A Computational Model of Driving for Autonomous Vehicles

Douglas A. Reece and Steven Shafer

April 1991
CMU-CS-91-122

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Driving models are needed by many researchers to improve traffic safety and to advance autonomous vehicle design. To be most useful, a driving model must state specifically what information is needed and how it is processed. Such models are called computational because they tell exactly what computations the driving system must carry out. To date, detailed computational models have primarily been developed for research in robot vehicles. Other driving models are generally too vague or abstract to show the driving process in full detail. However, the existing computational models do not address the problem of selecting maneuvers in a dynamic traffic environment.

In this paper we present a dynamic task analysis and use it to develop a computational model of driving in traffic. This model has been implemented in a driving program called Ulysses as part of our research program in robot vehicle development. Ulysses shows how traffic and safety rules constrain the vehicle's acceleration and lane use, and shows exactly where the driver needs to look at each moment as driving decisions are being made. Ulysses works in a simulated environment provided by our new traffic simulator called PHAROS, which is similar in spirit to previous simulators (such as NETSIM) but far more detailed. Our new driving model is a key component for developing autonomous vehicles and intelligent driver aids that operate in traffic, and provides a new tool for traffic research in general.

This research was sponsored by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597 and Contract DACA76-85-C-0003.

Keywords: Robotics, simulation and modeling, vision and scene understanding, computer applications—psychology, traffic engineering

1. Introduction

Driving models are needed by many researchers. Intelligent vehicle designers need them to make driver aids that work in dynamic traffic situations. Robot vehicle builders need them to drive vehicles autonomously in traffic. Traffic engineers need them to improve the safety of highways. To be most useful, a driving model must be *detailed* and *complete*. A *detailed* model must state specifically what decisions must be made, what information is needed, and how it will be used. Such models are called computational because they tell exactly what computations the driving system must carry out. A *complete* driving model must address all aspects of driving. In the course of our research in robotics, we have developed a computational model that addresses a level of driving which has not been previously addressed. We have implemented this model in a system called Ulysses.

The driving task has been characterized as having three levels: strategic, tactical and operational [27]. These levels are illustrated in Table 1-1. The highest level is the *strategic* level, which develops

Level	Characteristic	Example	Existing model
Strategic	Static; abstract	Planning a route; Estimating time for trip	Planning programs (Artificial Intelligence)
Tactical	Dynamic; physical	Determining Right of Way; Passing another car	Human driving models
Operational	Feedback control	Tracking a lane; Following a car	Robot control systems

Table 1-1: Characteristics of three levels of driving.

behavioral goals for the vehicle. These goals may be based on route selection and driving time calculations, for example. The strategic goals are achieved by activities at the middle, *tactical*, level, which involves choosing vehicle maneuvers within the dynamic world of traffic and traffic control devices (TCD's). The maneuvers selected at the tactical level are carried out by the *operational* level of speed and steering control.

Substantial progress has been made in automating various parts of the driving task, particularly at the strategic and operational levels, but no system has yet implemented the tactical level. At the strategic level, planning programs in the Artificial Intelligence community are quite good at planning errands, finding routes, and performing other abstract tasks. However, they do not solve the problems of vehicle maneuver selection because they are not designed to work in a dynamic domain and without complete knowledge of the problem. At the operational level, several robot projects have demonstrated the ability to drive a vehicle on roads that are essentially devoid of traffic and TCD's. These systems can track lanes, but have no real knowledge of driving laws and general vehicle behavior. Thus, they too avoid the problems of maneuver selection that form the tactical level of driving.

Traffic engineers and psychologists have long studied the driving task, including tactical driving, and have developed many good theories and insights about how people drive. Unfortunately, these models of human driving do not explain exactly what and how information is processed—what features of the world must be observed, what driving knowledge is needed and how it should be encoded, and how knowledge is applied to produce actions. A model must answer these questions before it can be used to compute what actions a robot should take.

Our driving program, Ulysses, is a computational model of tactical driving. The program encodes knowledge of speed limits, headways, turn restrictions, and TCD's as constraints on acceleration and lane choice. The constraints are derived from a general desire to avoid collisions and a strategic driving goal of obeying traffic laws. Ulysses evaluates the current traffic situation by deliberately looking for important traffic objects. The observations, combined with the driving knowledge and a strategic route plan, determine what accelerations and lane changes are permitted in this situation. The program then selects the one action that allows the robot to go as fast as possible. While this model has limitations, it drives competently in many situations. Since it is a computational model, Ulysses shows exactly what information a driver needs at each moment as driving decisions are made. Furthermore, the program can be tested objectively on a real vehicle.

Our eventual goal is to use Ulysses to drive a real vehicle. At this time, however, the model has only been implemented in computer simulation. We have constructed a microscopic traffic simulator called PHAROS (for Public Highway and ROad Simulator) [30] for our driving research. PHAROS represents the street environment in detail, including the shape and location of roads, intersections, lines, markings, signs, and signals. The cars in PHAROS are driven using a simplified version of the driving model. The simulator generates an animated display of traffic that can be used to observe driving behavior. PHAROS performs the perception and control functions for Ulysses so that Ulysses can drive a robot through a road network in simulation.

This paper describes our driving model for robots in detail. The next section reviews existing driving models, including both robot driving systems and human driving models. We then present Ulysses, and describe the important features of PHAROS. The paper concludes with a discussion of possible extensions to the model and other future research.

2. Related Work

2.1. Robots and Planners

Computers have been used to automate several aspects of the driving task. Various research groups have been investigating vehicle steering and speed control for at least 30 years [5, 13, 15, 24, 28, 34]. This work generally assumes that special guides are placed in the roads, so the vehicles are not completely autonomous. In the 1980's, robots began driving on roads autonomously [9, 20, 22, 29, 36, 37, 38, 39, 42]. The different robots have strengths in different areas; some are fast (60 mph), while others are very reliable in difficult lighting or terrain conditions. Autonomous vehicles can also follow other vehicles, with or without a special guide device on the lead vehicle [4, 7, 14, 19, 21, 36].

These existing robot systems address only the operational level. Consider the driver approaching the intersection in Figure 2-1 from the bottom. A robot with current operational capabilities could track the lane to the intersection, find a path across the intersection, and then start following a new lane. The robot could also perhaps detect the car on the right and stop or swerve if a collision were imminent. However,

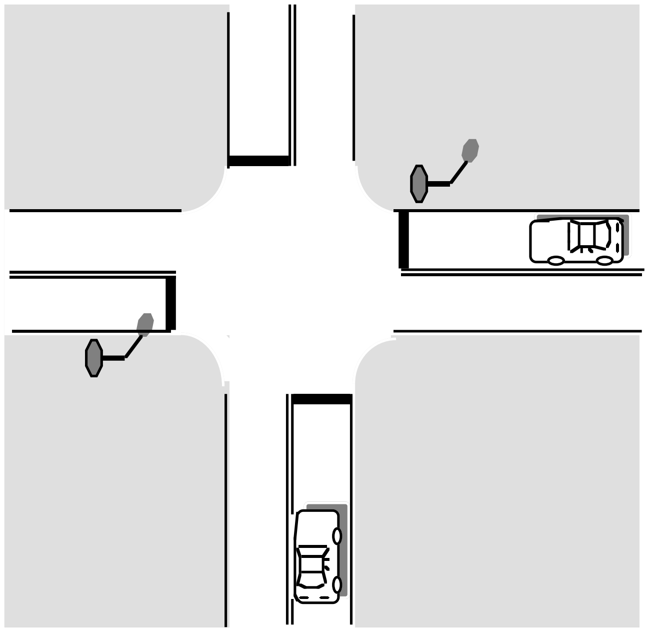


Figure 2-1: Example of tactical driving task: driver approaching crossroad.

in this situation a driver is required to interpret the intersection configuration and signs and decide whether it should pass in front of the other car. Current robot driving programs cannot perform this tactical task. Nevertheless, it is the success of automonous vehicles at the operational level that motivates us to study the tactical level.

Computers have also been used to automate strategic driving functions. Map-based navigation systems have advanced from research projects to commercial products in the last decade [3, 11, 18, 32, 35, 41]. Artificial Intelligence programs have been applied to many abstract problems, including the planning of driving errands [17]. These planners work on static problems using information conveniently encoded in a database ahead of time. Let us consider again the driver of Figure 2-1. A strategic planner may have already determined that this driver is going to a pizza store, and that he must turn left at this intersection, and that he should drive quickly because he is hungry. However, the planner (probably) does not know ahead of time what TCD's are present at this intersection, and certainly cannot predict the arrival of cross traffic. Since the situation is unknown ahead of time, the driver must get data by *looking* for signs and cars when he gets to the intersection. This data must be interpreted to form abstract concepts such as "the other car is stopping." In general, the world is uncertain and dynamic at this level, and the driver must continually use perception to assess the situation. Strategic planners are not designed to deal with these tactical problems.

2.2. Human Driving Models

Psychologists, traffic engineers and automotive engineers have studied human driving a great deal. Their goal is to make cars and roads safer and more efficient for people. The result of this work is a multitude of driving models spanning all levels of driving. Michon identified seven types of models [27]: task analysis, information flow control, motivational, cognitive process, control, trait, and mechanistic.

Each has different characteristics and addressed different aspects of driving.

Task analysis models. A task analysis model lists all of the tasks and subtasks involved in driving. The paragon of these models is McKnight and Adam's analysis [26]. Their work provides an exhaustive breakdown of all activities on the tactical and operational levels. We have found this listing very useful for determining whether our model performs all of the necessary subtasks in various situations. However, a list of tasks alone is not sufficient for describing a driving model. This is because a task list does not address the dynamic relations between tasks. The list does not specify how the driver chooses a task in a given situation, or whether one task can interrupt another, or how two tasks might be performed simultaneously (especially if they require conflicting actions). The McKnight work also leaves situation interpretation vague. For example, two tasks require a driver to "observe pedestrians and playing children" and "ignore activity on the sidewalk that has no impact on driving." However, there are no computational details about how to discriminate between these situations.

Information flow control models. Information flow control models are computer simulations of driving behavior. Early computer models of drivers were essentially implementations of task analysis models [27]. As such, they have the same weaknesses as the task analysis models. More recent microscopic traffic simulators such as SIMRO [6], TEXAS [16], and NETSIM [12, 43] do not have these weaknesses because they perform multiple tasks simultaneously. They also have the ability to start and stop tasks at any time in response to traffic. For example, NETSIM evaluates the traffic situation frequently (every second) and makes a fresh selection of appropriate subtasks. NETSIM is not intended to be an accurate description of human cognitive processing, but it produces reasonable driver behavior in many situations. NETSIM is a direct ancestor of our system, PHAROS, which we describe later in this paper.

NETSIM cannot be used as a complete driver model because it lacks several necessary components. There are gaps in its behavior; for example, cars are discharged from intersection queues into downstream streets without having to accelerate and drive normally to and through the intersection. Neither does NETSIM contain knowledge of how to interpret traffic conditions from observable objects. We also found unmodified NETSIM inadequate as a simulator testbed for robotics research because it cannot represent physical information such as road geometry, accurate vehicle location, or the location and appearance of TCD's.

Motivational models. Motivational models are theories of human cognitive activity during driving. Van der Molen and Botticher recently reviewed several of these models [40]. The models generally describe mental states such as "intentions," "expectancy," "perceived risk," "target level of risk," "need to hurry" or "distractions." These states are combined with perceptions in various ways to produce actions. Since motivational models attempt to describe the general thought processes required for driving, one would hope that they would form a basis for a vehicle driving program. However, the models do not concretely show how to represent driving knowledge, how to perceive traffic situations, or how to process information to obtain actions. Van der Molen and Botticher attempted to compare the operations of various models objectively on the same task [33, 40], but the models could be implemented only in the minds of the model designers. Some researchers are addressing this problem by describing *cognitive process models* of driving (for example, Aasman [1]). These models are specified in an appropriate symbolic programming language such as Soar [23].

The remaining types of models have limited relevance to tactical driving. *Control models* attempt to describe the driver and vehicle as a feedback control system (e.g., [31]). These models are mainly useful for lane keeping and other operational tasks. *Trait models* show correlations between driver characteristics and driving actions. For example, drivers with faster reaction times may have a lower accident rate. This correlation does not describe the mechanism by which the two factors are related. Finally, *mechanistic models* describe the behavior of traffic as an aggregate whole. These models express the movement of traffic on a road mathematically as a flow of fluid [10]. This type of model cannot be used to specify the actions of individual drivers.

2.3. Ulysses

In this paper we present a new computational model of driving called Ulysses. It is computational because it explains how to compute driving actions from sensed data. Ulysses encodes knowledge for performing a variety of tactical driving tasks, including maneuvering in traffic on multilane highways and negotiating intersections. Like the information flow control models, Ulysses performs all appropriate tasks simultaneously. Ulysses goes beyond models such as NETSIM because it describes how to drive in the transitions between states (e.g. between free flow and enqueued). Unlike existing models, Ulysses makes decisions based on observable phenomena. The perceptual part of driving is therefore described in terms of looking in specific areas for specific objects. This explicit situation assessment is much more concrete than the "risk perception" in the motivational models. Ulysses operationalizes concepts such as "risk avoidance" in these models with specific constraints on lane selection and speed. Since Ulysses is implemented as a computer program (in the language Allegro CommonLisp), it can be analyzed and tested objectively. It can be used in various driving situations to study the perceptual and information processing requirements of the task. And ultimately, Ulysses can be used to compute actions for an actual vehicle driving in traffic.

3. The Ulysses Driving Model

We have implemented a computational driving model in a program called Ulysses. Ulysses defines a complete system for driving — what a driver must look for, how situations are evaluated, what practical knowledge is needed, and how this knowledge is encoded and applied. Figure 3-1 is a schematic of the entire driver model. Only the tactical level is implemented in detail in Ulysses. The model treats the operational and strategic levels abstractly since the details have already been addressed by other work. These levels are discussed at the end of the section.

Figure 3-1 shows that at the tactical level, Ulysses performs several perception and decision functions. Ulysses uses the route plan created at the strategic level as a guide in scanning the road ahead along the intended route. The portion of the strategic route plan that is actually visible to the robot is called the *corridor* in our model. Ulysses identifies the corridor at the tactical level so it knows which signs, signals and other cars are important. The corridor is built up incrementally as the Ulysses finds intersections and chooses which direction to look further. As the corridor is found, implicit goals to drive lawfully and safely cause constraints to be generated. The constraints are triggered by the presence of various traffic objects. Ulysses must look for all objects that could trigger a constraint. After all constraints have been applied, Ulysses chooses an acceleration and lane-changing action from the available choices. This selection is based on the implicit goals of driving quickly and courteously. State variables are used to record decisions such as accepting gaps in traffic that affect actions for a period of time. Table 3-1

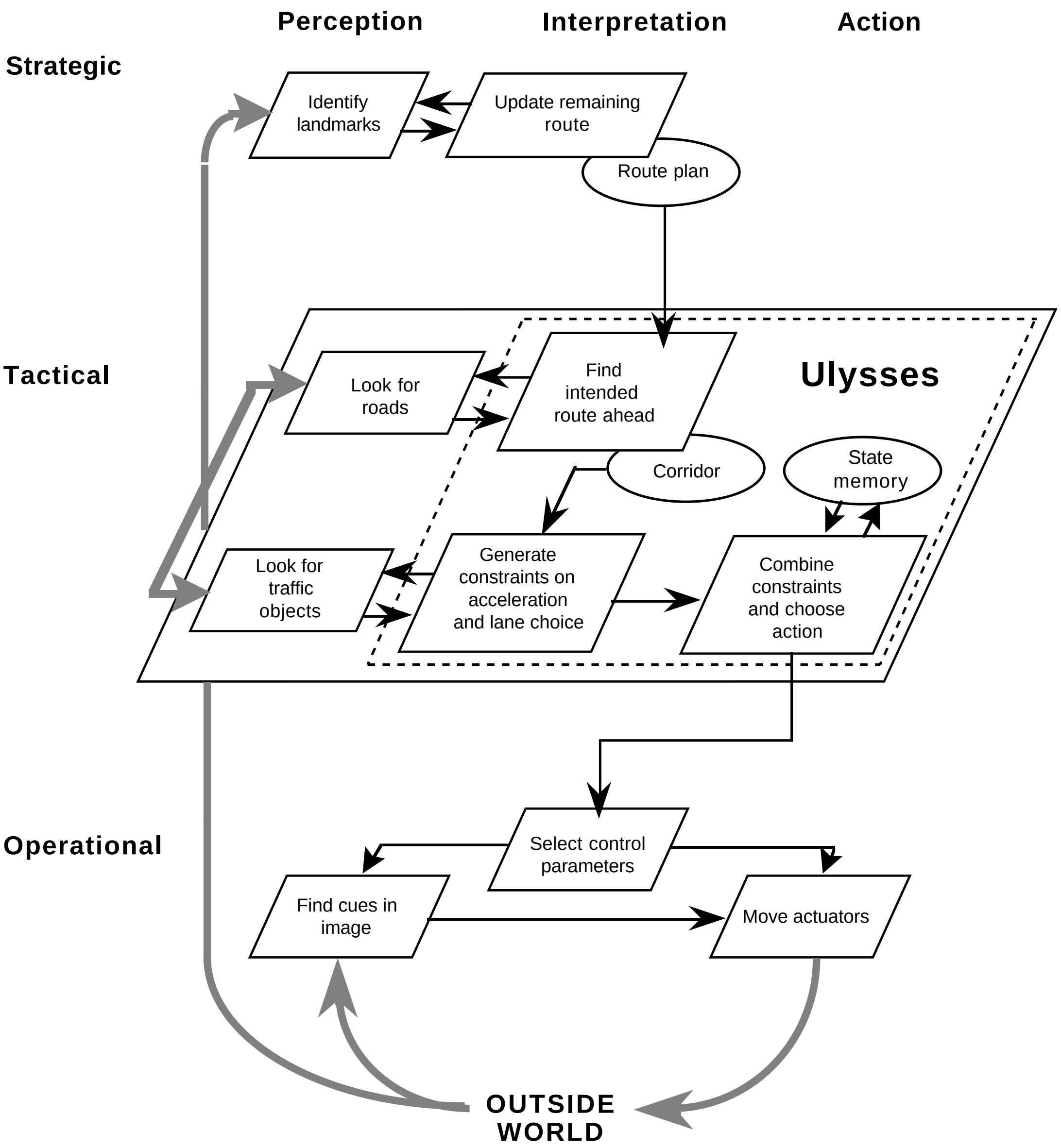


Figure 3-1: Schematic of the Ulysses driver model

---

summarizes the state variables used in Ulysses; these will be explained later in this section.

In order to be able to respond quickly to unpredictable events, Ulysses frequently repeats its visual search and evaluation of the current situation. The decision cycles have a period of 100ms. This rapid polling simulates the responsiveness of perceptual interrupts. However, the polling scheme allows constraint evaluations to activate perception rather than the other way around. Ulysses uses this demand-driven perception at the tactical level to provide a mechanism for focusing its perceptual attention. Focusing perception is important for robots because machine perception is very difficult; it would be impossible to perceive and analyze everything in the field of view in 100 ms.

State Variable	Values
Speed Limit	<current speed limit>
In intersection	Street, Intersection
Wait	Normal, Wait for gap, Wait for merge gap, Accept
Lane Position	Follow lane, Init-left (-right), Changing-left (-right)

Table 3-1: Ulysses state variables.

In the remainder of this section we explain the specific driving knowledge in Ulysses by describing a series of scenarios of increasing complexity. The scenarios include a simple highway with only one lane, intersections with various traffic control devices, and finally multi-lane roads. The section concludes with a discussion of perception, strategic plans and operational control.

3.1. Tactical Driving Knowledge

3.1.1. A Two-Lane Highway

Figure 3-2 depicts a simple highway driving situation. In this scenario no lane-changing actions are necessary, but there are several constraints on speed.

The first two constraints are derived from general safety goals—i.e., self-preservation. The speed of the vehicle must be low enough to allow it to come to a stop before the end of the road is reached; the speed on a curve must keep the vehicle's lateral acceleration below the limit imposed by friction between the tires and road surface. Although it is possible that these constraints can be met by the operational level systems of the robot, the prediction of future speed constraints from observations of distant conditions is in general a tactical activity. This is especially true if future conditions are detected not by tracking road features but by reading warning signs. Ulysses generates the road end and road curvature constraints by examining the corridor. The robot must be stopped at the end of the road if the road ends; furthermore, at each point of curvature change, the robot's speed must be less than that allowed by the vehicle's lateral acceleration limit. Ulysses also scans along the right side of the corridor for signs warning of road changes, and creates a speed constraint at the relevant signs.

These constraints—a maximum speed at a point somewhere ahead in the corridor—are typical of the motion constraints generated by various driving rules. Given the robot's current speed, we could compute a constant acceleration value that would yield the desired speed at the desired point. However, it is also possible to satisfy the constraint by driving faster for a while and then braking hard. Figure 3-3 shows these two possibilities as curves X and Y on a graph of speed versus distance. The figure shows that any speed profile is possible as long as it stays below the maximum braking curve. The profile that maximized vehicle speed would rise instantly to the maximum braking curve and then follow the curve to

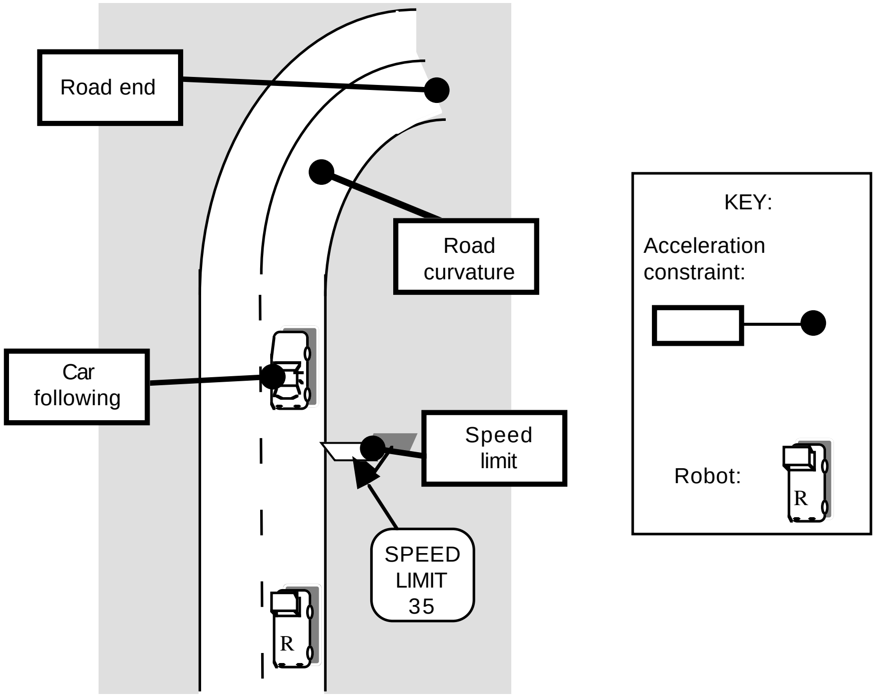


Figure 3-2: The two-lane highway scenario.

---

the constraint point. However, the program cannot change the acceleration command at arbitrary times; it looks at situations and changes acceleration only at discrete intervals. Therefore Ulysses computes an acceleration that will cause the robot's speed to meet the maximum deceleration curve exactly at the next decision time. This is curve Z in the figure.

While the high-acceleration, high-deceleration policy shown by curve Z maximizes vehicle speed, it seems to have the potential to cause jerky motion. In fact, the motion is usually smooth because the constraints relax as the robot moves forward. For example, suppose that the robot's sensors could only detect the road 150 feet ahead. Ulysses would constrain the robot to a speed of 0 at a distance of 150 feet. However, when the robot moved forward, more road could be detected, so the constraint point would move ahead. Figure 3-4 shows how this "rolling horizon" affects speed. The robot starts at distance 0 with a speed of 45 fps. A decision interval of 1.0 seconds is assumed in order to show speed changes more clearly. The dashed lines show deceleration constraint curves (-15 fps^2) at the first four decision times. The solid line shows the robot's speed. After a small overshoot, the speed reaches an equilibrium and remains constant.

The next constraint illustrated by Figure 3-2 is that generated by legal speed limits. Ulysses maintains a state variable which records the current speed limit. The vehicle is allowed to accelerate so that it reaches the speed limit in one decision cycle period. Furthermore, Ulysses scans the right side of the corridor for "Speed Limit" signs. Whenever the system detects a speed limit change, it creates an acceleration constraint to bring the robot's speed to the limit speed at the sign (as described above). As the robot nears the Speed Limit sign, Ulysses also updates the speed limit state variable.

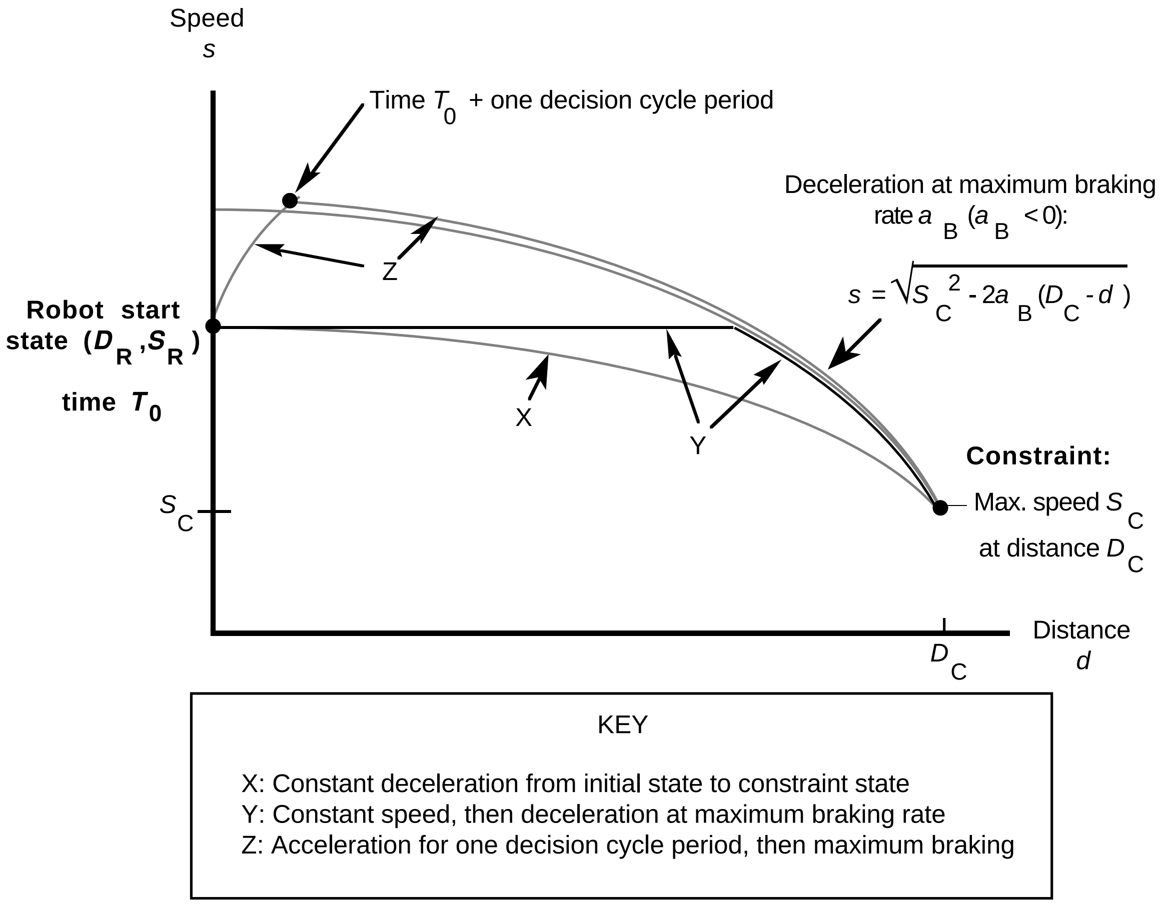


Figure 3-3: Deceleration options plotted on speed-distance graph

---

The final acceleration constraint is generated by traffic in front of the robot. The safety goal implicit to Ulysses requires that an adequate headway be maintained to the car in front of the robot in the same lane. Ulysses therefore scans the corridor to monitor the next car ahead. If the lead car were to suddenly brake, it would come to a stop some distance ahead. This is the constraint point for computing an acceleration limit. The program uses the lead car's speed, its distance, and its assumed maximum deceleration rate to determine where it would stop. This activity is commonly called "car following."

After all acceleration constraints have been generated, they are combined by taking their logical intersection. The intersection operation guarantees that all constraints are met simultaneously. Figure 3-5 illustrates this process. Since the constraints in effect allow a range of accelerations between the negative limit (the braking capability of the robot) and a computed positive value, the intersection results in a range from the negative limit to the smallest computed positive value. Ulysses chooses the largest allowed acceleration in order to further the implicit goal of getting to the destination as quickly as possible.

The perception system may not be able to detect objects beyond some range. Ulysses deals with the resulting uncertainty by making two assumptions: first, that there is a known range within which perception is certain; and second, that objects and conditions that trigger constraints are always present

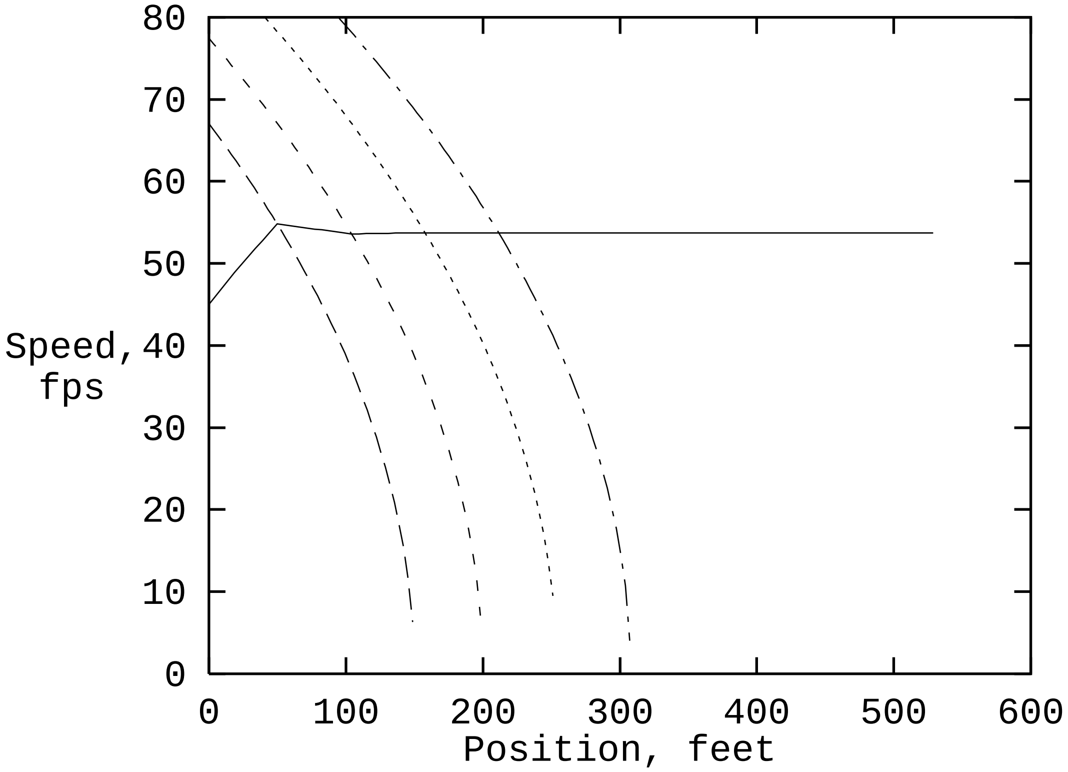


Figure 3-4: Robot speed profile (solid line) as it adjusts to a rolling road horizon. Maximum deceleration is -15.0 fps^2, decision time interval 1.0 sec, horizon at 150 feet. The four dashed lines show, for the first four decision times, the maximum-deceleration profile to the current horizon.

---

just beyond this range. This latter assumption is the worst case. For example, Ulysses always assumes that the road ends just beyond road-detection range, that the speed limit drops to zero just outside of sign-detection range, and that there is a stopped vehicle just beyond car-detection range. In this way Ulysses prevents the robot from "over-driving" its sensors.

3.1.2. An Intersection Without Traffic

The next traffic scenario we consider is a simple intersection with only one lane on the robot's approach, no other traffic, and no pedestrians. Figure 3-6 illustrates this scenario. Ulysses detects intersections when it visually tracks a lane ahead and finds that the markings end or the road branches. Other clues, such as traffic signals or signs (or cars, in a different scenario), may be detectable at longer ranges; however, since the robot is constrained to stop anyway at the end of the detected road, Ulysses does not consider these clues. Future versions of Ulysses may model the uncertainties of machine perception in more detail and use several clues to confirm observations.

When the robot is actually in an intersection, Ulysses can no longer detect a lane directly in front of the robot. When this first happens, Ulysses changes the In Intersection state variable from Street to Intersection. Later, when there is a lane in front of the robot, the state is changed back again. These state changes mark the robot's progress in its strategic route plan.

Finding the corridor is more difficult at an intersection than it is on a simple highway because the

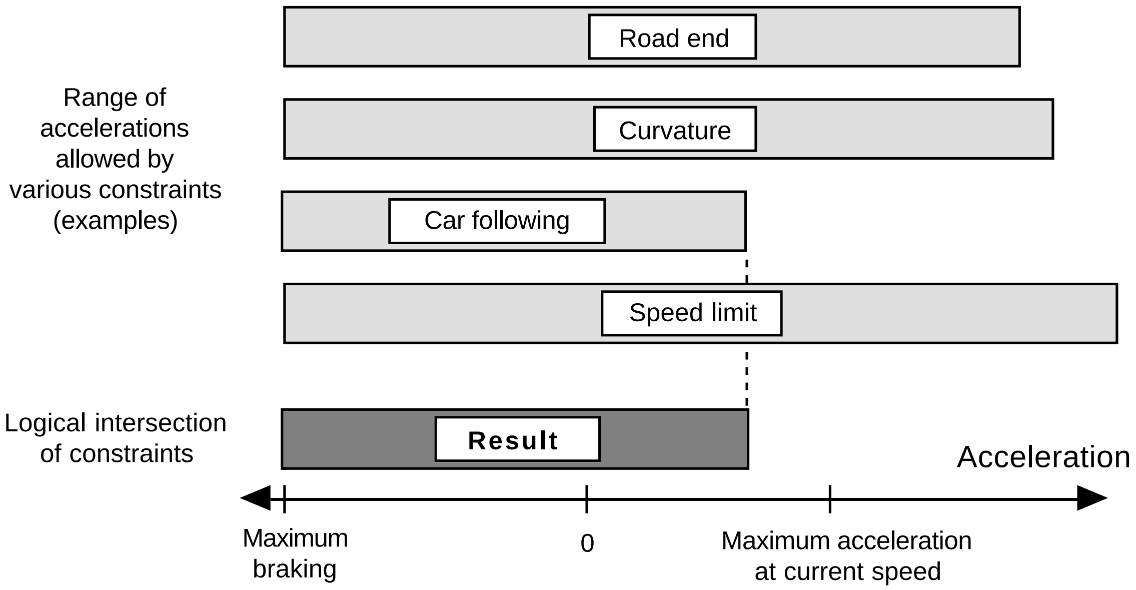


Figure 3-5: Combining acceleration constraints.

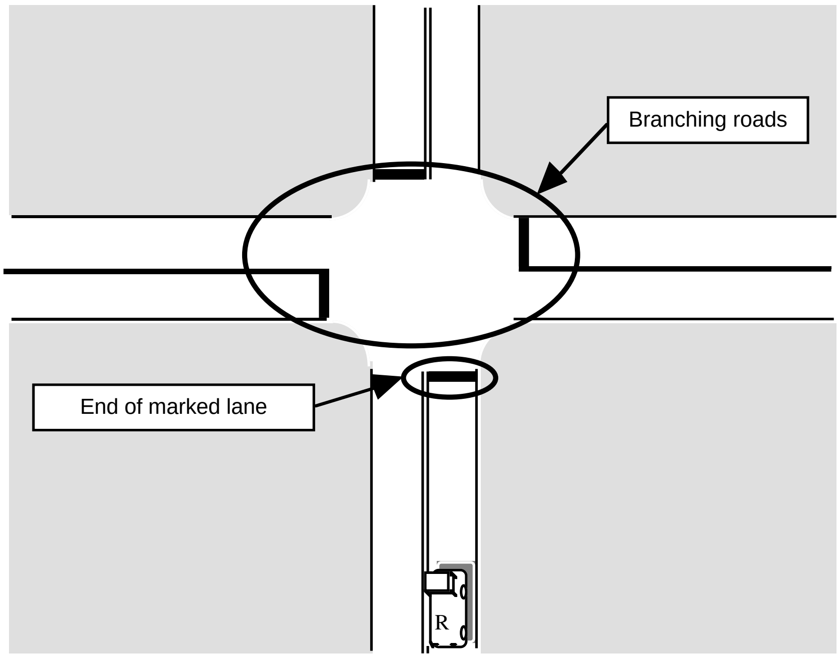


Figure 3-6: An intersection without traffic; detection by tracking lanes.

corridor no longer follows clearly marked lanes. The perception system must recognize the other roads at the intersection and identify the one on which the robot's route leaves. Figure 3-7 shows that the perception system must identify the "left" road if the strategic route plan requires the robot to turn left at this intersection. Ulysses can then extend the corridor by creating a path through the intersection from

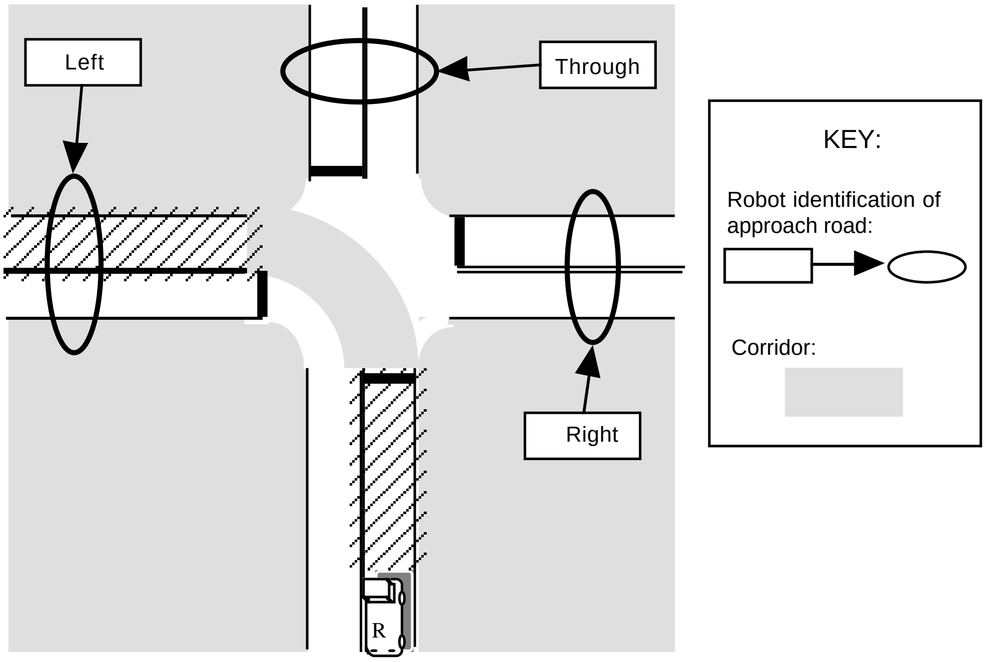


Figure 3-7: Finding the corridor for a left turn.

---

the end of the robot's approach road to the start of the proper lane in the road on the left.

With the corridor established through the intersection, Ulysses generates the same acceleration constraints as for the highway case. Figure 3-8 shows several examples of how these constraints may apply at an intersection. Constraints may apply to conditions before, within, and beyond the intersection.

The next task for the driving program is to determine the traffic control at the intersection. In this scenario the only important TCD's are traffic signals, Stop signs and Stop markings. Thus, as shown in Figure 3-9, Ulysses requests the perception system to look for signs and markings just before intersection and signal heads at various places around the intersection. When the intersection is beyond the given detection range for these objects, Ulysses assumes the worst and constrains the robot to stop at the intersection; however, within the detection range Ulysses assumes that all existing traffic control devices will be found.

Figure 3-10 diagrams the decision process for traffic signals required at this simple intersection. First, Ulysses must determine what the signal indication is. If there is an arrow in the direction of the corridor, the program takes this as the effective signal; otherwise, the illuminated solid signal is used. If the signal indication is red, Ulysses generates an acceleration constraint that stops the robot at the entrance to the intersection. No constraint is generated if the signal is green. If the signal is yellow, then Ulysses determines whether the robot can be stopped at the intersection using a reasonable braking rate. If so,

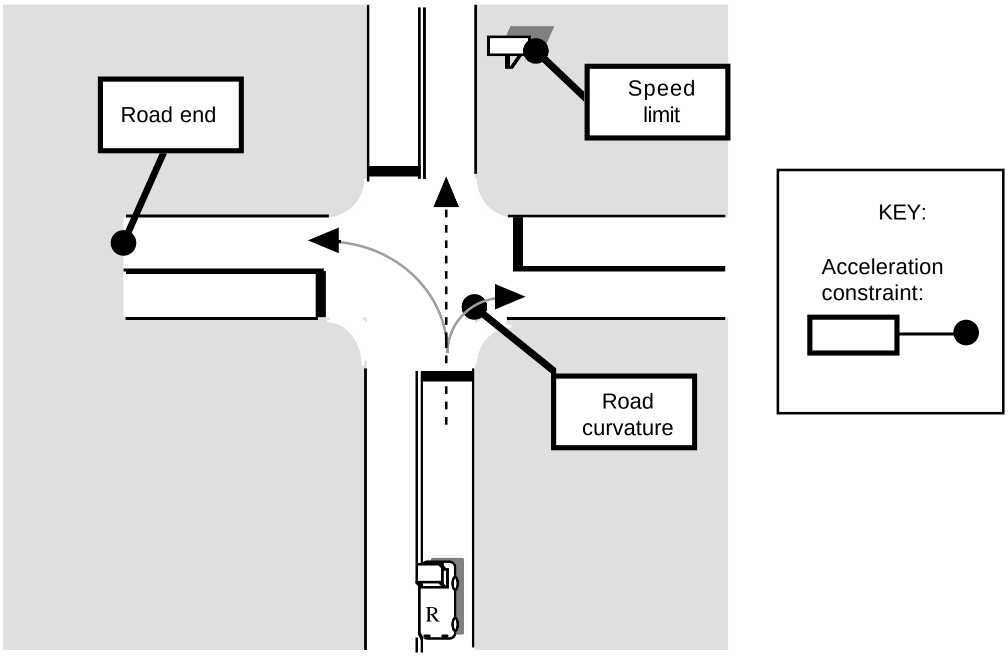


Figure 3-8: Examples of acceleration constraints from road features at an intersection.

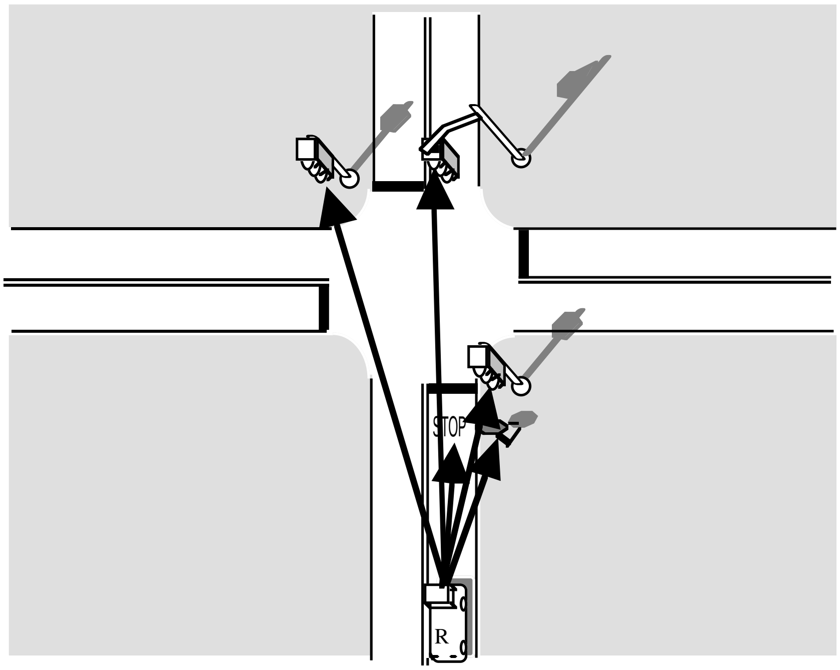


Figure 3-9: Looking for possible traffic control devices at an intersection.

then the program generates a constraint as if the light were red.

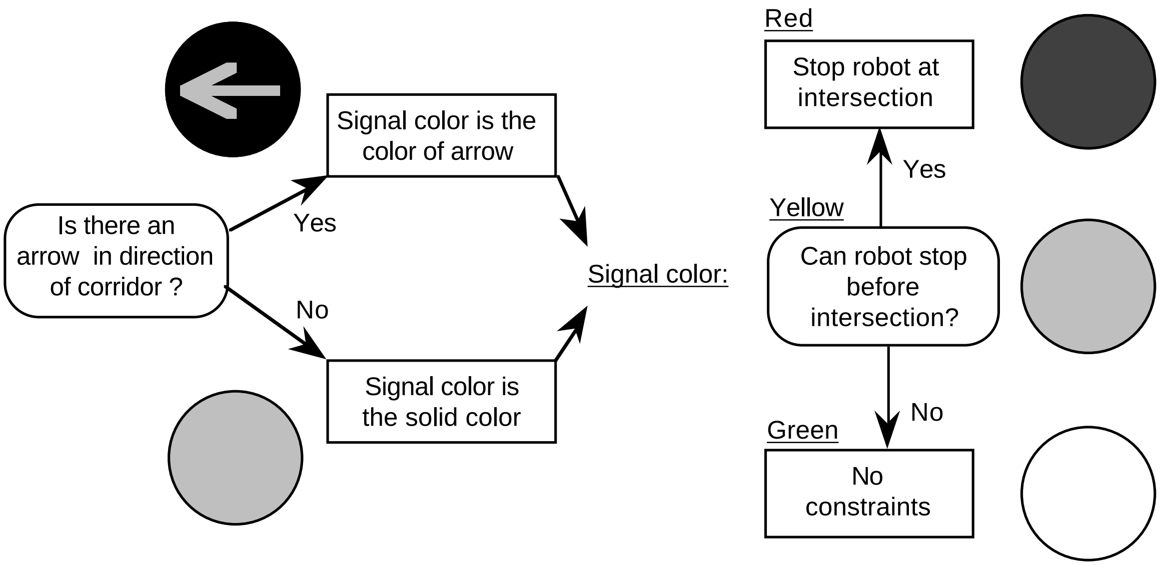


Figure 3-10: Traffic signal logic for intersection without traffic.

Stop signs require not only speed constraints, but a short sequence of actions. A vehicle approaching a stop sign must come to a complete stop, look for traffic, and then proceed when the way is clear. Figure 3-11 shows how Ulysses performs these actions with a "Wait" state variable. When a stop sign is

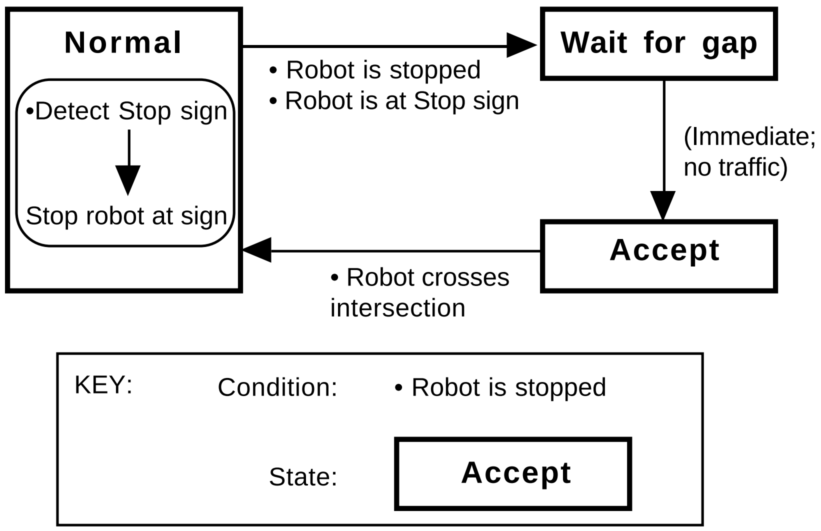


Figure 3-11: State transitions in Stop sign logic.

detected, Ulysses generates a constraint to stop the robot just before the sign. Later, Ulysses finds that the robot is stopped and that it is very near the sign, so it changes to a "wait for gap" state. In this scenario there is no traffic, so Ulysses immediately moves on to the "accept" state and releases the robot from this traffic control constraint.

3.1.3. An Intersection with Traffic

We will now add traffic to the simple intersection scenario. Ulysses considers three kinds of traffic at intersections: cars in the same corridor as the robot, which must be given headway; cross traffic blocking the intersection; and cars approaching the intersection on other roads which may have to be given the RoW. Human drivers are sometimes taught to watch the traffic behind them, in case another car is following too closely or approaching very fast. In such a case the driver could accept lower gaps or make more aggressive judgements at an intersection, thereby continuing through the intersection and not stopping quickly in front of the following car. Ulysses does not make such judgements, and thus does not look for traffic behind the robot.

Figure 3-12 illustrates the first kind of traffic that Ulysses considers. The driving program must

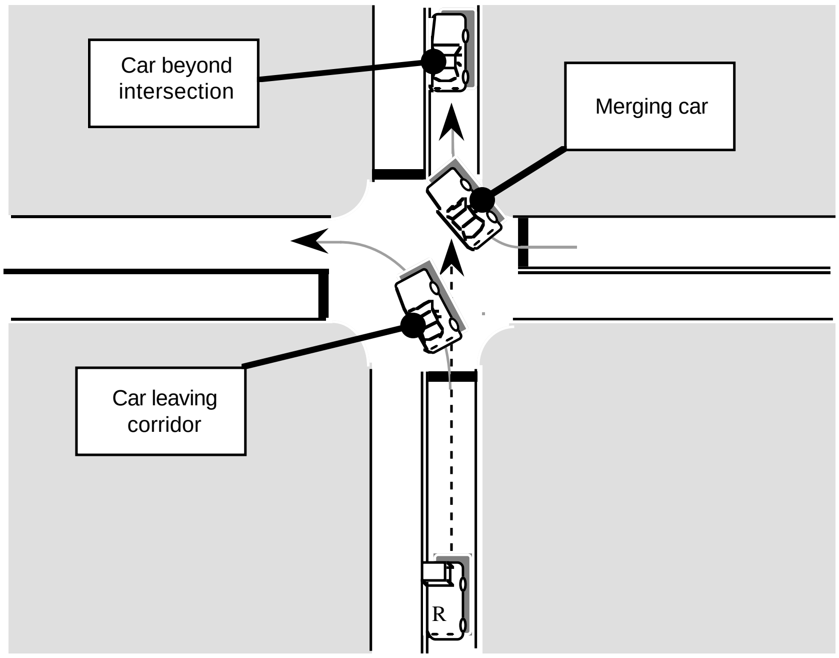


Figure 3-12: Potential car following conditions at an intersection.

generate car-following constraints from a lead car either in or beyond the intersection. If the lead car is turning off the robot's path, the robot must still maintain a safe headway until that car is clear of the path. Similarly, the robot must maintain a safe distance to cars that merge into its path.

The second kind of traffic that Ulysses looks for is cross traffic in the intersection. This includes cars sitting across the robot's path or about to cross it, as shown in Figure 3-13. Since the law requires that a vehicle yield the RoW to vehicles already in the intersection, Ulysses looks for such vehicles and stops the robot before the intersection if it finds one. Ulysses does not currently search beyond the closest car to find a gap. Future extensions to Ulysses may try to time the robot's arrival at an intersection to the presence of an upstream gap. However, this type of behavior could only be allowed if it were safe—that

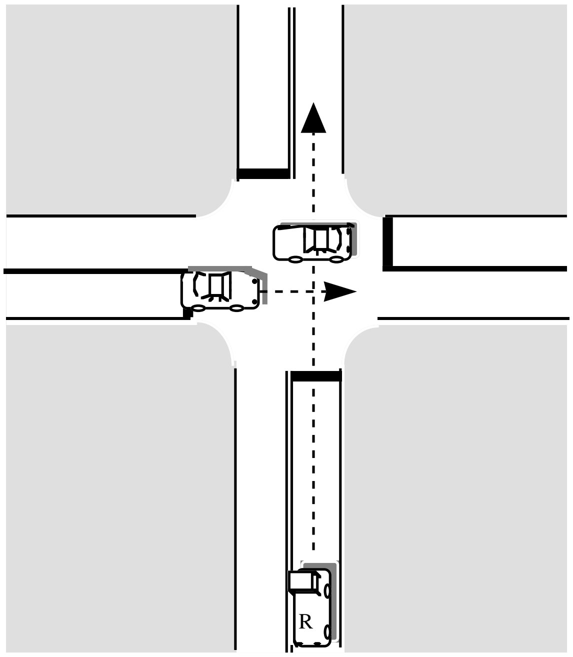


Figure 3-13: Cars blocking an intersection.

is, if the robot could still stop at the intersection if traffic changed unexpectedly.

The third type of traffic is traffic approaching the intersection on other roads. The driving program analyzes nearby TCD's and this approaching traffic to determine if the robot has the RoW. After RoW is determined, Ulysses may generate an acceleration constraint and change the Wait state of the robot. Figure 3-14 illustrates this process.

Traffic Control Devices. First, Ulysses detects and analyzes signs, markings and traffic lights. In addition to the TCD's needed for the previous scenario, the program now must recognize Yield signs and determine how many lanes are in the roads. The lane count is used to determine if the robot is on a "Minor Road"—that is, if the robot's road has only one or two lanes and the other road has more lanes. In the United States traffic on a Minor Road generally yields the RoW to other traffic, so in effect has a yield sign. This is a practical rule, not a legal one.

Table 3-2 summarizes the meanings of the various TCD's. The figure groups the TCD's into four equivalence classes. Yellow signals are treated as either red or green, depending on the robot's state and speed. If the robot is in a "wait for gap" state, a yellow signal is always treated as green. This practical rule allows the robot to proceed through an intersection just before the signal turns red. This allowance prevents the robot from being stuck forever in heavy traffic with no gaps. Cars with a given TCD normally yield the RoW to traffic with a TCD in a higher priority class, as will be explained below. Note that some actions in the figure do not depend on other cars; for example, red lights always stop the robot.

Approaching traffic. The second part of RoW evaluation is analyzing the robot's situation relative to other cars. This situation depends on the position and speed of the robot and other cars, and the relative

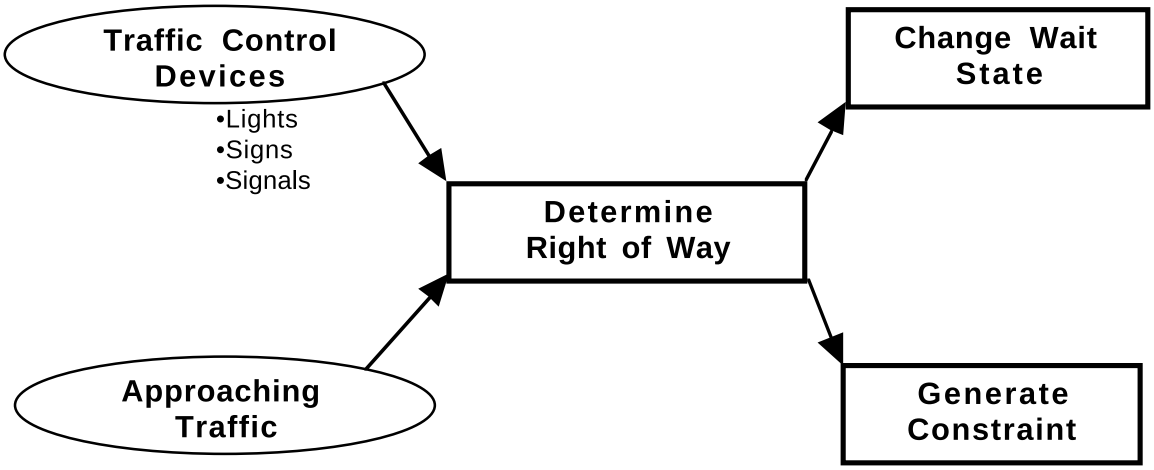


Figure 3-14: Generation of constraints from traffic and traffic control.

position of the approach roads. Figure 3-15 shows how this information is combined with traffic control information to decide the RoW with respect to each car. For each approach, Ulysses looks up the road to find the first car in each lane. The road to the right is ignored if the robot is turning right, as is the directly opposing road unless the robot or approaching car are turning left. Looking "up the road" essentially requires the perception system to find a corridor along a different road. If the perception system does not find a car, Ulysses makes RoW decisions as if there were a car at the range limit of the sensors or of sight distance. This hypothetical car is assumed to be going a little faster than the prevailing speed of traffic. If the approaching car is going too fast to stop before the intersection (assuming a nominal braking rate), Ulysses always yields the RoW. On the other hand, if the car is very far away from the intersection, Ulysses will ignore it. "Very far away" means that the time it will take the robot to cross the intersection at its current acceleration rate is less than the time it will take for the other car to reach the intersection.

If none of the above conditions exist, then Ulysses further analyzes RoW based on traffic control devices and the movements of the other cars. First, Ulysses guesses the traffic control for the approaching car. To do this, the perception system must look at the side of the approach road for a sign (the back side of a sign). If there is no sign, and the robot is facing a traffic light, Ulysses assumes that the approaching car also has a traffic light. If the car is approaching from a cross street, the light is assumed to be red (when the robot's is green), and otherwise green.[1] Once the program has an estimate of the approach car's traffic control, it compares the control to that facing the robot. If they are not equivalent, then the vehicle with the lowest priority signal is expected to yield the RoW.

If the traffic control for the approaching car and the robot are equivalent, Ulysses performs further analysis of the situation. If the approaching car is close to the intersection and stopped, the program assumes that it is waiting for a gap in traffic. In the case of a Stop sign, the robot takes the RoW if the

[1]This is a particularly clear example of how the explicit, computational rules in Ulysses elucidate the information requirements of driving. Drivers must make assumptions about the TCD's controlling other cars, and sometimes make errors if their assumptions are wrong.

Increasing priority

Traffic Control	Action
Green signal OR Nothing	Use RoW rules (see Figure 3-15)
Yield sign, OR Nothing and on Minor Rd	Yield to traffic with Green or no traffic control; otherwise use RoW rules (see Figure 3-15).
Stop sign, OR turning right at Red signal when allowed.	Stop at intersection; then proceed using RoW rules (see Figure 3-15)
Red signal	Stop at intersection
Yellow signal	IF Wait-state is "normal" AND robot can stop at intersection THEN treat as Red signal ELSE treat as Green signal.

Table 3-2: Actions required by four classes of Traffic Control Devices.

approaching car is not yet waiting. Otherwise, if one or both vehicles are moving, then the robot yields the RoW to cars on the right, and to cars ahead when the robot is turning left. If both the robot and the approaching car are stopped and waiting for a gap, then the drivers must use some deadlock resolution scheme to decide who goes first. In human drivers we have identified four such schemes, as illustrated in Figure 3-16. Humans use different schemes, depending on the local custom and the driver. Ulysses bases its decision only on road configuration; it will use the first-arrive, first-leave technique as well when we better understand how to recognize the same cars in images taken at different times.

After determining the requirements of the traffic control devices and evaluating the traffic situation, Ulysses may create an acceleration constraint and update the wait state. Figure 3-17 shows the conditions for changing states and for constraining the robot to stop at the intersection. Note that once the program has decided to accept a gap and take RoW, only the presence of new cars in the robot's path will cause it to stop again.

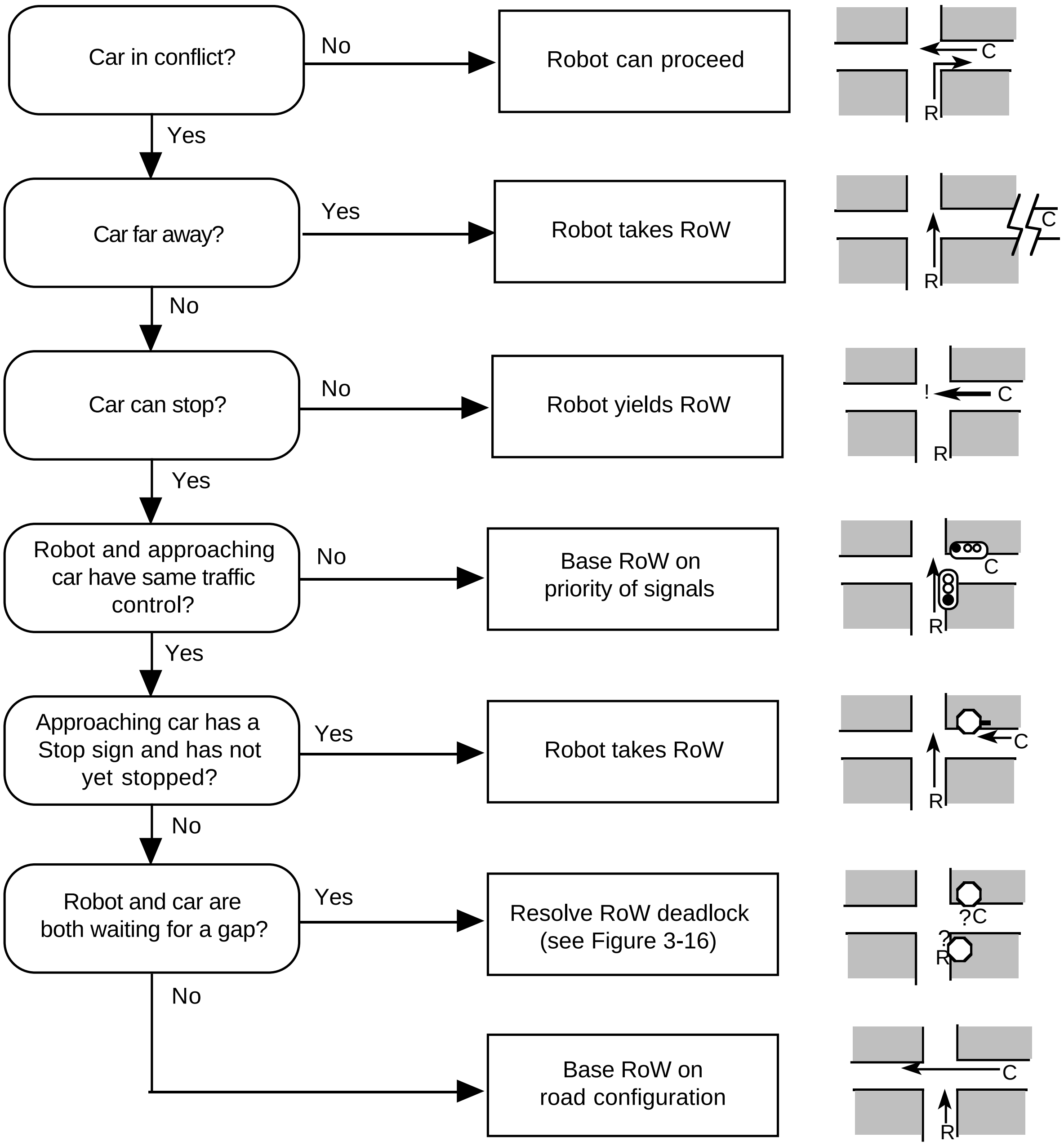


Figure 3-15: Right of Way decision process.
'R' is the robot, 'C' is the other car.

3.1.4. A Multi-lane Intersection Approach

The driving situations considered so far have not included streets with multiple lanes. With this complication, Ulysses must generate lane use constraints and make lane selections. We first consider the simple case of multiple lanes at an intersection, as depicted in Figure 3-18. In this scenario there is no other traffic on the same road as the robot.

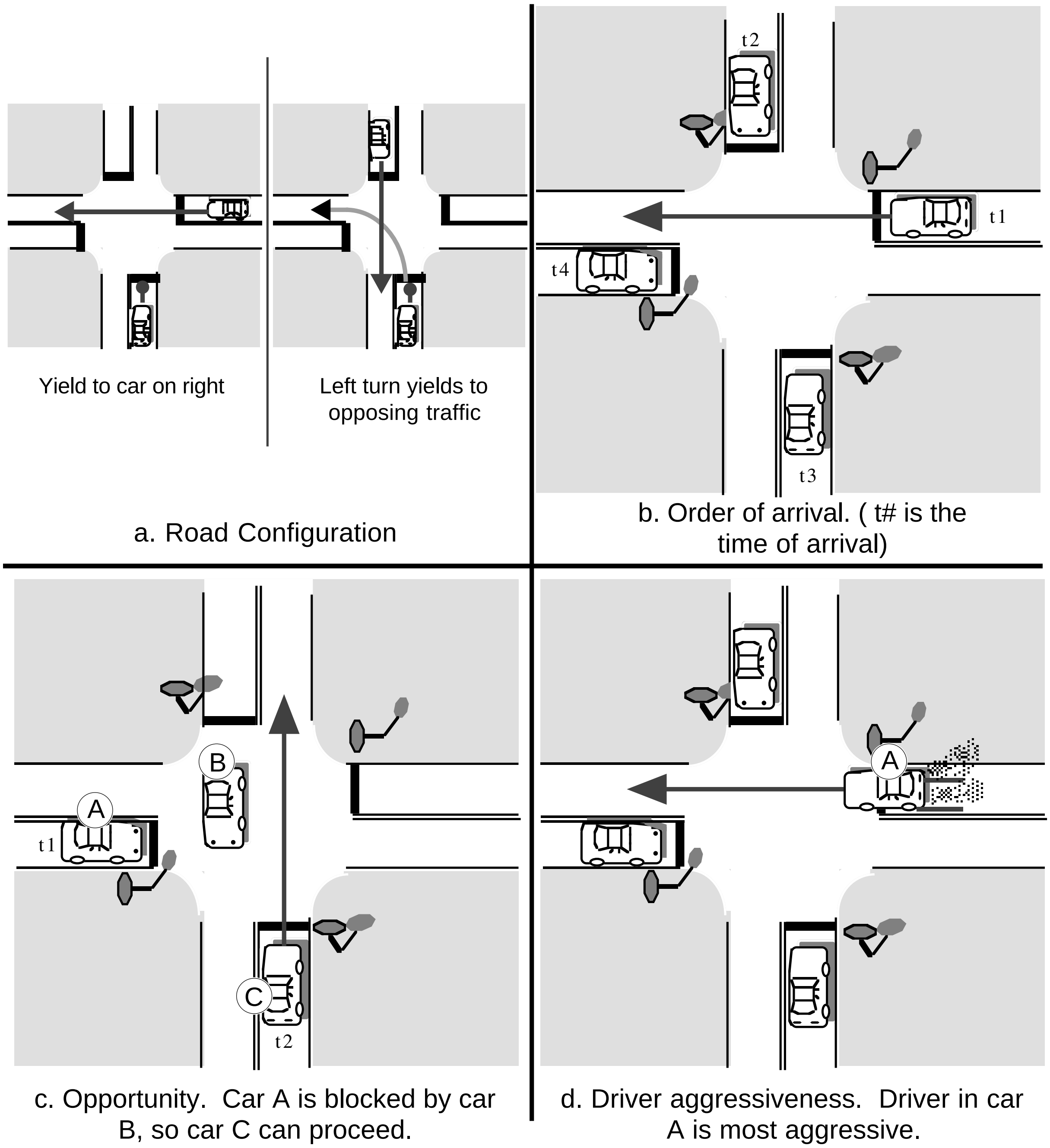


Figure 3-16: Right of Way deadlock resolution schemes.

As the robot approaches an intersection, constraints on turn maneuvers from different lanes—which we generally refer to as lane *channelization*— determine the lane-changing actions that Ulysses can take. Ulysses determines if the channelization of the robot's current lane is appropriate, and looks for a better lane if it is not. Channelization is estimated first by finding the position of the current lane at the

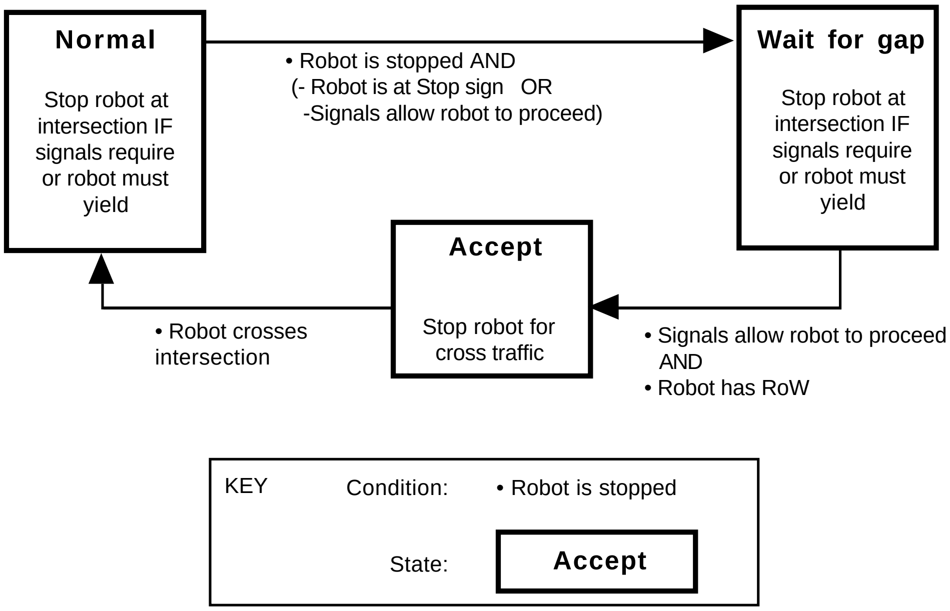


Figure 3-17: Changes in the Wait state at an intersection.

---

intersection. If it is the left-most lane, then the program assumes that a left turn is allowed from the lane; similarly with right turns if it is the right-most lane. Through maneuvers are allowed from any lane by default. Ulysses also looks for signs and road markings to modify its assumptions about channelization.

If the robot's intended maneuver at the intersection is not allowed by the channelization of the robot's current lane, Ulysses generates an acceleration constraint to stop the robot before the intersection. Figure 3-19a illustrates this constraint. Ulysses estimates the minimum distance required to change lanes (from the width of the lanes and the robot's minimum turn radius) and stops the robot that far from the intersection. The program next decides in which direction the robot should move to correct the situation, and looks to see if there is a lane on that side of the robot. If there is a lane, Ulysses constrains the robot to take a lane-change action in that direction. As a practical matter, solid lane lines are ignored, because making a turn from the correct lane is more important than strictly obeying these lane-change restrictions. Since there is only one allowed action, no selection preferences need be considered.

Once a lane-changing action is started, Ulysses makes sure that the robot completes the maneuver before it enters the intersection. The program does this by generating an acceleration constraint that slows the robot to a low speed at the minimum lane changing distance described above. Figure 3-19b shows this constraint. The low speed is set to keep the lateral acceleration on the robot below a certain threshold when turning with minimum radius. The lane-changing constraint is conservative, but our model of operational control does not provide Ulysses with enough information about the lane changing maneuver to predict just when it will be complete (see description of operational level, below). By driving at a speed low enough to use the minimum turn radius while far enough from the intersection, Ulysses guarantees that the maneuver can be completed.

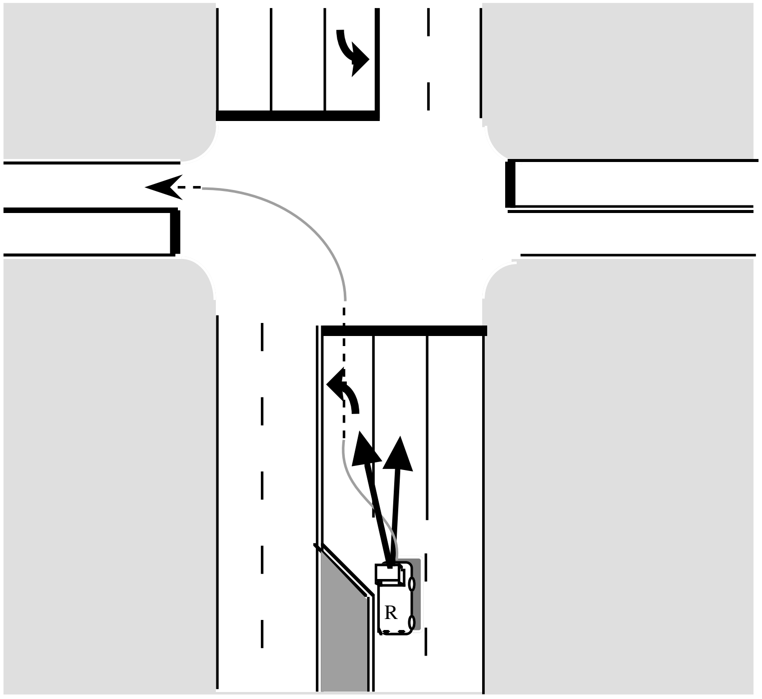
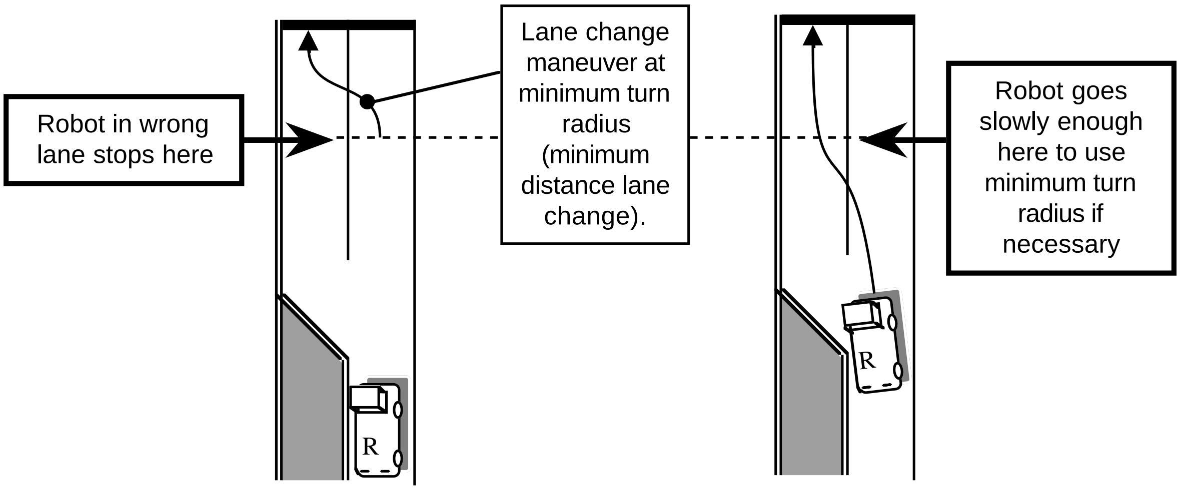


Figure 3-18: Robot observation of lane positions and markings at a multiple lane intersection approach.



a) Constraint before lane change b) Constraint during lane change

Figure 3-19: Channelization acceleration constraints.

With multiple lanes the driving program recognizes that there may be different signal indications for different lanes. Ulysses looks across the intersection from right to left and attempts to find all of the signal heads. If the robot is turning, Ulysses uses the indication on the head on the side of the turn. Otherwise, Uysses tries to find the head most nearly over the robot's lane.

When Ulysses decides to initiate a lane change, it sets a Lane Position state variable to Init-Left or Init-Right appropriately. Figure 3-20 shows these state transitions. The program does not consider lane

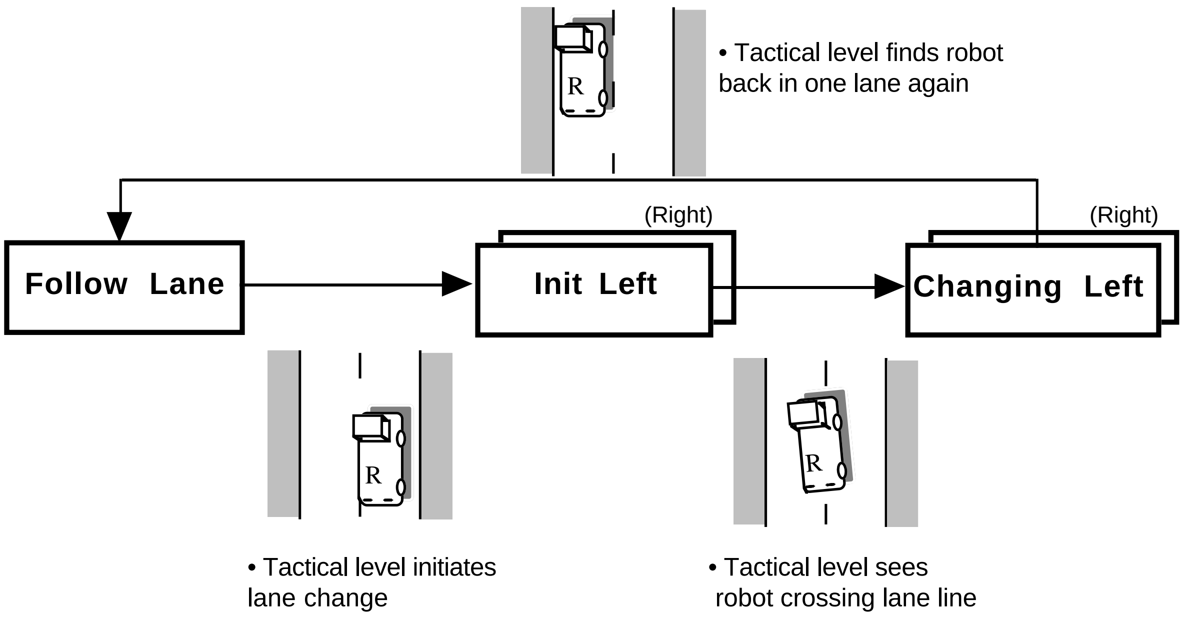


Figure 3-20: State changes during lane changing.

selection in future decision cycles while in one of these states. This state variable also helps to guide the perception functions when the robot is between lanes. Once a lane change is initiated, the operational control subsystem moves the vehicle to the new lane without further direction from the tactical level. When the robot is straddling a lane line, the state changes to Changing Left (Right). When tactical perception no longer sees the robot between lanes, the state reverts to Follow Lane.

3.1.5. A Multi-lane Road with Traffic

The next scenario is a highway far from an intersection. Unlike the first scenario above, this one is complicated by multiple lanes. Figure 3-21 shows some of the new considerations involved. First, a lane may end even without an intersection. Ulysses detects this in much the same way as it detects a complete road end, and creates an acceleration constraint to stop the robot before the end of the lane. This constraint disappears after the robot changes lanes. Multiple lanes also make car following more complex. If the car in front of the robot is between two lanes, Ulysses maintains a safe headway to that car but also looks for another car in the lane. If the robot is itself changing lanes, the program looks ahead for cars in both lanes until the robot has cleared the old lane.

The three lane actions available to the robot are following the current lane, moving to the right, or or

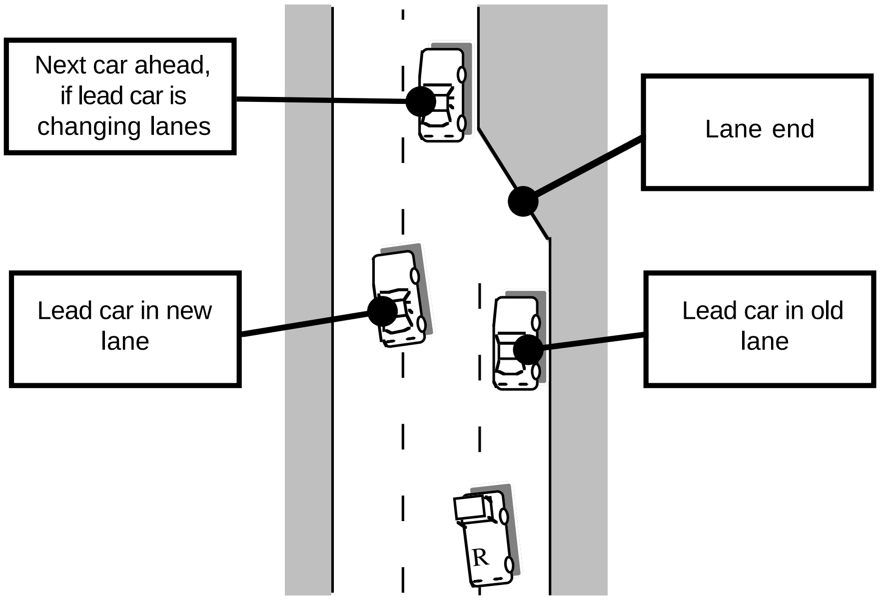


Figure 3-21: Multiple lane highway scenario.

moving to the left. Moving to the left does not currently include moving into the opposing lanes; thus Ulysses cannot yet overtake when there is only a single lane in each direction. Ulysses generates contraints and preferences for changing lanes if certain traffic conditions exist. Figure 3-22 shows the important conditions: adjacent lanes, gaps in traffic in adjacent lanes, blocking traffic in the robot's lane

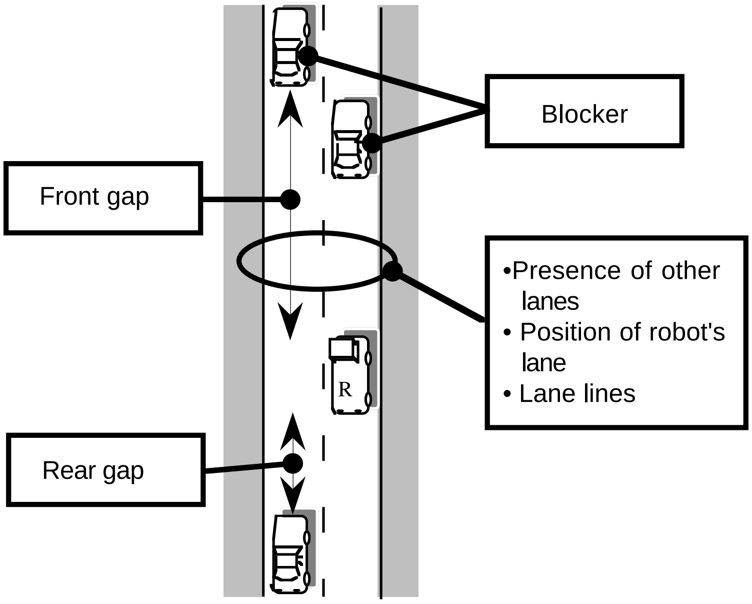


Figure 3-22: Traffic objects that affect lane changing decisions.

and blocking traffic in adjacent lanes. In this scenario, the program does not need to consider channelization.

Table 3-3 shows the constraints and preferences for each traffic condition. The absence of adjacent

CONDITION		SET OF ALLOWED ACTIONS	PREFERRED ACTION	Increasing Priority
Blocking car (Robot blocked by traffic in lane*)	• L - C > T_1 Broken lane line	All	Move Left	
	• R - C > T_1 Broken lane line	All	Move Right	
	• C > R, L	All	Keep Same	
Gap for merging	•No gap in traffic to left	(Move Right, Keep Same)	—	
	•No gap in traffic to right	(Move Left, Keep Same)	—	
Available lanes	•There is no lane to the left	(Move Right, Keep Same)	—	
	•There is no lane to the right	(Move Left, Keep Same)	—	
	•(There is a lane to the right) AND (R $\geq$ C)	All	Move Right	
Default		All	Keep Same	

KEY		
	* :	True when (Car following accel) < (Other accel constraints) - T_2
	T_i :	Arbitrary threshold value.
	L, R, C:	Max. acceleration allowed in the lane to the Left, lane to the Right, and Current lane, respectively

Table 3-3: Lane action preferences for a highway with traffic.

lanes or broken lane lines or gaps in traffic eliminates the option to move to adjacent lanes. Ulysses generally prefers the rightmost lane to others. Traffic *blocks* the robot if the acceleration allowed by car following is significantly less than acceleration allowed by other constraints. Blocking traffic creates a preference for an adjacent lane if the acceleration constraint in that lane is significantly higher than the car following constraint in the current lane. Ulysses estimates the allowed acceleration in the adjacent lane by hypothesizing what the car following constraint would be and combining this with the speed limit, lateral acceleration and road end constraints. The program combines the constraints from various conditions by taking their logical intersection—retaining only actions that appear in all constraints. If there is more than one lane action available after constraints have been combined, preferences determine which action is chosen. Preferences due to blocking traffic are given priority over the rightmost lane preference.

Ulysses judges gaps for changing lanes based on two criteria: first, the deceleration required of the robot to create a safe headway to the lead car in the other lane; and second, the deceleration required of the following car to leave space for the robot. The program does not search for a gap farther back in traffic if the adjacent one is not big enough. However, if the robot is blocked, traffic in the adjacent lane will tend to pass the robot, thereby providing new merging opportunities. Ulysses also does not have the ability to use a physical "language" (beyond directional signals) to communicate to other drivers that it is anxious to change lanes. It is thus possible that the robot will never find a gap if traffic is congested.

3.1.6. Traffic on Multi-lane Intersection Approach

The next scenario combines the previous two multi-lane cases. This situation, illustrated in Figure 3-23, adds channelization constraints to traffic considerations. When the adjacent lane does not permit

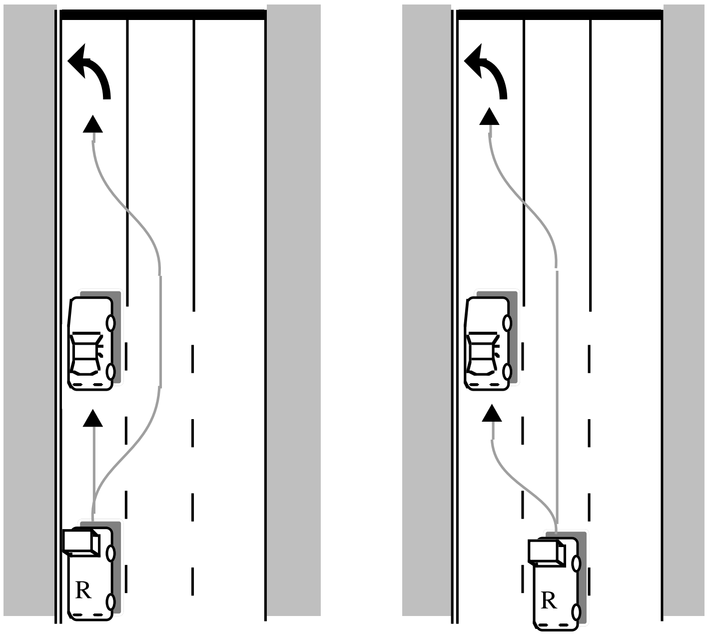


Figure 3-23: Decisions at a multiple lane intersection approach with traffic.

the robot’s next turn, Ulysses must decide whether a pass can or should be made. Similarly, if the robot needs to move to a turn lane that has traffic in it, Ulysses must decide whether to merge immediately or wait. Table 3-4 lists the constraints and preferences that must be added to those in Table 3-3. When combining preferences, Ulysses gives the channelization demands the highest priority unless the robot is far from the intersection.

When the robot is near the intersection ($d < D_{\text{Far}}$ in Table 3-4), it is not allowed to change lanes to

CONDITION		SET OF ALLOWED ACTIONS	PREFERRED ACTION	PRIORITY
Channelization				
• $d < D_{Min}$		Keep Same	—	Highest
• $D_{Min} < d < D_{Far}$	Current lane OK	(Keep Same, Move to <OK lane>)	—	
	Current lane not OK	(Keep Same, Move to <OK lane>)	Move to <OK Lane>	
• $d > D_{Far}$		All	Move to <OK Lane>	Just below **Blocking Car** in Table 3-3
• No intersection visible		All	—	—

KEY		
	d :	Distance from robot to intersection
	D_{Min}:	Minimum distance needed to change lanes.
	D_{Far}:	Range of influence of intersection on upstream traffic.
	OK:	Channelization allows (or is closest to) robot's intended route

Table 3-4: Lane action constraints and preferences at an intersection with traffic.

move away from an acceptable ("OK") lane. Thus Ulysses will not attempt to pass a blocking car near an intersection unless multiple lanes are acceptable. This restriction may cause the robot to get stuck behind a parked vehicle, but it avoids the problem of deciding whether the lead vehicle is really blocking the road or just joining a queue that extends all the way to the intersection.

The "Far from intersection" ($d > D_{Far}$) condition in Table 3-4 depends on a distance threshold D_{Far}. When the intersection is at least D_{Far} away from the robot, Ulysses can pass blockers rather than staying in a turn lane. The threshold distance is properly a function of traffic—how far traffic is backing up from the intersection, whether there are gaps in the traffic ahead of the blocker, and how long (distance) the robot would take to complete a pass. Ulysses determines D_{Far} by estimating passing distance and adding a margin to cover the other conditions. The margin is a function of the speed limit, but does not explicitly incorporate downstream queuing conditions. In our simulated world we have found that the margin needed to minimize inappropriate passing maneuvers puts the robot in the "near intersection" condition soon after it detects the intersection.

The presence of traffic introduces additional factors into the tests of blocking conditions. When Ulysses determines what acceleration is possible for the robot in an adjacent lane, it must consider constraints from lane channelization and intersection traffic control. In effect, Ulysses must hypothesize the robot in the adjacent lane, look ahead in a new corridor, and recompute all of the acceleration constraints from that lane.

When the robot wishes to change lanes because of channelization, but cannot because that lane change action is not allowed (for example, because a gap in traffic is not available), Ulysses changes the robot's Wait state to "wait for merge gap." This action signals a standing desire to change lanes. When the robot is in this state, Ulysses adjusts the calculation of the car-following constraint so that extra space is left in front of the robot for maneuvering.

3.1.7. Closely spaced intersections

The final complication comes with multiple intersections spaced closely enough together that they all affect the robot. Figure 3-24 depicts multiple intersections ahead of the robot and on cross streets. The

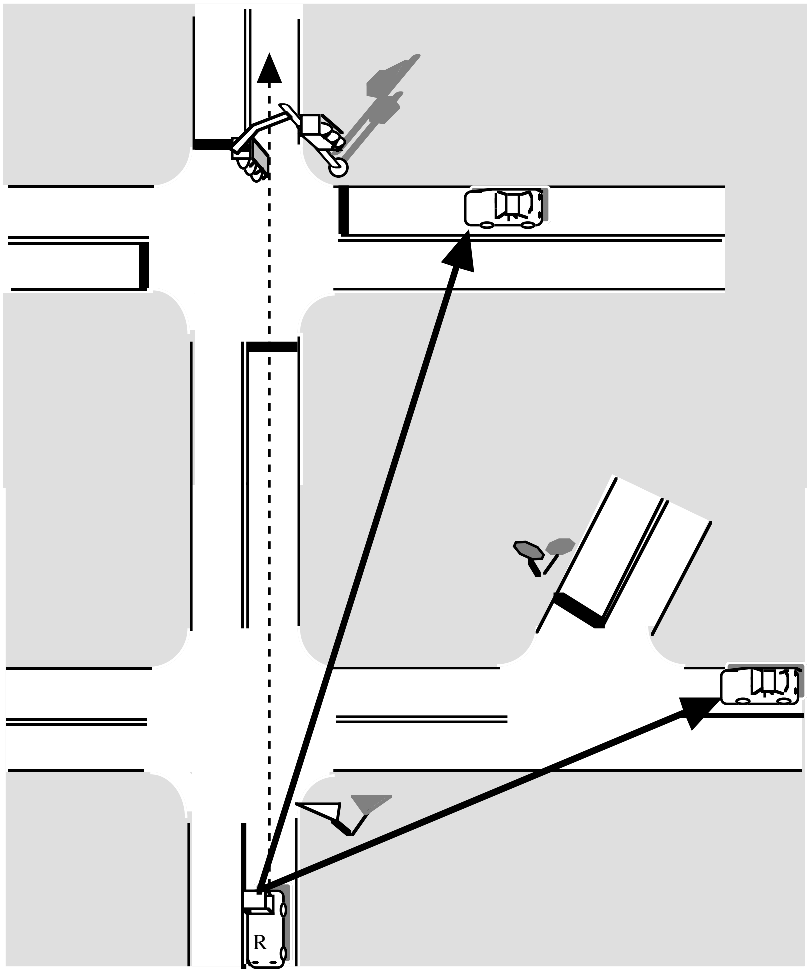


Figure 3-24: Visual search through multiple intersections

driving knowledge already described is sufficient to get the robot through such a situation. This scenario illustrates, though, the necessity of tracing a corridor ahead through multiple intersections, analyzing traffic and traffic control at every intersection along the corridor, and looking through intersections on cross streets for approaching cars.

3.2. Tactical Perception

Tactical driving decisions require information about roads, intersections, lanes, paths through intersections, lane lines, road markings, signs, signals, and cars. Our driving model assumes that the robot has a perception subsystem that can detect these traffic objects and determine their location and velocity. In addition, the perception system must estimate the distance to objects, the distance between objects, and the velocity of cars. There is more perception at the operational and strategic levels, but we do not address it in this model.

Tactical driving requires information about spatial relations between objects. When Ulysses looks for an object, it is really interested in objects that have a specific relation to another object. Figure 3-25 illustrates this concept. In the figure, Ulysses is at a point in its analysis where it is looking for *cars* on the *right-hand approach road* at the *second intersection* ahead. There are at least two ways to find the objects in the desired relations. One way would be to detect all cars, and then test each car to see if it was on the road to the right at the intersection. "The road to the right" would itself be found by finding all roads and checking to see if they connected to the the robot's road at the intersection in question. This method would be very difficult because there may be many cars and roads in the robot's field of view, and each one requires significant computation to find and test.

Ulysses uses a different technique to relate objects to one another. The perception subsystem uses the reference object and a relation to focus its search for a new object. In the example above the robot would track the corridor ahead to the second intersection, scan to the right to find the approach road, and then look along the road for a car. The car detection process is thus limited to a specific portion of the field of view, determined by the location of the road. All objects found are identified implicitly by their current relation to the robot. (This object identification method has been called "indexical registration" by Agre and Chapman [2].) Ulysses gathers almost all of its data using these *perceptual routines*. Table 3-5 lists the perceptual routines used by the program. Although it may seem that driving decisions could be made with simpler, more general sensing functions, Figure 3-25 shows that very simple sensing is not sufficient; "detect converging objects to the right," for example, cannot distinguish between the two intersections.

The perception subsystem uses *markers* to provide reference points for the perceptual routines. Our simple model of sensing currently assumes that one image—a snapshot picture of the world—is taken each time Ulysses repeats its decisions. Markers are placed in an image as a result of running perceptual routines that can mark cars, lane endings or beginnings, signs, markings, signal heads, or other locations. The markers are essentially names that different perceptual requests can use for the objects in an image. In the example of Figure 3-25, various routines would put markers in the corridor where it enters and leaves intersections. When Ulysses analyzed the second intersection, a routine would mark the approach roads. Finally, when Ulysses had to look for the approaching car at that intersection, it would start a car detection routine and provide the marker for the approach road on the right.

Even using the routines described above, perception remains very difficult for a robot. Tactical perception in Ulysses is a focus of our ongoing research.

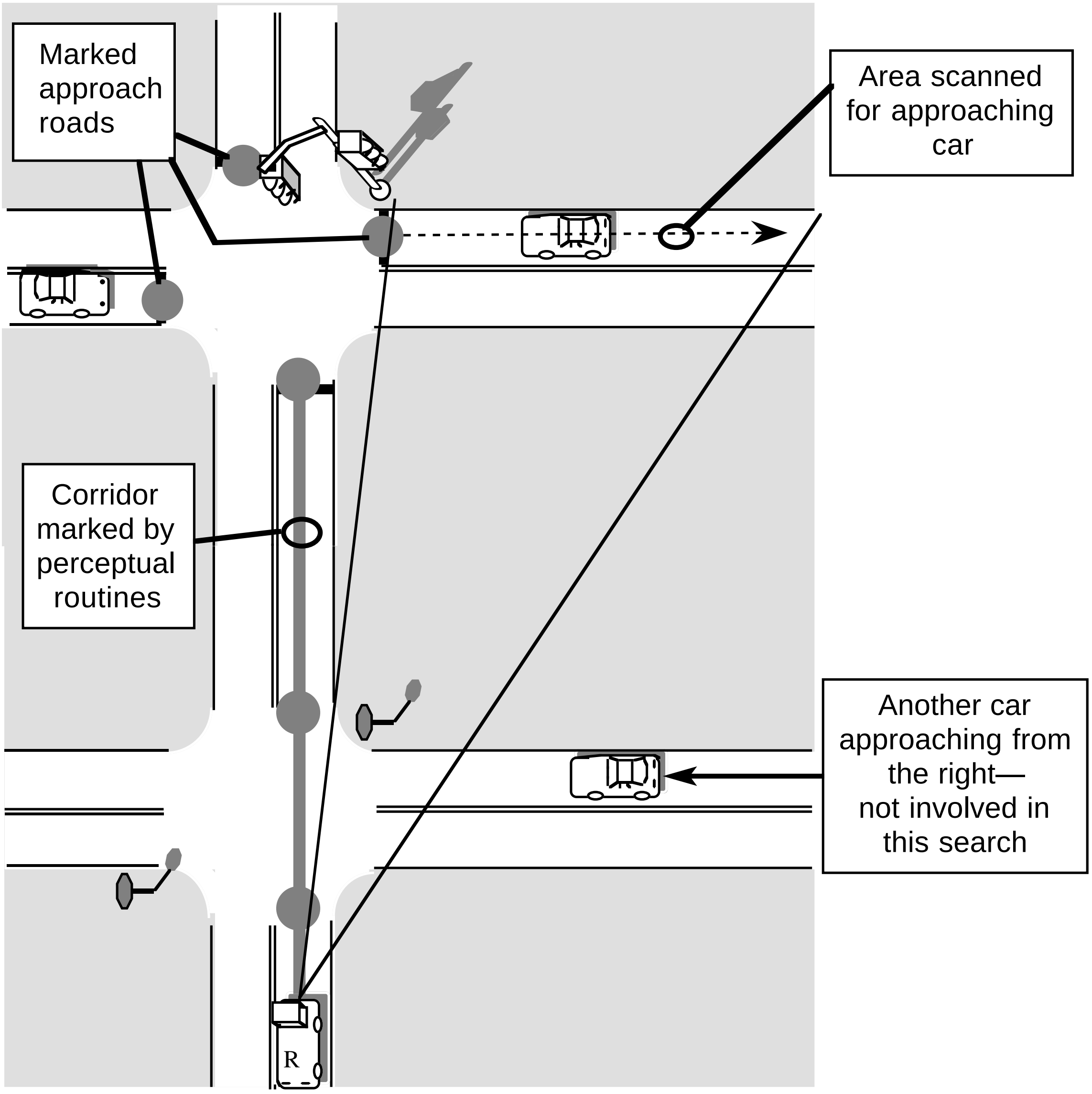


Figure 3-25: Searching for a car approaching a downstream intersection from the right.

3.3. Other Levels

3.3.1. Strategic Level

The strategic level of our model is very simple and serves only to provide a route plan to the tactical level. Figure 3-1 shows that perception at the strategic level looks for landmarks that mark the progress of the robot along the route. Ulysses currently assumes that the route at every intersection is provided by some map-based planner, and therefore uses only intersections as landmarks. A more sophisticated route maintainer would use street signs, road configurations, buildings, etc. to locate the robot in the map.

Find current lane	Find next car in lane
Mark adjacent lane	Find crossing cars
Track lane	Find next sign
Profile road	Find next overhead sign
Find intersection roads	Find back-facing signs
Find path in intersection	Find signal
Find next lane marking	Marker distance

Table 3-5: Perceptual routines in Ulysses.

3.3.2. Operational Level

The operational level makes emergency reactions and executes commands passed down from the tactical level. The operational level is assumed to have its own perception functions to detect emergencies and provide feedback to the actuator controllers. Emergency reactions include avoiding collisions with nearby objects and keeping the wheels away from road hazards. These are considered emergencies because the tactical level is responsible for finding problems in the distance. The tactical model uses four operational functions to execute the commands it generates. These functions are maintaining an acceleration, tracking a lane, changing lanes, and traversing an intersection.

Acceleration. The operational level keeps the vehicle's acceleration at the value commanded by the tactical level. Speed control is not used because an instantaneous change in speed requires an infinite impulse of force, while a change in acceleration requires only a step change in force. Vehicle dynamics dictate that an operational controller is more likely to achieve the latter. The choice of control variables is not crucial to the model. However, we expect that it might be difficult in some robot implementations to provide frequent command updates at the tactical level, so acceleration commands would prove to be more convenient.

Lane Tracking. The operational level is assumed to be able to drive the robot down a lane without intervention from the tactical level. This function involves only steering, as do lane changing and intersection traversal. As described earlier, the tactical level adjusts the speed independently to keep lateral accelerations within limits on turns.

Lane Changing. Lane changing requires the operational level to detect the adjacent lane and begin a double-curve maneuver to move the robot over. When the robot reaches the new lane, lane tracking automatically takes over. At most speeds, the steering angle is adjusted to limit the robot's lateral acceleration to a safe value. This constraint results in a lane change maneuver that takes a roughly constant amount of time, independent of speed. The tactical level can use this fact to estimate the length of a lane change. However, if the robot is changing speeds (e.g., braking while approaching an intersection), it is impossible to predict exactly how far the robot will travel during the maneuver. At low speeds, robot can only increase the steering angle to the physical stops of the vehicle. This angle determines the minimum lane change distance for a given road width.

Intersection Traversal. Since intersections generally do not have marked lanes, the operational system is required to find a path to the given departure lane and drive the robot along the path. As with the other operational functions, there are several ways this may be accomplished. We expect that the robot would follow an imaginary path created from general knowledge about the size of the intersection, and then start up lane tracking when the new lane became clearly visible.

3.4. Summary

Ulysses is a computational model of driving implemented as a computer program. It emphasizes the tactical level. Strategic functions are limited to providing a route plan step at every intersection. The operational level abstractly implements tactical commands.

Ulysses can drive the robot in traffic on multi-lane roads, through intersections, or both. Driving knowledge is encoded as constraints and preferences. Ulysses looks for lane endings, curves, speed limit signs, and other cars to constrain the robot's acceleration on a road. When approaching an intersection, the program directs the perception subsystem to look for signs, signals, markings, other roads, and approaching traffic as well as the general road features just mentioned. Ulysses determines the semantics of the traffic control devices and the road configuration and decides whether the robot has the right of way to enter the intersection. Stop signs and conflicting traffic cause the robot stop and then enter a waiting state until traffic clears or the signal changes. The program also evaluates lane channelization, potential speed increases in adjacent lanes, traffic in adjacent lanes, and lane position to decide whether the robot should change lanes. Additional acceleration constraints prevent the robot from approaching the intersection in the wrong lane or in the middle of changing lanes.

Ulysses uses demand-driven perception. The robot looks at the world frequently so that it can respond quickly to changing traffic situations. The model uses perceptual routines and visual markers to find traffic objects with the appropriate spatial relationships to the robot.

4. The PHAROS Traffic Simulator

Our motivation for creating a driving model was to describe how to drive an autonomous vehicle. Actually driving a vehicle requires that the driving model be made concrete. The model must specify how all knowledge is encoded and how information is processed. In the preceding section we described how this is done in our model. However, the proof of the pudding is in the eating— descriptions cannot drive real vehicles. Our first step toward using our model on a real robot has been to implement the model in a computer program, Ulysses, and drive a simulated robot in a simulated traffic environment. PHAROS is the traffic simulator we developed for this work.

There are several advantages to driving in simulation before trying to drive an actual vehicle. First, a simulator is flexible. It is possible to generate almost any traffic situation to test various kinds of driving knowledge. These situations can be recreated at will to observe how new driving knowledge changes a vehicle's behavior. Simulators are also convenient, because simulated driving is not dependent on working vehicle hardware, convenient test sites, or weather. Finally, simulated driving is safe and avoids the problem of placing other drivers at risk while testing the robot.

Figure 4-1 shows how PHAROS and Ulysses work together. PHAROS maintains a database

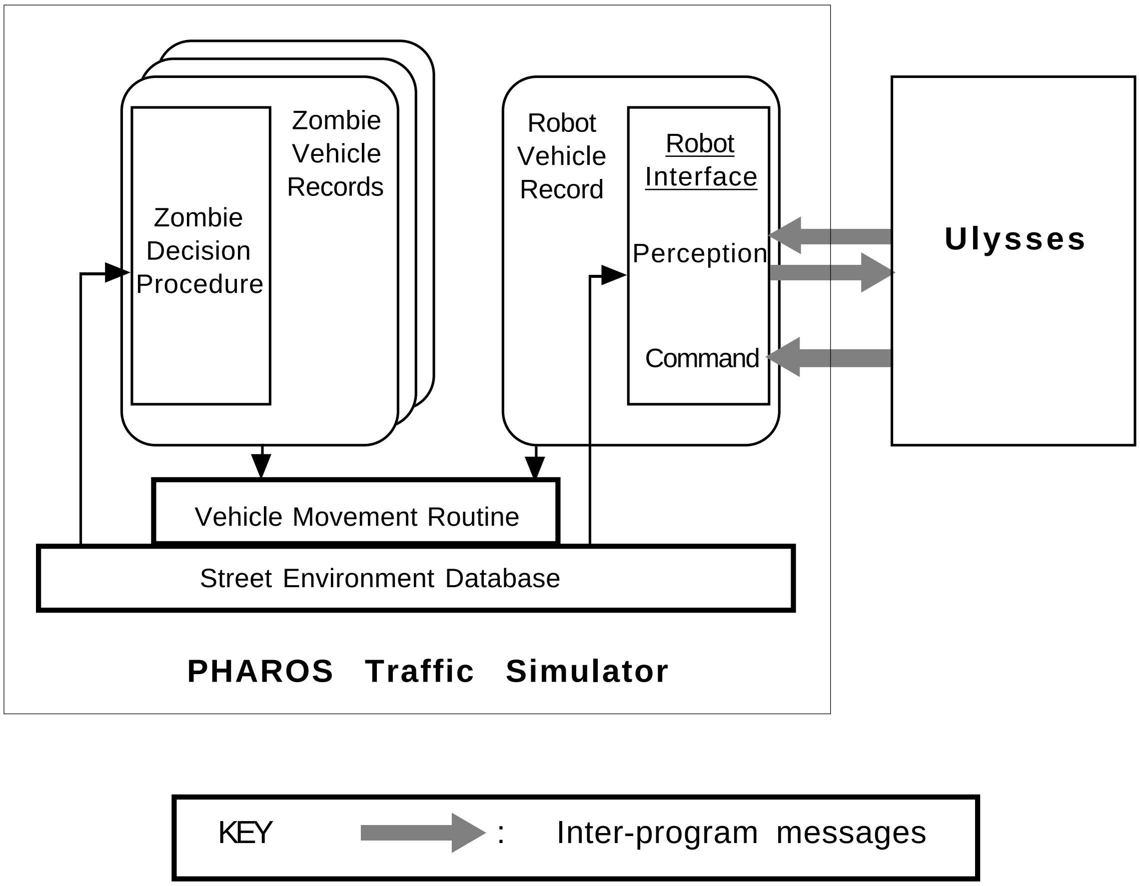


Figure 4-1: Ulysses control of a vehicle in PHAROS.

---

describing the street environment and records for each car. PHAROS also controls the behavior of the simulated cars, which we call *zombies*. The simulator executes a decision procedure for each zombie to determine its actions, and then moves it along the street. A robot is simulated by replacing the decision procedure with an interface to Ulysses. Ulysses runs as a separate program—sometimes on a different computer—and gets perceptual information by sending requests to PHAROS. The interface simulates the perceptual routines described earlier. After Ulysses has the information it needs, it sends commands through the interface to the vehicle record. PHAROS then moves the robot using the normal movement routine. The requests for (simulated) perception and the commands for (simulated) control are the *only* communications between Ulysses and PHAROS. This arrangement ensures that Ulysses cannot "cheat" by examining the internal state of the simulator.

The remainder of this section discusses the representation of the street environment, the driving decisions made by the zombies, and the general operation of the simulator.

4.1. The Street Environment

PHAROS is a model of the street environment. All models of the world use abstractions so they can capture just the important characteristics of a problem. Model designers must find a compromise between model accuracy and excess complexity. We have attempted to encode many characteristics of the street domain into PHAROS, including enough geometric information to study the perceptual requirements of driving. While PHAROS includes many traffic objects, it uses strong abstraction and structure to simplify their description and use.

PHAROS represents traffic objects with abstract symbols augmented by a few parameters to provide important details. Figure 4-2 illustrates how some objects are encoded. A curved road segment is described by a cubic spline (8 parameters) and a road width. A sign is represented by a type (one of 8), an identifying number, and a position along the road. The general shape and color of the sign is determined by its general type; the lateral distance from the road edge is assumed to vary only a little, so is not encoded at all. PHAROS describes a signal head by its location at the intersection (one of 5 generic positions), whether it is vertical or horizontal, the number of lenses, the symbols on the lenses, and the color of the lenses.

PHAROS groups traffic object symbols into structures to give them meaning for driving. It would be difficult to determine how streets connected or whether a car could move to an adjacent lane if the lane and line objects in the database were not organized. PHAROS connects objects to one another to form hierarchies and networks of related objects. Figure 4-3 shows how different street objects are connected to one another.

The abstraction and structure used in PHAROS limit the accuracy with which it can simulate the world. For example, PHAROS cannot easily simulate roads without marked lanes. PHAROS vehicles cannot abandon the lane abstraction to drive around a broken down vehicle at an intersection. However, our representation scheme does allow PHAROS to simulate many common traffic situations on highways and arterial streets. We also feel that the scheme could be extended to deal with situations that are beyond the current abstraction. Similarly, there are several types of traffic objects missing from PHAROS' world—pedestrians, cyclists, blinking signals, etc. Future versions of PHAROS may include such objects.

4.2. PHAROS Driving Model

The driving decisions made by the zombies are based on the logic in the Ulysses driving model. However, there are several differences between Ulysses and the PHAROS-controlled zombies. The key differences are in perception, interpretation, and speed control.

Perception. Zombies are not required to use simulated perception to get information from the simulated world. When PHAROS is running the decision procedure for the zombies, it uses the data structures directly. For example, to find a vehicle across the intersection from a zombie, PHAROS traces symbolic links from the zombie to its road, to the connecting road ahead, and to the rearmost car on that road. Tracing these pointers in the data structures is similar in concept to running perceptual routines, but much simpler in practice.

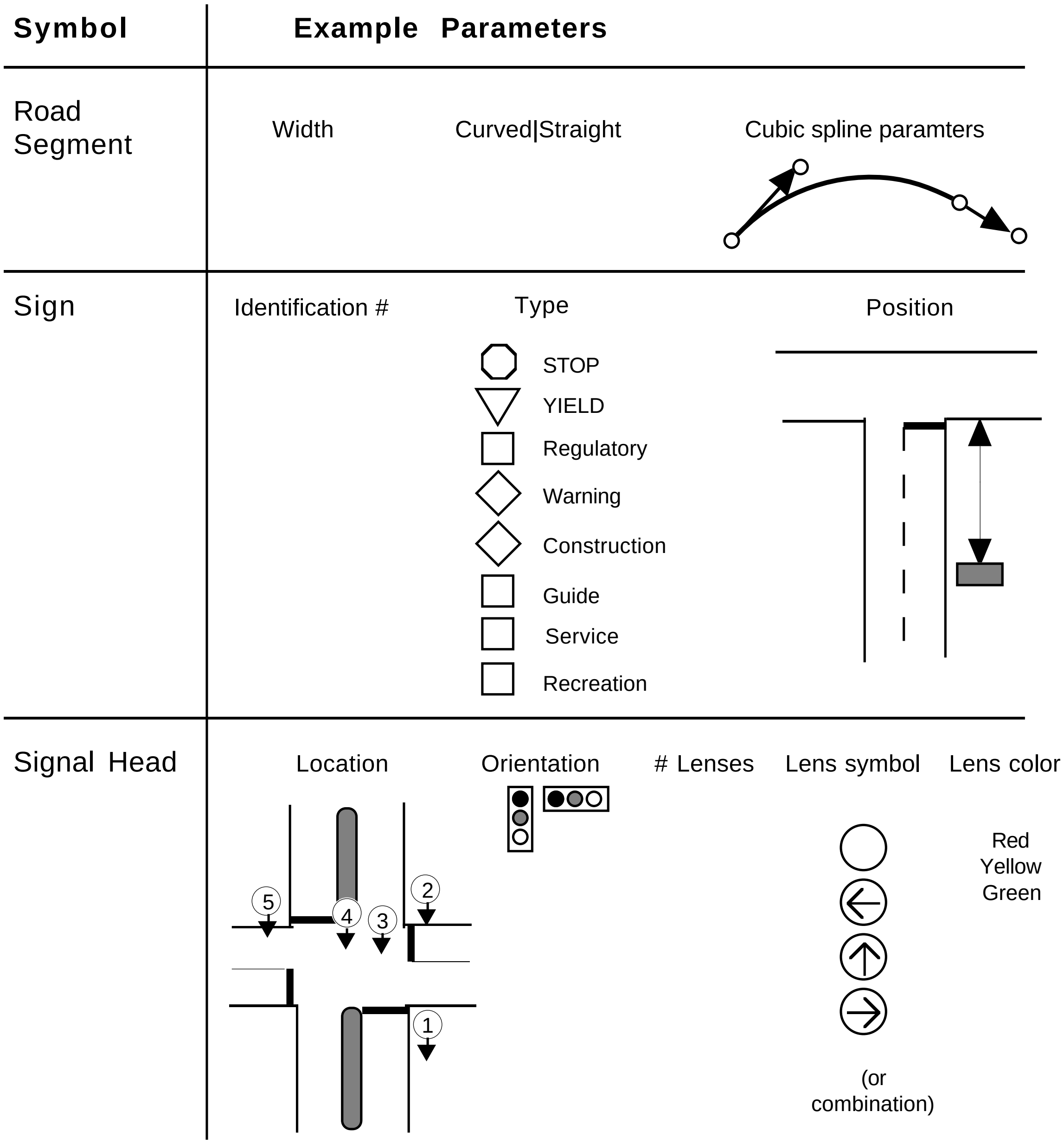


Figure 4-2: Examples of traffic object encodings.

Interpretation. Several traffic situations that Ulysses encounters are difficult to interpret. Some situations, such as determining the traffic signal for other approaches or the intended turn maneuver of another car, are problematical for Ulysses because they cannot be observed directly. This type of problem is simple for PHAROS, because zombies have access to the entire database and are not limited by what they can see. This access extends even to examining the internal state of other zombies. Other situations, such as deciding which signal head applies to the robot's lane, require several observations followed by a reasoned guess. Some situations are complex enough that we cannot yet describe how to recognize them. For example, we stated earlier that Ulysses cannot tell whether a blocking car in front of

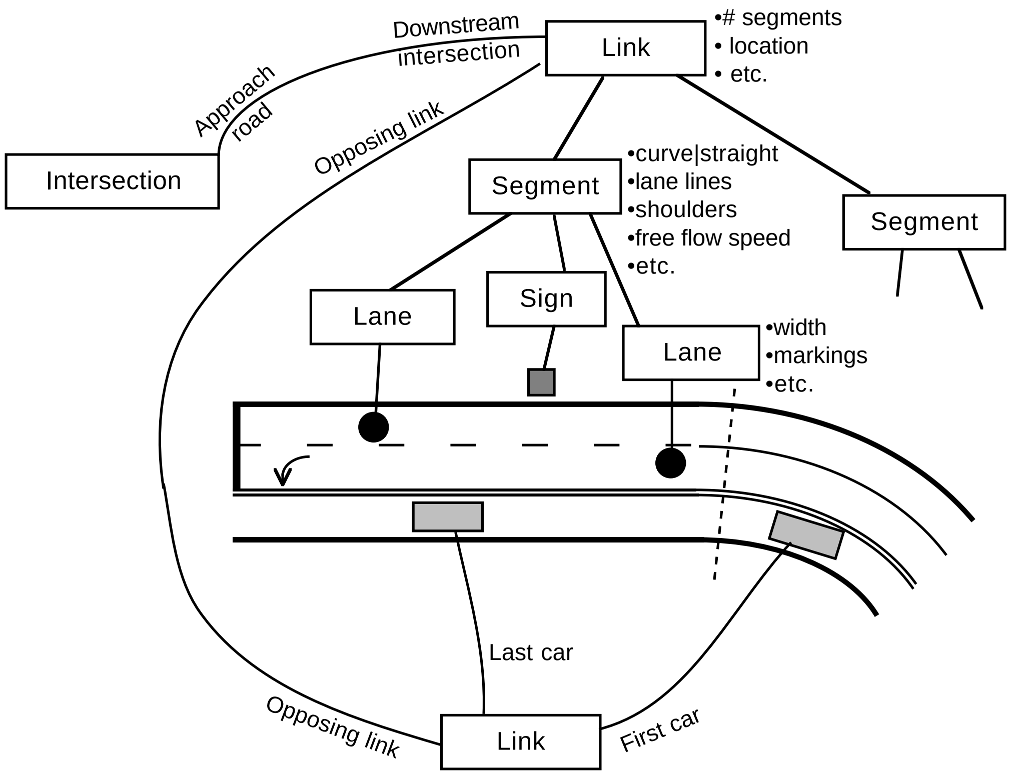


Figure 4-3: Structure of street objects.

---

the robot is really broken down or merely part of a queue that extends to the intersection.

PHAROS shortcuts these difficult interpretation problems by encoding the semantics of situations directly into the database. For example, the traffic control for each lane is provided as part of the input data for the simulation and is stored with the other characteristics of the street. Lane channelization is also provided externally and stored. Zombies determine if the car in front of them is in a queue simply by checking that car's "Queue" status; a zombie sets its own status to "enqueued" if it is close to the car in front and the car in front is already in a queue. The direct encoding of interpreted information, as well as the availability of physically unobservable state information, allows PHAROS to move zombies realistically without the complete expertise of a human driver.

Speed control. PHAROS drivers are allowed perfect perception and access to special information so that they can choose actions as realistically—i.e., with human competence—as possible. However, zombies behave unrealistically if they are allowed to control speed too well. Therefore, PHAROS incorporates a reaction delay into zombie speed control. When zombies wish to decelerate quickly, they must switch to a braking "pedal" (state), and when they wish to accelerate again they must switch to an accelerator "pedal." There is a delay of 0.8 s, on average, in switching between pedals. This imposed reaction delay, when combined with the car-following law already discussed, results in fairly realistic behavior during car following, free-flow to car following transitions, free-flow to enqueued transitions, and queue discharge.

4.3. Simulator Features

Input. PHAROS generates a street database from a data file that it reads in at the start of each simulator run. Since PHAROS can represent a lot of detail in a variety of traffic objects, the data file can be fairly complex. The PHAROS user must encode the desired street environment into symbols and numbers and type them into the file. Figure 4-4 shows an example of how one face of a traffic signal head is encoded. The terms "face" and "lens" are keywords. The first line indicates that there are 5

```
face vertical 5 L9 LEFT_MARGIN 0 0
        lens  RED_SIGNAL SC_SOLID 8          1 1 1 1 1 0 0 1
        lens AMBER_SIGNAL SC_SOLID 8         0 0 0 0 0 0 1 0
        lens GREEN_SIGNAL SC_SOLID 8         0 0 0 0 0 1 0 0
        lens AMBER_SIGNAL SC_LEFT_TURN 8     0 0 0 0 1 0 0 0
        lens GREEN_SIGNAL SC_LEFT_TURN 8     0 0 0 1 0 0 0 0
```

Figure 4-4: Example of signal head encoding in PHAROS data file.

lenses on this face of the signal head, that they are arranged vertically, and that the head is located on the left shoulder of the street named "L9." If the head were directly over a lane, the last two numbers on the first line would indicate which lane and where along the lane the head was located. The remaining lines describe each lens. The information includes the color, symbol, and diameter of the lens, and whether the lens is lit in each phase.

Encoding and typing this information is tedious and error-prone for multi-intersection networks. A future version of PHAROS should include a graphical input interface that generates the file automatically from pictorial descriptions.

Time. PHAROS has features of both a discrete time and a discrete event simulator. Some events, such as zombie creation and traffic signal changing, happen at irregular intervals; others, such as updating the positions of vehicles, occur at regular intervals. PHAROS maintains an event queue that keeps track of both types of events. The position-update events compute the vehicles' new positions exactly from their (constant) accelerations over the preceding time interval.

Display and user interaction. The output of PHAROS is an overhead view of the street environment with an animated display of the vehicles. This graphical output provides us with our primary means of evaluating the behavior of both zombies and robots. Figure 4-5 is a picture of the computer screen while PHAROS is running. The figure shows a black-and-white approximation of the computer's high-quality color display. In the lower-left corner is a window showing the entire street network schematically. A portion of the network (enclosed by a small rectangle in the schematic) is drawn at larger scale in the top-center of the screen. Cars are visible as white rectangles, with dark regions at the rear showing brake lights and directional signals. The small window to the right of the network schematic shows simulator time. A user can interactively pan and zoom around the streets to examine various areas in more detail. It is also possible to select individual zombies and trace their decision processes. The simulator can be paused or stepped one decision cycle (100 ms) at a time.

To simulate a robot, a user selects the "button" on the screen marked "create;" PHAROS then pauses and asks where the robot should be injected into the street network. The robot is rendered in a unique color. We have created various displays to study the activity of Ulysses, including the location of each mark created by the perceptual routines, a count of the number of perceptual requests, and a chart of perceptual activity in each direction.

Figure 4-5: The output display of PHAROS.

Performance. The simulator is written in C and runs on a Sun workstation under Sunview. A Sun 4/40 can drive about 30 zombies 3 times faster than real time. With Ulysses driving a robot, the simulation runs much slower than real time. The slowdown is due to the large number of messages that must be passed between the Ulysses and PHAROS and the frequent switching of control between the programs.

PHAROS has be invaluable for us in our research on robot driving. It allows us to test driving decision procedures and examine perception and information flow in Ulysses. PHAROS has been sent to several other sites doing research in driving, robotics, and artificial intelligence.

5. Conclusions

Research in autonomous vehicle technology in the past 30 years has provided vehicles that can competently perform operational driving tasks. Although computers can also plan errands and find routes, they cannot make the immediate decisions necessary to drive in traffic. Ulysses is a model for driving that describes how to perform these tactical tasks. Ulysses incorporates driving knowledge that describes how various traffic objects and situations constrain a vehicle's actions. This knowledge recognizes situations from specific relations among physical objects. Ulysses includes tactical perceptual routines for finding these objects in the world and relating them to the vehicle. The model is implemented as a computer program so that it may be developed and tested objectively; it is currently used to drive a robot in the simulated world of PHAROS.

Ulysses was designed to drive safely in the PHAROS world. Our goal was to prevent the simulated robot from having or causing accidents, and from unnecessarily constraining itself to stop. We have achieved this goal for several traffic situations, including different types of intersections, multiple lane roads, and freeway interchanges with light traffic. Future improvements to Ulysses might include new behaviors such as overtaking, improved merging, or "edging" into an intersection while waiting for a gap. A more significant improvement in driving knowledge that we have not yet examined closely is the analysis of nearby traffic to predict how other cars will move. This analysis would involve the detection of many other cars, the *abstraction* the relative motions of the cars, and the projection of the motions into the future. Eventually the model must be combined with a strategic driving program; the parameters in Ulysses would then be determined by variable strategic goals, such as time stress, pleasure in driving, worry about citations, fuel efficiency, etc. PHAROS and Ulysses could also be expanded to include more objects such as blinking signals, activated signals, bicycles, pedestrians, roads without lanes, and different vehicle types.

Rather than adding more sophisticated driving knowledge, our more immediate goal is to determine how a real robot can perform the basic tactical tasks. As Ulysses drives a robot in PHAROS, it generates many perceptual requests. Machine perception is very expensive computationally, so it is important to find ways to minimize the cost of the requests. We are studying ways to take advantage of the coherence of situations over time and the difference in criticality of constraints to eliminate unnecessary requests. In the future we will study how a resource-limited robot might choose the best perception and constraint-generation actions to perform when it cannot perform them all. We would also like to model uncertainty in sensing and interpretation explicitly, and empower Ulysses to choose actions with the maximum expected utility. Finally, a truly intelligent driving model should be able to *learn* how to improve it behavior as it gains experience driving.

Ulysses has four major strengths as a driving model: it comprises many driving tasks; it models individual drivers; it is grounded in (perceived) physical objects and operational maneuvers; and it specifies how to compute actions from perceptions. Of these characteristics, physical grounding and explicit computation are probably the most important. They allow the model to be designed and used objectively, independent of human interpretation. A computational model can help to detect problems that human designers may not see. For example, we described earlier how Ulysses looks at the backs of Stop signs facing the cars on cross streets. We did not plan to include such a perceptual action in Ulysses, but after the intersection rules were encoded it became clear that this information is necessary for determining right of way. None of the driving task descriptions we studied—including the McKnight analysis [26], and the official driver’s manual [8] and vehicle code [25] for our state—call for this action. A model such as Ulysses can predict a driver’s real information needs in any situation.

The characteristics described above make Ulysses uniquely suitable for driving an autonomous vehicle. They also give Ulysses the potential to contribute to general driving research. Research in driving seeks to answer many questions that apply equally to a robot and human driver, such as "How can a driver learn to make better decisions?" and "what information should be provided along roads to make driving easier?" We can answer these questions if we have an accurate, concrete driver model. We feel that computational driver models will prove to be invaluable tools for driving research.

References

[1] Aasman, Jans.
Implementations of Car-Driver Behaviour and Psychological Risk Models.
In J. A. Rothengatter and R. A. deBruin (editors), *Road User Behaviour: Theory and Practice*, pages 106-118. Van Gorcum, Assen, 1988.

[2] Agre, P. E. and D. Chapman.
Pengi: An Implementation of a Theory of Activity.
In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268-272. Morgan Kaufman Publishers, Los Altos, 1987.

[3] Belcher, P. L. and Catling, I.
Autoguide Electronic Route Guidance in London and the U.K..
Technical Report 89102, ISATA, June, 1989.

[4] Bender, J. G. and Fenton, R. E.
A Study of Automatic Car Following.
IEEE Transactions on Vehicular Technology VT-18:134 - 140, Nov., 1969.

[5] Cardew, K. H. F.
The Automatic Steering of Vehicles -- An Experimental System Fitted to a Citroen Car.
Technical Report RL 340, Road Research Laboratory, February, 1970.

[6] Chin, H.C.
SIMRO: A Model To Simulate Traffic at Roundabouts.
Traffic Engineering and Control 26(3):109 - 113, March, 1985.

[7] Cro, J. W. and Parker, R. H.
Automatic Headway Control -- An Automobile Vehicle Spacing System.
Technical Report 700086, Society of Automotive Engineers, January, 1970.

[8] Department of Transportation.
Pennsylvania Manual for Drivers (Publication 95)
Commonwealth of Pennsylvania, Harrisburg, PA, 1984.

[9] Dickmanns, E. and Zapp, A.
A Curvature-Based Scheme for Improving Road Vehicle Guidance by Computer Vision.
In *Mobile Robots (Vol. 727)*. SPIE, 1986.

[10] Drew, Donald.
Traffic Flow Theory and Control.
McGraw-Hill Book Company, New York, 1968.

[11] Elliott, R. J. and Lesk, M. E.
Route Finding in Street Maps by Computers and People.
In *Proceedings of AAAI 82*, pages 258 - 261. AAAI, 1982.

[12] FHWA.
Traffic Network Analysis with NETSIM: A User Guide
Federal Highway Administration, Washington, D.C., 1980.

[13] Fenton, R. and Mayhan, R.
Automated Highway Studies at the Ohio State University -- An Overview.
IEEE Transactions on Vehicular Technology 40(1):100 - 113, February, 1991.

[14] Gage, D. W. and Pletta, B. J.
Ground Vehicle Convoying.
In *Mobile Robots II (Vol. 852)*, pages 319 - 328. SPIE, 1987.

[15] Gardels, K.
Automatic Car Controls For Electronic Highways.
Technical Report GMR-276, General Motors Research Labs, Warren, MI, June, 1960.

[16] Gibson, D. R. P.
Available Computer Models for Traffic Operations Analysis.
In *The Application of Traffic Simulation Models. TRB Special Report 194*, pages 12 - 22. National Academy of Sciences, 1981.

[17] Hayes-Roth, B.
A Blackboard Architecture for Control.
Artificial Intelligence 26:251 - 321, 1985.

[18] Kawashima, H.
Two Major Program Demonstrations in Japan.
IEEE Transactions on Vehicular Technology 40(1):141 - 146, February, 1991.

[19] Kehtarnavaz, N., Lee, J. S., and Griswold, N. C.
Vision-Based Convoy Following by Recursive Filtering.
To be published .

[20] Kluge, K. and C. Thorpe.
Explicit Models for Road Following.
In *Proceedings of the IEEE Conference on Robotics and Automation*. IEEE, 1989.

[21] Kories, R., Rehfeld, N. and Zimmermann, G.
Toward Autonomous Convoy Driving: Recognizing the Starting Vehicle in Front.
In *Proceedings of the 9th International Conference on Pattern Recognition*, pages 531 - 535. IEEE, 1988.

[22] Kuan, D., Phipps, G, and Hsueh, A-C.
Autonomous Robotic Vehicle Road Following.
IEEE Transactions on PAMI 10(5):648 - 658, 1988.

[23] Laird, John E; Newell, Allen; and Rosenbloom, Paul S.
Soar: an Architecture for General Intelligence.
Artificial Intelligence 33:1 - 64, 1987.

[24] Lang, R. P. and Focitag, D. B.
Programmable Digital Vehicle Control System.
IEEE Transactions on Vehicular Technology VT-28:80 - 87, Feb, 1979.

[25] Legislative Reference Bureau.
Pennsylvania Consolidated Statutes, Title 75: Vehicles (Vehicle Code)
Commonwealth of Pennsylvania, Harrisburg, PA, 1987.

[26] McKnight, J. and B. Adams.
Driver Education and Task Analysis Volume I: Task Descriptions.
Final Report, Department of Transportation, National Highway Safety Bureau, Washington, D.C., November, 1970.

[27] Michon, J. A.
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.

[28] Oshima, R. *et al*.
Control System for Automobile Driving.
In *Proceedings of the Tokyo IFAC Symposium*, pages 347 - 357. , 1965.

[29] Pomerleau, D. A.
ALVINN: An Autonomous Land Vehicle In a Neural Network.
Technical Report CMU-CS-89-107, Carnegie Mellon University, 1989.

[30] Reece, D. A. and S. Shafer.
An Overview of the Pharos Traffic Simulator.
In J. A. Rothengatter and R. A. deBruin (editors), *Road User Behaviour: Theory and Practice*. Van Gorcum, Assen, 1988.

[31] Reid, L. D.; Graf, W. O.; and Billing, A. M.
The Validation of a Linear Driver Model.
Technical Report 245, UTIAS, March, 1980.

[32] Rillings, J. H. and Betsold, R. J.
Advanced Driver Information Systems.
IEEE Transactions on Vehicular Technology 40(1), February, 1991.

[33] T. Rothengatter and R. de Bruin (editor).
Road User Behaviour.
Van Gorcum, 1988.

[34] Shladover, S. E. *et al*.
Automatic Vehicle Control Developments in the PATH Program.
IEEE Transactions on Vehicular Technology 40(1):114 - 130, Feb, 1991.

[35] Sugie, M., Menzilcioglu, O., and Kung, H. T.
CARGuide -- On-board Computer for Automobile Route Guidance.
Technical Report CMU-CS-84-144, Carnegie Mellon University, 1984.

[36] Texas Engineering Experiment Station.
BART. Binocular Autonomous Research Team.
Research brochure, Texas A&M University.
1989

[37] Thorpe, C., Hebert, M., Kanade, T., and Shafer, S.
Vision and Navigation for the Carnegie Mellon NAVLAB.
IEEE Transactions on PAMI 10(3), 1988.

[38] Tsugawa, S., Yatabe, T., Hirose, T., and Matsumoto, S.
An Automobile with Artificial Intelligence.
In *Proceedings of the 6th IJCAI*, pages 893 - 895. IJCAI, 1979.

[39] Turk, M., Morgenthaler, D., Gremban, K., and Marra, M.
Video Road-Following for the Autonomous Land Vehicle.
In *Proceedings of the International Conference on Robotics and Automation*. IEEE, 1987.

[40] van der Molen, H.H., and Botticher, A.M.T.
Risk Models for Traffic Participants: A Concerted Effort for Theoretical Operationalizations.
Road Users and Traffic Safety.
In J. A. Rothengatter and R. A. de Bruin,
Van Gorcum, Assen/Maastricht, The Netherlands, 1987, pages 61-82.

[41] von Tomkewitsch, R.
Dynamic Route Guidance and Interactive Transport Management with ALI-Scout.
IEEE Transactions on Vehicular Technology 40(1):45 - 50, February, 1991.

[42] Waxman, Allen; LeMoigne, Jacqueline J.; Davis, Larry S.; Srinivasan, Babu; Kushner, Todd R.; Liang, Eli and Siddalingaish, Tharakesh.
A Visual Navigation System for Autonomous Land Vehicles.
IEEE Journal of Robotics and Automation RA-3(2), April, 1987.

[43] Wong, Shui-Ying.
TRAF-NETSIM: How It Works, What It Does.
ITE Journal 60(4):22 - 27, April, 1990.

Table of Contents

List of Figures

List of Tables