEEE/AIAA 13th Digital Avionics Systems Conference*, 1994.

[10] S. Baluja. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man and Cybernetics*, 26(3), 1996.

[11] S. Baluja. *Expectation-Based Selective Attention*. PhD thesis, Carnegie Mellon University, 1996.

[12] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of the International Conference on Machine Learning (ML-95)*, 1995.

[13] S. Baluja and D. Simon. Evolution-based methods for selecting point data for object localization: Applications to computer assisted surgery. Technical Report CMU-CS-96-183, Carnegie Mellon University, 1996.

[14] S. Baluja, R. Sukthankar, and J. Hancock. Prototyping intelligent vehicle modules using evolutionary algorithms. In D. Dasgupta and Z. Michalewicz, editors, To appear in *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, 1997.

[15] M. Bayouth and C. Thorpe. An AHS concept based on an autonomous vehicle architecture. In *Proceedings of the Third Annual World Congress on Intelligent Transportation Systems*, 1996.

[16] J. Bender and R. Fenton. A study of automatic car following. *IEEE Transactions on Vehicular Technology*, 18:124–140, 1969.

[17] M. Booth, J. Cremer, and J. Kearney. Scenario control for real-time driving simulation. In *Proceedings of 4th Eurographics Animation and Simulation Workshop*, 1993.

[18] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

[19] K. Cardew. The automatic steering of vehicles – an experimental system fitted to a Citroen car. Technical Report RL340, Road Research Laboratory, February 1970.

[20] H. Chin. SIMRO: A model to simulate traffic at roundabouts. *Traffic Engineering and Control*, 26(3), 1985.

[21] J. Cremer, J. Kearney, Y. Papelis, and R. Romano. The software architecture for scenario control in the Iowa driving simulator. In *Proceedings of the 4th Computer Generated Forces and Behavioral Representation*, 1994.

[22] J. Cro and R. Parker. Automatic headway control — an automobile vehicle spacing system. Technical Report 700086, Society of Automotive Engineers, January 1970.

[23] E. Dickmanns and A. Zapp. A curvature-based scheme for improving road vehicle guidance by computer vision. In *Proceedings of the SPIE Conference on Mobile Robots*, 1986.

[24] R. Elliot and M. Leak. Route finding in street maps by computers and people. In *Proceedings of AAAI*, 1982.

[25] M. Endsley. Design and evaluation for situation awareness enhancement. In *Proceedings of Human Factors Society 32nd Annual Meeting*, volume 1, 1988.

[26] M. Endsley. Towards a theory of situation awareness. Technical report, Texas Technical University, Department of Industrial Engineering, 1993.

[27] F. Eskafi and D. Khorramabadi. SmartPath user's manual. Technical report, University of California, Berkeley, December 1993.

[28] F. Eskafi, D. Khorramabadi, and P. Varaiya. SmartPath: An automated highway system simulator. Technical Report UCB-ITS-94-3, University of California, Berkeley, 1994.

[29] O. Etzioni, S. Hanks, and D. Weld. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 1992.

[30] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), 1972.

[31] Linux `fortune` program, 1996. Personal communication.

[32] M. Fracker. A theory of situation awareness: Implications for measuring situation awareness. In *Proceedings of Human Factors Society 32nd Annual Meeting*, 1988.

[33] M. Fracker and S. Davis. Explicit, implicit, and subjective rating measures of situation awareness in a monitoring task. Technical report, Wright-Patterson Air Force Base, 1991.

[34] D. Gage and B. Pletta. Ground vehicle convoying. *SPIE Mobile Robots II*, 852:319–328, 1987.

[35] K. Gardels. Automatic car controls for electronic highways. Technical Report GMR-276, General Motors Research Labs, June 1960.

[36] P. Garnier, C. Novales, and C. Laugier. An hybrid motion controller for a real car-like robot evolving in a multi-vehicle environment. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[37] D. Gibson. *The Application of Traffic Simulation Models*. National Academy of Sciences, 1981.

[38] A. Glass and K. Holyoak. *Cognition*. Random House, 1986.

[39] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[40] D. Goldberg, K. Deb, and J. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6(8):333–362, 1992.

[41] A. Göllü. *Object Management Systems*. PhD thesis, University of California–Berkeley, May 1995.

[42] M. Gough et al. NTT's virtual expo project. Technical report, NTT Data Communications Systems Corporation, 1996. `http://www.construct.net/projects/ntt/about/`.

[43] J. Gowdy. *Object Oriented Architecture and Planning for Outdoor Mobile Robots*. PhD thesis, Carnegie Mellon University, 1996.

[44] J. Hancock. Dynamic vehicle simulation. Unpublished report, May 1996.

[45] K. Harwood, B. Barnett, and C. Wickens. Situational awareness: A conceptual and methodological framework. In *Proceedings of the Symposium on Psychology in the Department of Defense.*, 1988.

[46] Human Interface Technology Laboratory. Communicating situation awareness in virtual environments: Year 2 interim report. Technical report, HITL, University of Washington, Seattle, 1995. Submitted to the AFOSR.

[47] T. Jochem, D. Pomerleau, B. Kumar, and J. Armstrong. PANS: A portable navigation platform. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[48] T. Jochem, D. Pomerleau, and C. Thorpe. Vision guided lane transitions. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[49] H. Kamada, S. Naoi, and T. Goto. A compact navigation system using image processing and fuzzy control. In *Proceedings of IEEE Southeastcon*, 1990.

[50] S. Kass, D. Hershler, and M. Companion. Are they shooting at me: An approach to training situational awareness. In *Proceedings of Human Factors Society 34th Annual Meeting*, 1990.

[51] J. Kearney, 1997. Personal Communication.

[52] N. Kehternavaz, N. Griswold, and S. Lee. Visual control of an autonomous vehicle (BART) — the vehicle following problem. *IEEE Transactions on Vehicular Technology*, 40(3), 1991.

[53] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, 1990.

[54] K. Kluge and C. Thorpe. Explicit models for road following. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1989.

[55] R. Kories, N. Rehfeld, and G. Zimmermann. Towards autonomous convoy driving: Recognizing the starting vehicle in front. In *Proceedings of the 9th International Conference on Pattern Recognition*, 1988.

[56] B. Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, 1991.

[57] D. Kotchetkov. Creation of automobile automatic control system algorithms with use of computer simulation. Unpublished document, 1996.

[58] B. Krogh. A generalized potential field approach to obstacle avoidance control. In *Proceedings of Robotics Research: The Next Five Years and Beyond*, 1984.

[59] B. Krogh and C. Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proceedings of IEEE Conference on Robotics and Automation*, 1986.

[60] D. Langer, J. Rosenblatt, and M. Hebert. A reactive system for off-road navigation. In *Proceedings of IEEE Conference on Robotics and Automation*, 1994.

[61] D. Langer and C. Thorpe. Sonar based outdoor navigation and collision avoidance. In *Proceedings of IROS*, 1992.

[62] R. Larsen. AVCS: An overview of current applications and technology. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[63] I. Masaki, editor. *Vision-Based Vehicle Guidance*. Springer-Verlag, 1992.

[64] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, U. Rochester, 1995.

[65] T. McKendree. An AHS concept based on an cooperative architecture. In *Proceedings of the Third Annual World Congress on Intelligent Transportation Systems*, 1996.

[66] J. McKnight and B. Adams. Driver education and task analysis volume 1: Task descriptions. Technical report, Department of Transportation, National Highway Safety Bureau, November 1970.

[67] J. Michon. A critical view of driver behavior models: What do we know, what should we do? In L. Evans and R. Schwing, editors, *Human Behavior and Traffic Safety*. Plenum, 1985.

[68] National Safety Council. Coaching the experienced driver II, 1995. Defensive driving course training manuals.

[69] A. Niehaus and R. Stengel. Probability-based decision making for automated highway driving. *IEEE Transactions on Vehicular Technology*, 43(3), 1994.

[70] R. Oshima et al. Control system for automobile driving. In *Proceedings of the Tokyo IFAC Symposium*, 1965.

[71] S. Pentland and A. Liu. Towards augmented control systems. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[72] Richard Pew. Situational awareness: The buzzword of the 90s, 1994. Online document available on the WWW as `http://www.dtic.dla.mil/iac/cseriac/gv194sa.html`.

[73] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. PhD thesis, Carnegie Mellon University, February 1992.

[74] D. Pomerleau. RALPH: Rapidly adapting lateral position handler. In *Proceedings of IEEE Intelligent Vehicles*, 1995.

[75] D. Pomerleau, 1997. Personal Communication.

[76] D. Pomerleau, C. Thorpe, D. Langer, J. Rosenblatt, and R. Sukthankar. AVCS research at Carnegie Mellon University. In *Proceedings of Intelligent Vehicle Highway Systems*, 1994.

[77] B. Posch and G. Schmidt. A comprehensive control concept for merging of automated vehicles under a broad class of traffic conditions, 1983.

[78] D. Prerau. *Developing and Managing Expert Systems*. Addison-Wesley, Reading, MA, 1990.

[79] R. Reddy. The challenge of artificial intelligence. *IEEE Computer*, 1996.

[80] D. Reece. *Selective Perception for Robot Driving*. PhD thesis, Carnegie Mellon University, May 1992.

[81] D. Reece and S. Shafer. An overview of the Pharos traffic simulator. In J. Rothengatter and de Bruin R., editors, *Road User Behavior: Theory and Practice*. Van Gorcum, Assen, 1988.

[82] L. Reid, E. Solowka, and A. Billing. A systematic study of driver steering behavior. *Ergonomics*, 24:447–462, 1981.

[83] J. Rillings and R. Betsold. Advanced driver information systems. *IEEE Transactions on Vehicular Technology*, 40(1), 1991.

[84] J. Rosenblatt. *DAMN: A Distributed Architecture for Mobile Navigation*. PhD thesis, Carnegie Mellon University, 1996.

[85] P. Rosenbloom, W. Johnson, R. Jones, F. Koss, J. Laird, J. Lehman, R. Rubinoff, K. Schwamb, and M. Tambe. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 1994.

[86] E. Salas. Situation awareness in team performance: Implications for measurement and training. *Human Factors*, 37(1), 1995.

[87] N. Sarter and D. Woods. Situation awareness — a critical but ill-defined phenomenon. *International Journal of Aviation Psychology*, 1(1), 1991.

[88] N. Sarter and D. Woods. How in the world did we ever get into that mode? mode error and awareness in supervisory control. *Human Factors*, 37(1), 1995.

[89] S. Schuster. An AHS concept based on an maximum adaptability. In *Proceedings of the Third Annual World Congress on Intelligent Transportation Systems*, 1996.

[90] S. Selcon and R. Taylor. Evaluation of the situational awareness rating technique (SART) as a tool for aircrew systems design. In *AGARD Conference Proceedings 478 – Situation Awareness in Aerospace Operations*, 1989.

[91] R. Sengupta. An infrastructure assisted concept for AHS. In *Proceedings of the Third Annual World Congress on Intelligent Transportation Systems*, 1996.

[92] Motor Vehicle Services. *New Jersey Driver Manual*. New Jersey Department of Law and Public Safety, 1988.

[93] J. Shepanski and S. Macy. Manual training techniques of autonomous systems based on artificial neural networks. In *Proceedings of IEEE IJCNN*, 1987.

[94] L. Shrestha, C. Prince, and E. Salas. Understanding situation awareness: Concepts, methods and training. In W. Rouse, editor, *Human/Technology Interaction in Complex Systems*, volume 7. JA I Press, San Francisco, 1994.

[95] R. Small and C. Howard. A real-time approach to information management in a pilot's associate. In *Proceedings of IEEE/AIAA 10th Digital Avionics Systems Conference*, 1991.

[96] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4), 1986.

[97] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical Report CMU-RI-TR-93-20, Robotics Institute, Carnegie Mellon University, 1993.

[98] M. Stoneridge, editor. *Practical Horseman's Book of Riding, Training, and Showing Hunters and Jumpers*. Doubleday, 1989.

[99] M. Stytz. Providing situation awareness assistance to users of large-scale, dynamic, complex environments. *Presence*, 2(4), 1993.

[100] M. Sugie, O. Menzilcioglu, and H. Kung. CARGuide — on-board computer for automobile route guidance. Technical Report CMU-CS-84-144, Carnegie Mellon University, 1984.

[101] R. Sukthankar. RACCOON: A Real-time Autonomous Car Chaser Operating Optimally at Night. In *Proceedings of IEEE Intelligent Vehicles*, 1993.

[102] R. Sukthankar, J. Hancock, D. Pomerleau, and C. Thorpe. A simulation and design system for tactical driving algorithms. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*, 1996.

[103] R. Sukthankar, J. Hancock, and C. Thorpe. Tactical-level simulation for intelligent transportation systems. To appear in *Journal on Mathematical and Computer Modeling*, 1997. Special Issue on ITS.

[104] R. Sukthankar, D. Pomerleau, and C. Thorpe. Panacea: An active sensor controller for the ALVINN autonomous driving system. In *Proceedings of International Symposium on Robotics Research*, 1993.

[105] C. Thorpe, O. Amidi, J. Gowdy, M. Hebert, and D. Pomerleau. Integrating position measurement and image understanding for

autonomous vehicle navigation. In *Proceedings of the Second International Workshop on High Precision Navigation*, 1991.

[106] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Vision and navigation for the Carnegie Mellon Navlab. *IEEE Transactions on PAMI*, 10(3), 1988.

[107] J. Treat et al. Tri-level study of the causes of traffic accidents: Final report volume 1. Technical report, Federal Highway Administration, U.S. DOT, May 1979.

[108] H. van der Molen and A. Botticher. Risk models for traffic participants: A concerted effort for theoretical operationalizations. In J. Rothengatter and R. de Bruin, editors, *Road Users and Traffic Safety*. Van Gorcum, Assen, 1987.

[109] R. Vanstrum and G. Caples. Perception model for describing and dealing with driver involvement in highway accidents. *Highway Research Record*, 365:17–24, 1972.

[110] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2), 1993.

[111] M. Venturino, W. Hamilton, and S. Dvorchak. Performance-based measures of merit for tactical situation awareness. In *AGARD Conference Proceedings 478 — Situation Awareness in Aerospace Operations*, 1989.

[112] M. Vidulich and E. Hughes. Testing a subjective metric of situation awareness. In *Proceedings of the Human Factors Society 35th Annual Meeting*, 1991.

[113] R. von Tomkewitsch. Dynamic route guidance and interactive transport management with ALI-Scout. *IEEE Transactions on Vehicular Technology*, 40(1), 1991.

[114] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, and T. Kanade. First results in robot road-following. In *Proceedings of the IJCAI*, 1985.

[115] J. Want and R. Knipling. Single-vehicle roadway departure crashes: Problem size assessment and statistical description. Technical Report DTNH-22-91-C-03121, National Highway Traffic Safety Administration, 1993.

[116] E. Weiss. *Untersuchung und Rekonstruktion von Ausweich und Fahrspurwechselvorgangen*. VDI-Verlag, 1987.

[117] D. Wharton. How to train your eyes for better driving. *Readers' Digest*, November 1958.

[118] P. Wherrett. *Motoring Skills and Tactics*. Ure Smith, Sydney, 1977.

[119] B. Widrow, D. Rumelhart, and M. Lehr. Neural networks: Applications in industry, business and science. *Communications of the ACM*, 37(3), 1994.

[120] G. Wilde. *Target Risk*. PDE Publications, Toronto, 1994. Also on WWW as `http://pavlov.psyc.queensu.ca/target/`.

[121] D. Willner, C. Chang, and K. Dunn. Kalman filter algorithms for a multi-sensor system. In *Proceedings of IEEE International Conference on Decision and Control*, 1976.

[122] S. Wong. TRAF-NETSIM: How it works, what it does. *ITE Journal*, 60(4), 1990.

[123] C. Yang, M. Milacic, and K. Kurami. A longitudinal control concept for merging of automated vehicles. In *Proceedings of IEEE Intelligent Vehicles*, 1993.

[124] Q. Yang and H. Koutsopoulos. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research*, 1995.

[125] G. Zacharias, A. Miao, C. Illgen, J. Yara, and G. Siouris. SAMPLE: Situation awareness model for pilot in-the-loop evaluation. In *Proceedings of Situation Awareness in Tactical Aircraft Conference*, 1996.

[126] L. Zadeh. Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence, 1991. Introduction in Bart Kosko's book.