Appears in: Proceedings of IEEE Intelligent Vehicles 1995

SHIVA:
Simulated Highways for Intelligent Vehicle Algorithms

Rahul Sukthankar, Dean Pomerleau and Charles Thorpe
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3891
Internet: {rahuls|pomerlea|cet}@ri.cmu.edu

Abstract

SHIVA (Simulated Highways for Intelligent Vehicle Algorithms) addresses deficiencies present in existing microscopic traffic simulators with:

1. Realistic sensor modeling.
2. Support for communicating vehicles.
3. A variety of driver models.
4. Efficient integration with real robots.

SHIVA runs on Sun Sparcstations (under X) and SGI workstations (using Inventor). Class hierarchies for vehicles, sensors and displays allow users to develop algorithms for specific vehicle/sensor configurations and to debug them with graphical viewing tools. Simulated scenarios can be interactively displayed online in 3-D. Off-line execution is also available if only statistics (such as throughput) are desired.

1. Introduction

Intelligent vehicles must make tactical-level decisions to safely drive in mixed traffic environments. SAPIENT [9], a reasoning system under development at Carnegie Mellon, will integrate strategic-level goals and sensor-based constraints in real-time. Since repeatable testing of different algorithms in rare and potentially dangerous traffic scenarios is necessary before SAPIENT can drive on real roads, we have developed a custom simulator for this task. SHIVA (Simulated Highways for Intelligent Vehicle Algorithms) mirrors many aspects of the Carnegie Mellon Navlab [10] environment, enabling algorithms developed in simulation to be implemented on the robot with little modification. Accurate sensor modeling (with noise and occlusion) encourages developers to create algorithms that will work on real robots. Asynchronous vehicle-to-vehicle communication and driver models (with reaction delays) are also available.

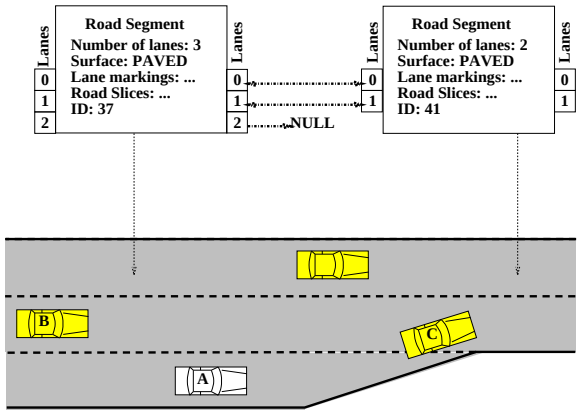

Figure 1: Road Segment Architecture.

2. System Architecture

SHIVA is implemented in C++ using the object-oriented paradigm. The major classes are described below:

2.1. Roads

Highways are organized as groups of connected RoadSegments, where each segment is a stretch of road with arbitrary shape, but a fixed number of lanes (See Figure 1). Segments connect to other segments on a lane by lane basis, allowing us to model any given highway topology. Each segment also contains information relevant to that stretch of road (e.g. lane marking information, speed limits and surface type). Special types of road segments (e.g. straight sections) are derived from the base class RoadSegment.

A RoadSegment's geometry is contained in its list of RoadSlices. Each "slice" (See Figure 2) is a line cutting across the lanes at right angles to the road and is represented by an *origin* point (centered in the left-most lane) and an *offset* vector pointing across the road, with magnitude equal to the local lane width. The spacing between adjacent RoadSlices

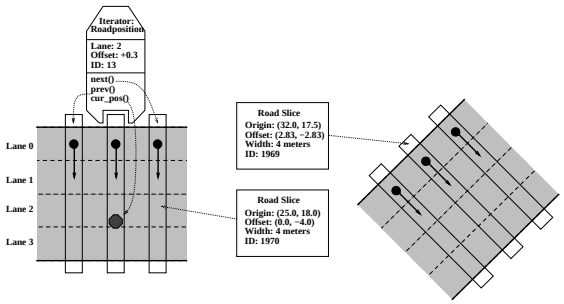

Figure 2: Road Slices and Iterators.

may vary according to the expected traffic speed and local curvature of the road. It is important to note that although the road is represented discretely, vehicles in SHIVA may have arbitrary poses — in fact, under the control of a bad road tracking algorithm they may leave the road entirely. The road coordinates of a point are represented by its closest slice and lateral displacement. Thus the global (Cartesian) position of a given road point is given by:

$$P_G = \text{origin}_s + l \times \text{offset}_s \quad (1)$$

where s is the slice associated with the point and l is the lateral displacement (in lane units). Relative road coordinates are used for reasoning once cars observed by the on-board sensors are mapped into appropriate lanes.

The `RoadPosition` class is an iterator that provides a transparent interface between the road representation and other SHIVA objects such as sensors. A `RoadPosition` instance, once placed at a given slice and lateral displacement, can advance or retreat as needed – crossing segment boundaries seamlessly. This lane-tracking feature is used by road-following sensors (See Section 2.2.2) to provide steering information for simulated vehicles. The notion of a real-numbered lane displacement (rather than an integral "lane number") allows intelligent lane trackers to function during lane changes and is a feature lacking in many existing road simulators [2].

2.2. Vehicles

The variety of vehicles on the highway is best represented by a hierarchy. In addition to the obvious physical attributes (including pose and velocity), all SHIVA vehicles contain the following subsystems:

- Controller: (See Section 2.2.1)
- Sensors (See Section 2.2.2)
- Driver model: (See Section 2.2.3)

Additionally, automated vehicles may possess the ability to communicate asynchronously with other similarly equipped vehicles (See Section 2.2.4).

The subsystems described below each have their own class hierarchies, providing an open-ended structure which can be extended to suit the simulation needs. We also expect to be able to replace a given simulated subsystem with its real counterpart with minimal modifications to other code.

2.2.1. Controller

SHIVA's simulated controllers have the same interface as the real controller in the Navlab II project. This ensures that the simulated vehicles only request actions that are feasible on a real vehicle, and also allow control algorithms developed in simulation to run on the Navlab vehicles without modification. The following commands are recognized:

1. Steering: expressed as a curvature (analogous to steering wheel angle).
2. Velocity: given as a target velocity, to be maintained once reached.
3. Query: returns current steering and/or velocity.

A hierarchy of controllers (mirroring the vehicle hierarchy) allows specific vehicle characteristics to be expressed cleanly. In the current implementation the controllers check actions against vehicle limits (acceleration constraints etc) but do not model vehicle dynamics.

2.2.2. Sensors

Since most existing simulators model sensors at a very abstract level, their vehicles are able to drive with unrealistically complete knowledge about the environment. For example, Niehaus[5] assumes that cars can directly (and perfectly) sense other vehicles' accelerations. Other simulators assume transparent vehicles [7] or curve-free highways [3, 5]. Although simulators are always forced to trade realism for performance, it is important to ensure that the algorithms developed in simulation can successfully operate with real inputs.

SHIVA sensor characteristics include both sensor types (e.g. lane trackers, car detectors, GPS) and level of detail (e.g. abstract, realistic, noisy). The following sections deal briefly with some of the simulated sensors.

Lane Trackers

Lane tracking is a challenging robotics problem [6] but one that is largely taken for granted in those highway simulations where lane tracking and lane

occupancy are equated. Such "slot-car" simulations cannot capture behaviors such as corner cutting which affect decision making in real life, and also view lane changes as instantaneous events. Simulated vehicles communicate with SHIVA's lane tracking sensors in the same way that the Navlab communicates with the ALVINN [6] road follower: the sensors return information about the local road curvature through pure-pursuit points.

Positioning

SHIVA simulates two types of positioning sensors: GPS and dead-reckoning. Both sensors return the vehicle's position in global coordinates, but differ in their noise characteristics: GPS returns readings that are corrupted by Gaussian noise (invariant over time) while dead-reckoning returns measurements that become more inaccurate with distance traveled (differential errors accumulate with vehicle motion).

Car Detection

SHIVA models both abstract and realistic car detection sensors. The abstract sensors automatically perform obstacle-to-lane mapping and return position, velocity and vehicle-id of a vehicle within the sensor's field of view. An option to corrupt these measurements by Gaussian noise is available — but since the noise is injected only in the last step, the output is not equivalent to that obtained from a real sensor model (where error rates for ρ and θ can be specified independently).

The realistic sensors are similar to scanning 1-D lighthouse range sensors — returning an array of distances to objects in the field of view. Associated with each range sensor is a covariance matrix containing error rates for θ (the scan angle) and ρ, the measured distance to the object. Sensors can be mounted in different places on the vehicle and overlapping fields of view can be used to reduce errors if needed. Range readings can be segmented into obstacles to give approximate size and position relative to the sensor. Mapping these obstacles onto an internal local lane model enables the vehicle to sense whether other vehicles are in the same or adjacent lanes. Velocities are not directly returned — they must be extracted by tracking obstacles from frame to frame and measuring the changes in their position over time. Since the range readings are corrupted by noise, filters (e.g. Kalman) for position and velocity may be needed. Higher order derivatives (acceleration etc) cannot be reliably extracted from noisy range measurements. Because this processing is computationally significant when many sensors are involved, most scenarios involve small numbers of realistic range sensors (on the vehicles of interest) supplemented by large numbers of abstract sensors.

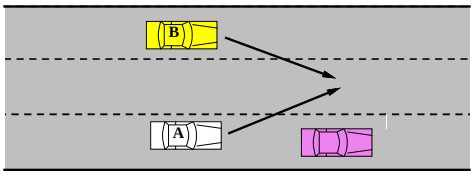

Figure 3: Pinch Scenario

2.2.3. Driver Models

In SHIVA, the "driver model" refers to the vehicle's decision making algorithm. With appropriate algorithms, automated and manually driven vehicles can both be modeled. However all driver models must operate on incomplete (possibly noisy) information and execute in real time. Although there are no intrinsic differences between automated and manual vehicles, the latter can be characterized as possessing longer reaction delays and incomplete sensing fields (blind-spots). Automated vehicles may also be able to communicate (See Section 2.2.4) with other similarly equipped vehicles.

In initial versions of SHIVA, driver models were independent of sensor configuration, enabling runtime changes of drivers and sensors. Our experience showed that the flexibility that is gained through such freedom is outweighed by the disadvantages. The current version of SHIVA tightly couples sensor configurations with driver models (See Section 2.2.5).

2.2.4. Communication

Communication between similarly equipped intelligent vehicles is made possible through an asynchronous message passing system. Each vehicle has an incoming message buffer and a transmitter which can send to vehicle(s) within range. This allows automated vehicles to send *intentions* in advance of acting, reducing the risks of collision. For example, the classic "pinch maneuver" (See Figure 3) is averted if Car A broadcasts its intention of shifting left before beginning the lane change maneuver. Car B can decide not to change right given this information. Without communication, both cars could decide to change into the same lane simultaneously (and at least one would need to abort the lane change to avoid collisions).

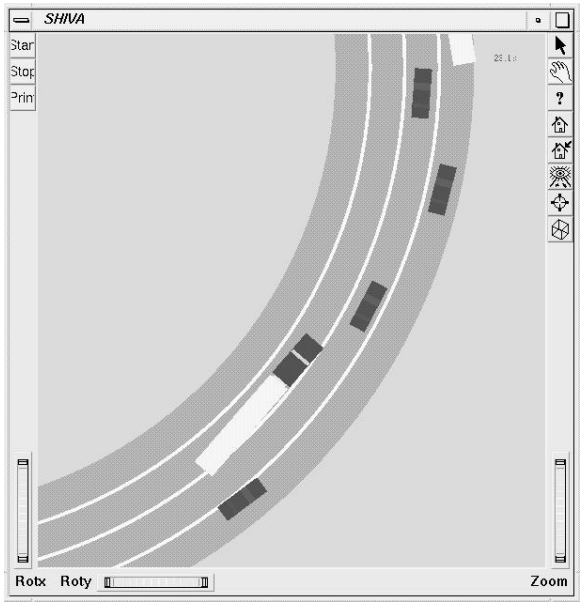

Figure 4: Overhead view

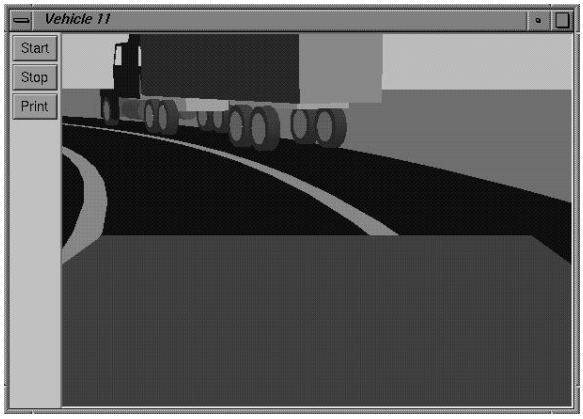

Figure 5: Driver's eye view

2.2.5. Vehicle Types

Vehicle *types* are derived classes with compatible configurations of subsystems. Rather than forcing each subsystem to rely only on the methods declared in the base classes, individual subsystems can take advantage of the specific features in the other subsystems on the same vehicle. Thus vehicles with communication should have driver models which expect to send and receive messages, while cars with human drivers will have blindspots and should reason accordingly. SHIVA guarantees that driver models and vehicle configurations are compatible through `C++`'s type-checking mechanism.

3. User Interface

SHIVA's design decouples the simulation and the user interface without resorting to offline animation (in contrast to SmartPATH [3]). The display types are all derived from an abstract base class `World` which contains lists of `RoadSegments`, `Vehicles` and simulation parameters. Different display implementations can show the current state of the simulation as desired (bird's eye view, road side view etc). SHIVA currently supports three interfaces: a `tty` interface, an `X` based interface and an SGI interface. SHIVA's SGI interface provides a number of different displays including an interactive overhead view (See Figure 4), a driver's eye view (See Figure 5) and a vehicle tracking view (See Figure 6). Users may examine selected vehicles' attributes using configurable `InfoBoxes`.

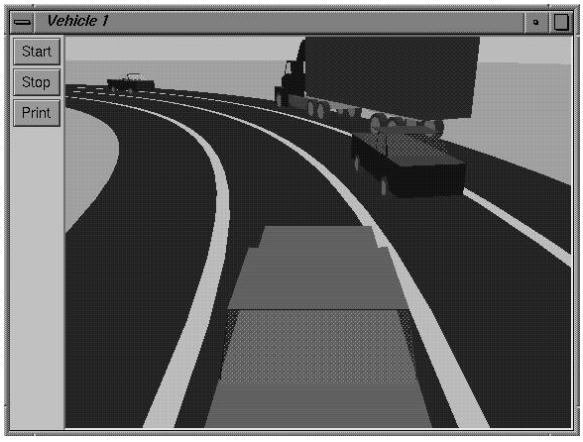

Figure 6: Vehicle tracking view

4. Comparisons with Existing Simulators

While microscopic ground vehicle simulators have been in use for over thirty years, no existing simulator provides all of the features needed for developing intelligent vehicle systems. Some (e.g. NETSIM [11]) lack detailed models of lane changing; others use very simplified road models (e.g. SmartPATH [3]) or model sensors at a very abstract level (e.g. Pharos [8]). The University of Iowa simulator [1] has a very realistic user interface but is designed primarily for observing the actions of a human driver in scripted scenarios.

Many of SHIVA's advantages are seen clearly in tactical-level driving situations. Consider the common scenario of merging into traffic (See Figure 1). Our vehicle (labeled A) must find a gap in traffic, accelerate to the correct speed and change lanes. For a robot with existing sensor technology, this problem is challenging. However in a simulator that gives Car A direct access to the positions, velocities and accelerations of other vehicles, the task is relatively straightforward. Algorithms developed in such simulations tend to rely on measurements that are unavailable in real implementations. SHIVA enables designers to face these difficulties in simulation by providing (at the most detailed level) realistic rangefinder images which require significant processing before attributes such as position or velocity can be determined. Algorithms [4] which require perfect and complete measurements cannot operate in this environment. We have implemented a robust gapfinder in simulation and successfully tested it on image sequences obtained from Navlab II's rangefinder. We hope to demonstrate a merge scenario on the Navlab once longer range sensors with faster scan rates are available.

If Figure 1 were a snapshot of an on-ramp to an intelligent highway, Car A might communicate with Cars B or C before merging. Simulators such as SmartPATH [3] implement communication between vehicles in dedicated automated lanes but have no provisions for situations involving both automated and manual traffic. SHIVA does not advocate a specific AHS concept and allows users to simulate any given mix of communicating and non-communicating vehicles. We are interested in investigating the effects of optional communication (e.g. Car A may request Car B to change left to open a gap) in mixed traffic environments.

Simulating different *types* of intelligent vehicles (with different sensors, different algorithms etc) is generally impossible in most simulators since vehicles are hard-coded. SHIVA allows users to simulate any number of vehicle configurations simultaneously: for example, Car A in the figure may be a vehicle with communication, Car B may be a human driver with intelligent cruise control, and Car C a human operated vehicle without a rear view mirror. Thus if Car A, after making unheeded requests for a suitable gap, were to aggressively merge into the space between B and C, Car B would slow down, while Car C would not react to A at all.

Lane changing implementations in simulation generally ignore the role of the lane tracker. Since changing lanes on curved roads is non-trivial, many simulators model either discrete lane changes (e.g. Niehaus [5]) or straight roads only (e.g. SmartPATH [3]). SHIVA recognizes the road following algorithm as a sensor and implements lane changing as either:

- a combination of two lane tracker outputs (assuming a passive road follower which tracks only 1 lane).
- a direct output from an intelligent lane tracker (that takes lateral offsets as inputs).

SHIVA's modules for lane tracking and lane changing correspond closely to their real counterparts both in behavior and in interface.

5. Conclusion

Simulated testbeds are essential in the development of intelligent highway systems for economic and safety reasons. SHIVA addresses deficiencies in existing simulators by adding realism to sensors and ensuring that actions in the simulated environment accurately map to vehicle maneuvers. SHIVA has been used to develop a number of vehicle control algorithms and will be used to simulate tactical level scenarios for SAPIENT. SHIVA's close ties to the NAVLAB project make it an ideal development platform for algorithms that need to be ported from simulation to real-life with minimal modification.

Future versions of SHIVA will allow users to edit vehicle and sensor configurations using a graphical user interface. Similar tools for roadway design and scenario specification are also planned. Extensions to the interactive interface will enable human operators to control simulated vehicles and interactively create simple scenarios for the computer controlled vehicles.

Mixed-traffic scenarios with communicating vehicles will be a primary focus of our investigations. We will use SHIVA to examine the impact of loosely-coupled cooperation among intelligent vehicles on highway throughput by varying the proportion of communicating cars in different scenarios.

6. Acknowledgements

The authors wish to thank John Hancock, who developed SHIVA's SGI interface and Alonzo Kelly who wrote the `X` interface's display primitives.

References

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

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

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

[4] Simin Nadjm-Tehrani. Analysis of the overtaking scenario: Specification of an autonomous car and a driver support system. Technical Report LiTH-IDA-R-91-07, Department of Computer and Information Science, Linkoping University, Linkoping, Sweden, 1991.

[5] A. Niehaus and R. F. Stengel. Probability-based decision making for automated highway driving. *IEEE Transactions on Vehicular Technology*, 43(3):626–634, August 1994.

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

[7] Douglas A. Reece. *Selective Perception for Robot Driving*. PhD thesis, Carnegie Mellon University, May 1992.

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

[9] Rahul Sukthankar. Situational awareness for driving in traffic. Thesis Proposal, October 1994.

[10] Charles E. Thorpe, Martial Hebert, Takeo Kanade, and Steven A. Shafer. Vision and navigation for the Carnegie Mellon NAVLAB. *IEEE Transactions on PAMI*, 10(3), 1988.

[11] Shui-Ying Wong. TRAF-NETSIM: How it works, what it does. *ITE Journal*, 60(4):22–27, April 1990.