Presented at:
SPIE International Symposium on Intelligent Systems & Advanced Manufacturing
Technical Conference on Intelligent Transportation Systems
13-17 October 1997, Pittsburgh, PA, USA

Functional sensor modeling for Automated Highway Systems simulations

Cem Ünsal[1], Rahul Sukthankar[1,2], and Chuck Thorpe[1]

[1]Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213-3890
[2]Justsystem Pittsburgh Research Center, 4616 Henry Street, Pittsburgh, PA 15213

ABSTRACT

Sensor technology plays a critical role in the operation of the Automated Highway System (AHS). The proposed concepts depend on a variety of sensors for positioning, lane-tracking, range and vehicle proximity. Since large subsystems of the AHS will be designed and evaluated in simulation before deployment, it is important that simulators make realistic sensor assumptions. Unfortunately, the current physical sensor models are inadequate for this task since they require detailed world state information that is unavailable in a simulated environment.

In this paper, we present an open-ended, functional sensor hierarchy, incorporating geometric models and abstract noise characteristics, which can be used directly with current AHS tools. These models capture the aspects of sensing technology that are important to AHS concept design such as occlusion, and field of view restrictions, while ignoring physical-level details such as electromagnetic sensor reflections. Since the functional sensor models operate at the same level of granularity as the simulation platform, complete integration is assured. The hierarchy classifies sensors into functional groups. The models at a particular level incorporate characteristics that are common to all sensors in its subgroups. For example, range sensors have a parameter corresponding to a maximum effective range, while lane-trackers include information pertaining to lateral accuracy.

Keywords: sensor models, vehicle simulations, automated highway systems

1. INTRODUCTION

The Automated Highway System (AHS)[1], a solution prescribed by the USDOT to the problem of traffic congestion and accidents is an ambitious project incorporating new advances in sensor design, vehicle control, and robotics. Simulation has emerged as an important tool for studying and evaluation the merits of different choices in this design space for several reasons:

- The complex nature of phenomena such as congestion, accidents, and driver behavior eludes most analytic approaches.
- Prototyping systems in real traffic is potentially dangerous and expensive. Furthermore, many of the important components (such as reliable vehicle tracking systems) are still under development.
- Simulations enable researchers to explore rare and critical situations safely and repeatably. This is particularly valuable during the development of intelligent vehicle reasoning systems.

Tools that explicitly model the behavior of individual vehicles are known as *microsimulators*[2,3,4]. Since these vehicles must realistically respond to the conditions in the simulated environment, they should be able to sense their surroundings in a realistic manner. The sensor models discussed in this paper act as a filter between the internal representation of the microsimulator and the driver model of the simulated vehicle (See Figure 1).

Further author information:
CÜ: E-mail: unsal@ri.cmu.edu, WWW: http://www.cs.cmu.edu/~unsal, Telephone: (412) 268-5594, (412) 268-5571 (fax).
RS: E-mail: rahuls@cs.cmu.edu, WWW: http://www.cs.cmu.edu/~rahuls, Telephone: (412) 683-8046, (412) 683-4175 (fax).
CT: E-mail: cet@ri.cmu.edu, WWW: http://www.cs.cmu.edu/~cet, Telephone: (412) 268-3612, (412) 268-5571 (fax).

Most traffic simulators ignore the problem of sensor modeling entirely. The driver models which control the behavior of simulated vehicles are typically expected to traverse the internal data structures of the simulator and extract attributes about the environment such as the speed of the preceding vehicle or the distance to the next exit. While such systems may be straightforward to implement, it is clear that the vehicles in these simulators are operating under radically different perceptual assumptions from real cars on the highway. For example, these "omniscient" simulated vehicles are equipped with error-free sensors that are unaffected by problems of occlusion or finite range, and can easily react to events before they would be "realistically" perceivable. Some simulators even permit vehicles to directly access the internal states of other vehicles' driver models allowing one vehicle to react to the intentions of another vehicle before any actions have been taken. It is therefore unsurprising that, given the shortcomings of such microscopic models, optimistic results from simulations are often met with skepticism.

On the other hand, creating a simulator that perfectly models every aspect of the problem is an impossible task. Even if such a complex model were feasible, issues of efficiency and development cost could render such an effort impractical. The alternative is to build several models at appropriate levels of abstraction, each of which focuses on a selected aspect of the problem. Most aspects of the world are only coarsely modeled (or left unmodeled) in any particular such simulator. For example, a simulator that focuses on determining optimal platooning distance would be unlikely to model weather effects—even though icy roads can drastically affect the braking ability of a vehicle, and therefore impact its platooning capability. Similarly, a detailed sensor model may depend on characteristics that are unmodeled in the simulator. For instance, a radar model[5] requires information about the material and pose of the target patch, yet we know of no traffic microsimulator that models vehicles at this level of detail.

Sensor models for AHS simulation must therefore satisfy two constraints. First, they must make sufficiently realistic perceptual assumptions, from the perspective of the driver model. Second, they must only rely on inputs that are available from the typical microsimulation environment. This paper discusses our solution: functional sensor models. The following section introduces our approach to the problem. Design issues and sensor implementation are discussed in Section 3. Section 4 presents several simulation results, followed by our concluding remarks in Section 5.

2. APPROACH

Intelligent vehicles in the Automated Highway System may be equipped with a variety of sensors, each with its own particular characteristics (See Table 1). The behavior of each of these sensors has been extensively studied[6,7,8,9]. While implementing these models in an AHS simulator is not difficult, acquiring many of the inputs and parameters required by these models (from a simulated environment) is impossible. Furthermore, since many of these sensor models are computationally intensive, simultaneously simulating several of these models for each of the vehicles in a AHS micro-simulation could prove to be impractical.

The only choice is to create lower-fidelity sensor models, suitable for this application. However, to do this, one must first understand the relationship between the various sensors, as applied to the intelligent vehicle domain. Table 1 shows a sensor hierarchy that classifies sensors by function (from an intelligent vehicle's perspective). While this sensor hierarchy is also generally applicable to mobile robots, specific environments such as underwater or space applications, may demand a different classification. As suggested by this hierarchy, sensors can be classified into the following classes:

- Long-range sensors that measure the distance to an obstacle or vehicle in the environment: These include both active sensors such as laser and radar, as well as passive ones such as stereo vision.

- Short-range sensors that detect objects in the immediate vicinity of the ego-vehicle: Again, these include both active sensors such as sonar, and passive ones such as capaciflexor. These sensors are typically used to check blind spots or parking spaces before attempting lateral maneuvers.

- Sensors that measure the relative velocity of a sensed vehicle: Some range sensors (such as Doppler radar) do this automatically. For other sensors, this may involve sophisticated processing (such as optical flow), or require differential range measurements.

- Positioning sensors that enable estimation of current ego-vehicle location: These include on-board systems such as encoders, in addition to the Global Positioning System (GPS).

- Sensors that measure the ego-vehicle's current velocity: This may be done through odometry, by sensing relative motion with the ground using various techniques, or by using GPS.

- Lateral lane-tracking systems: This includes direct measurement of lane deviation through infrastructure-assisted methods such as magnetic nails as well as sophisticated software for vision-based lane-trackers.

- Clocks: Since modern clocks do not lose an appreciable amount of time over the duration of an traffic scenario, modeling a time sensor for AHS simulations is trivial. This is certainly not true for all applications.

- Environment sensors: This category includes sensors that detect dangerous road conditions such as ice or wet pavement, or weather conditions such as fog and snow (which can adversely affect other sensors). Since these phenomena are typically unmodeled in microsimulations, environment sensor models do little more than reporting the initial conditions as specified by the user.

Note that some entries, such as vision-based lane-trackers, may not traditionally be classified as sensors (since the actual "sensor" is just a video camera); however, from the perspective of an intelligent vehicle's cognition system, a vision-based lane-tracker is an abstract sensor that returns some information about the ego-vehicle's current pose relative to the road. Similarly, much of the perceptual processing involved with certain sensor (such as tracking objects over time) can be considered to be part of the sensor model. Depending on the fidelity of the underlying microsimulation and the needs of the driver models, the list above could be expanded to include sensors to measure engine-state, tire pressure, etc.

Each of the classes described above is represented in the microsimulator by a *functional sensor model*. The functional sensor model contains three important parts. First, it implements algorithms that access the microsimulator's internal state to create the appropriate outputs (e.g., a range sensor should return the distance to the appropriate target). Second, it is responsible for realistically corrupting the sensor measurements. Parameters such as maximum range or field of view are represented explicitly when the microsimulator's fidelity can support them. In other cases, the characteristics are captured in a more abstract manner (e.g., expected accuracy of lane-tracking). Third, the sensor model should document the unmodeled features of the sensor that may be relevant to the AHS domain. For instance, GPS sensors do not work in tunnels because the satellites are occluded; the GPS sensor model (which is an instance of the Positioning Sensor class) should note this fact even though the AHS microsimulator models neither GPS satellites nor tunnels. This reduces the likelihood that an AHS system will rely on assumptions that may become invalid in real situations.

Table 1. Sensor Hierarchy and Its Possible Implementations

Classes	Sensors	Characteristics, Problems, Implementation
Long Range	- Laser Ranging - Radar - Ultrasonic - Stereo Vision	- All except the visual sensor are implemented at three different levels of detail. The definitions in Section 3 cannot be used for vision. Visual sensor implementations are very complex.
Short Range (Proximity)	- Ultrasonic - Infrared	- Similar to long range sensors; time delays for arrays. - Single ray definition
Object Velocity	- Doppler radar - Optical flow - Differential Range Radar	- Interference, incidence angle, filtering problems - Difficult to implement - Similar to range sensor implementation
Positioning	- GPS, differential GPS - Dead reckoning (Encoders) - Triangulation (Beacons) - Inertial Navigation Systems	- Gaussian noise characteristics - Error accumulation must be modeled - Simple noise injection sufficient - Error accumulation due to integration must be modeled
Personal Velocity	- Odometry - Ground Speed Radar - Carrie Phased GPS - Optical flow	- Slippage, tire condition must be modeled - Interference, incidence angle, filtering problems - Same as GPS - Represent as abstract lane-tracker
Lateral Lane-Tracking	- Magnetic Nails - Magnetic bands - Vision	- Constant distance sampling - Constant time sampling - Difficult to implement
Time & Environment	- Clock - Ice, rain detectors	- A single global clock is sufficient - Report initial conditions in simulator.

3. DESIGN ISSUES AND IMPLEMENTATION

The computer implementation of a functional sensor is a module that processes information from the simulation environment ("real world") to create inputs suitable for the cognition system ("perceived world"). Additionally, the functional sensors are used to "corrupt" the actual state of the world, as represented in the microsimulator internals, into realistic sensor readings, useful for evaluating control methodologies.

If the simulation assumes perfect knowledge of the world, the control loop for an automated vehicle model is similar to the one given in Figure 1, without the additional sensor block. A vehicle model reacts to its input and affects the simulated environment in return. In general terms, the changes in the environment are compared to the desired conditions/parameters in the cognition model (e.g., human driver or intelligent vehicle controller model), and a control input to the vehicle is created. The sensor models we add are modules that "filter" the perfect knowledge, thus creating realistic measurement values to be used in design and analysis of vehicle controllers and/or AHS scenarios.

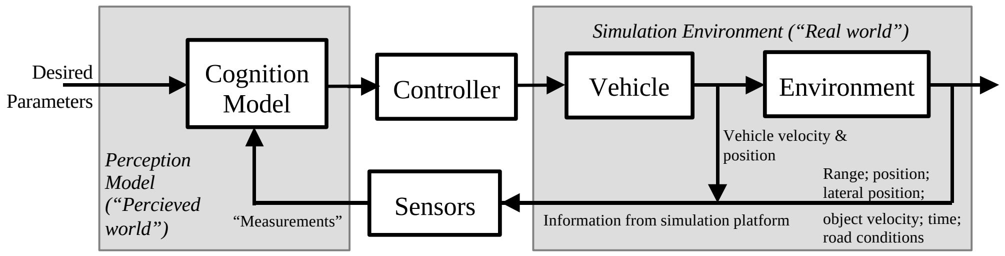

Desired Parameters
Cognition Model
Controller
Vehicle
Environment
Simulation Environment ("Real world")
Vehicle velocity & position
Range; position; lateral position;
object velocity; time; road conditions
Information from simulation platform
Sensors
"Measurements"
Perception Model ("Percieved world")

Figure 1. The control loop for AHS simulations with functional sensors.

The functional sensor models discussed in this paper do not include detailed sensors for measuring ego-vehicle status, such as engine or suspension state. Such sensor models are usually developed in conjunction with the low-level controllers. Therefore, the focus of this paper is on sensors that interact with other objects in the simulation environment. As mentioned above, the functional sensors are designed at the same level of granularity as the simulation platform to ensure complete integration. Consider a generic range sensor used for vehicle and/or obstacle detection. The definition of the sensor may include just a few parameters describing a very simple functional model returning only the distance to the object in range. Alternatively, multiple parameters can be used in order to describe a complicated model capable of capturing the shape of the sensed objects. Intermediate levels of detail are also possible. Of course, the more capable the model, the more intensive the computation. Again, several real-life parameters that are not compatible with the simulation platform, e.g. the reflection characteristics of the vehicle surfaces are not modeled. Inclusion of such characteristics is well beyond what an AHS microsimulator can implement.

Let us consider several functional models for a two-dimensional generic range sensor. We have defined several levels of detail for a sensor that can return the distance and/or azimuth angle to the vehicle(s) in range. Figure 2 illustrates the four levels of detail we considered for implementation of range sensors. The simplest model for range sensing uses the position information on all the vehicles to compute the distance to the closest vehicle (Figure 2a). This approach treats the vehicles as particles, therefore the error is sensed distance can be large for shorter ranges. Furthermore, the exact position of the vehicle with respect to the sensor/ego-vehicle is not known. The implementation however is very simple, and the calculations are minimal. If the sizes of the vehicles, and their relative orientations are known for a specific scenario (e.g., a simulation for platooning on a straight road), necessary adjustments in sensed values can be made.

A natural extension of the first sensor model is the addition of sensor position and orientation relative to the vehicle center of gravity to the sensor parameters (Figure 2b). This enables a range sensor to return the orientation of the closest vehicle in range as well as its distance. The computational burden for this sensor is again minimal. This approach also provides a horizontal field of view definition for the sensor by simply filtering the azimuth readings beyond a predefined value.

The next two levels of detail for the range sensor model are designed to provide additional information about the orientation of the sensed objects as well as more accurate information about the position and the range of the sensed vehicle.

The model which employs "pseudo-vertex" definitions enables the user to define points on the rectangles describing the vehicles, and uses these to evaluate the range and azimuth angle (Figure 2c). This approach is useful in relatively long range sensor implementations; the returned azimuth angle may be drastically different than the actual value in close range as illustrated in Section 4. The sensor module created for pseudo-vertices approach can return the range and azimuth angle of all the defined points that are in sensor range, thus providing additional information about the orientation of the vehicle/obstacle surface. The price for this additional information is of course the complexity of computation: the length and width of all the vehicles, as well their current orientation, must be fed to the sensors, since they are required for calculating the positions of multiple pseudo-vertex points. An additional computation loop for all these points are needed for each vehicle in range.

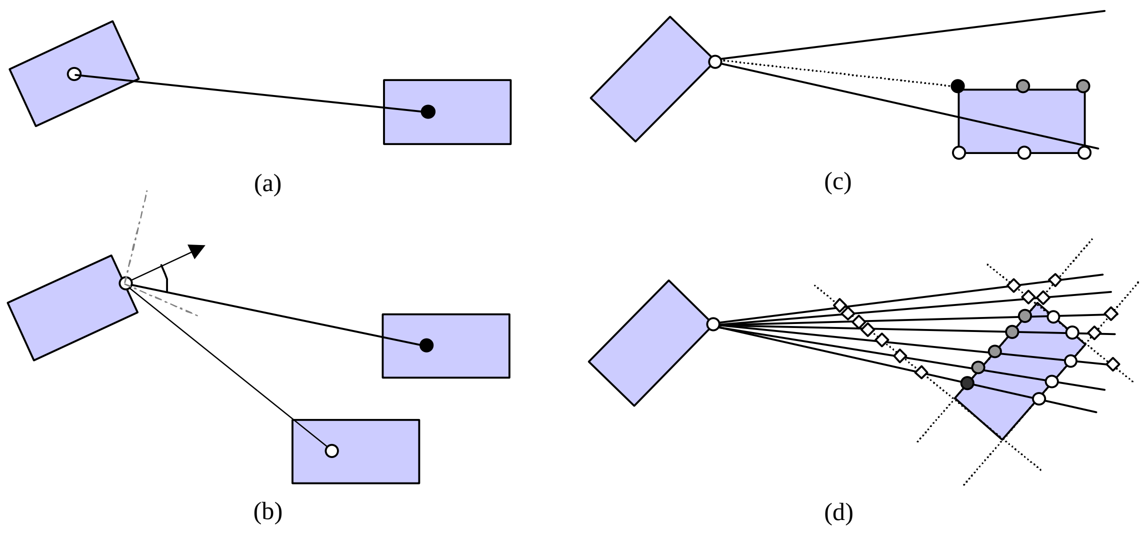


Figure 2. Four different levels of detail for generic range sensor: (a) point vehicles, (b) addition of sensor position and orientation as well as azimuth angle, (c) pseudo-vertex calculations for vehicle orientation, (d) ray definitions.

The most complex range sensor definition includes ray definitions where the number of scanning rays are parameterized in addition to previous definitions of horizontal field of view, sensor position and orientation. In this model, the functional model computes the intersection of the defined rays (for which the maximum range and the azimuth angle are known) and the lines defining the vehicles. For each ray of the sensor and for each line of all vehicles in range, the intersections are computed. Intersection points outside of the segments that constitute the vehicle (indicated by "diamonds" in Figure 2d) are discarded as well as those that are in the opposing side of the vehicle (white circles in Figure 2d). The closest intersection point is returned as the range; it is also possible to return a vector of sensed values (a 1-D range scan) for more complex driver models.

All sensor models discussed here use periodical updates where the sampling frequency is a user-defined parameter that can be as small as the simulation time step. Gaussian measurement noise is injected into the range and azimuth readings; the mean and variance of the noise are user-defined. Furthermore, the models are designed so that different modes of operation may be defined. For example, the noise characteristics of a range sensor may depend on the weather conditions on the road segment that the vehicle travels, or the user may introduce operation modes that emulate faults in the system. It is also possible to represent sensor failure modes abstractly by failure rates, ghost targets, and false positives.

As seen from the definitions above, different types of functional sensor models at different levels of detail bring complexity to the computational task due to the added information. The last two models can provide a very clear "picture" of the obstacle/vehicle in range with multiple pseudo-vertex points or a large number of ray, but the computational burden will increase with the added number of vehicles.

The implementations of the range sensors discussed above have many common points except the computational subroutines and the level of complexity. The information source for vehicle positions and orientation, noise characteristics, main sensor modules and preprocessing of the data (such as filtering the vehicles in sensor range from the set of all vehicles) are common to all range sensors. The general structure of a range sensor is illustrated in Figure 3. The environment conditions are used to evaluate noise characteristics for a specific sensor, although the initial noise model is defined at the setup phase with other sensor characteristics such as level of detail and pertaining parameters. The information about the vehicles is preprocessed in order to extract necessary information, such as current sensor and vehicle positions and orientations, and the set of vehicles in the field of view. The preprocessing is followed by the computation of the desired sensor outputs. Fault modes of the sensors are defined as discrete states, and the parameters characterizing the sensor capabilities are updated while transitioning between these states.

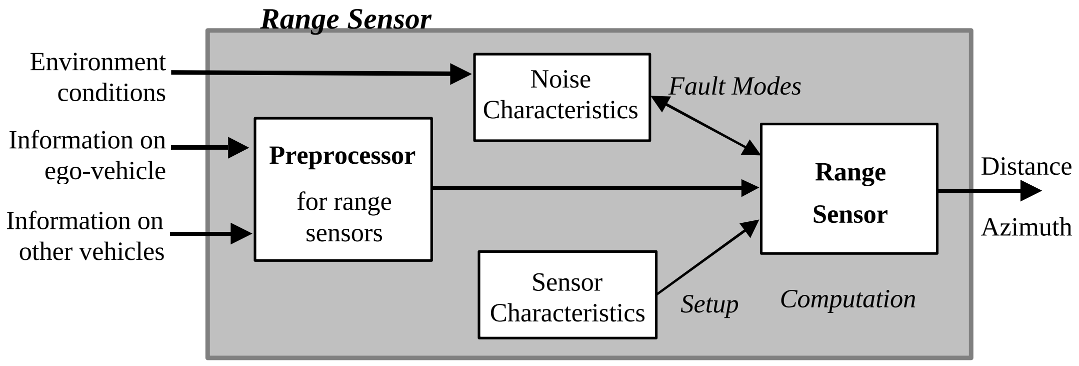


Figure 3. Implementation structure for a functional range sensor.

A second type of sensor class we present here is for positioning. We will only consider two simple implementations using global positioning systems and dead-reckoning. The implementation structure of a positioning sensor is very similar to the one given in Figure 3, except for the preprocessing phase. The position of the vehicle provided by the simulation environment is “corrupted” with the defined noise model based on the sensor characteristics, and the desired output is generated. A positioning system using a satellite network can be implemented by simply adding Gaussian measurement noise to the actual position of the vehicle. The data provided[10] for GPS (and GLONASS), and the data for differential global positioning systems obtained by Navlab vehicles[11] suggest that the longitude and latitude readings from a global satellite system can be characterized by independent Gaussian distributions. While the errors in individual encoder readings for dead-reckoning may be assumed to be Gaussian distributions (with a relatively small variance as compared with differential GPS), the dead-reckoning errors grow as the measurement error accumulates over time.

The functional sensor models are implemented using *Shift*, a hybrid system simulation programming language. The Shift simulation platform[12,13] is suitable for abstraction of complex applications such as automated highway systems, air traffic control systems, and other systems whose operation cannot be captured easily by conventional models. This language enables the user to define his/her own simulation domain for a particular application such as Automated Highway Systems[14]. The Shift programming language is chosen because of its ability to simulate discrete and continuous models, and the existence of a simulation environment for AHS applications. The sensor models in this paper are designed using part of the existing simulation environment definitions in Shift[14]. The structure given in Figure 3 is implemented as state transitions between fault modes and periodic sampling of vehicle positions and orientations for range calculations. Tables 2 and 3 show simplified examples of Shift code for range and position sensors.

Table 2. Simplified Shift[12] code for range sensor with ray definitions.

```
/* External functions */
function dist5 (number sen_x; number sen_y; number sen_or;
                [...]
                array(number) out distvec) -> number;
function min_of_array (array(number) vec; number vecsize;
                       array(number) out result) -> number;

/* Rangesensor with ray definitions
type RangeSensor_R
{
  input number gxp, gyp, gzp;         // Position of the vehicle
        number vgam11, vgam12,        // Orientation of the vehicle
  output continuous number distance;  // Distance
         continuous number angle;     // and angle to the closest vehicle
  state AutomatedVehicle vehicle;     // Vehicle associated with the sensor
        number maxrange;              // Maximum range in meters
        number hfov, nray;            // Half of the horizontal field of view, n. of
rays
        number xs, ys, sor;           // Sensor position and orientation
        number procspd;               // Processing speed for the sensor
        array(number) rayvec;           // Definition of ray angles
        set(AutomatedVehicle) inrange;  // set of the vehicles in range
        array(number) range;            // Returned readings for all rays
        number vl :=5, vw :=2;        // Vehicle length and width
        [...]
  discrete sense;
  setup
    do{
        t := 0;
        // Slopes of the lines defining the rays (vehicle coord. frame)
        rayvec := [(hfov-2*hfov/(nray-1)*i)/180*PI : I in [0 .. nray-1]];
      };
  flow default {t' = 1;    };
  transition
    sense -> sense {} when t >= procspd
    define
    {/* Set of the vehicles in range
     set(AutomatedVehicle) inRange := {z : z in AutomatedVehicles | z /= vehicle };
     /* Create arrays (pos. and orientation) for all vhicles in range */
     array(number) vehx := [gxp(x) : x in inRange];
     [...]
     /* Sensor position and orientation in global coordinate frame */
     number sen_x := gxp+xs*vgam11+ys*vgam21;
     [...]
     /* Evaluate the array for range readings */
     number dum := dist5(sen_x,sen_y,sen_or,vehx,vehy,veh11,veh12,
                         size(inRange),vl,vw,rayvec,raydist);
     /* Find minimum value and its index */
     number dum2 := min_of_array(raydist,nray,result);
    }
    do
    {t := 0;
     /* NOTE: If noise to be added to range readings, this can be done here;
        an alternative method is to add noise in the c-function */
     [...]
     distance := result[0];                   // Minimum distance
     angle := hfov-2*hfov/(nray-1)*result[1]; // Azimuth for the ray with minimum
reading
    };
}
```

Table 3. Simplified Shift[12] code for positioning sensor (GPS).

```
Type PositionSensor_GPS
{
  output continuous number pos_x, pos_y;   // Sensor readings
         continuous number s;              // Measurement signal:
                                           // "exists" [1] or "not" [0]
         continuous number err_x, err_y;   // Measurement errors;
  input  continuous number gxp, gyp;      // Actual position and
         continuous number precip;        // Precipitation
  state  AutomatedVehicle vehicle;        // Vehicle associated with sensor
         continuous number mean, var;     // Measurement noise parameters
         Gaussian err_signals;

  discrete normal{measure;},      // discrete states for operation modes
           problem{measure;},
           nodata{nomeasure;};
  setup
   define{Gaussian tnoise := create(Gaussian);}
   do{
      err_signals := tnoise;
      mean := 0;         // both measurements are characterized as
      var := 0.3*0.3;    // Gaussian distributions};
  flow
   measure{
           err_x = sqrt(var) * sg1(err_signals) + mean;
           err_y = sqrt(var) * sg2(err_signals) + mean;
           pos_x = gxp + err_x; pos_y = gyp + err_y;},
   nomeasure{
             pos_x = 0.0; pos_y = 0.0; err_x = 0; err_y = 0;};
  transition
   normal -> problem {} when precip >= 10 and precip < 60
    do {var := 0.9*0.9; s := 1;},
   normal -> nodata {} when precip >= 60
    do {s := 0;},
   problem -> normal {} when precip <= 10
    do {var := 0.3*0.3; s := 1;},
   problem -> nodata {} when precip > 60
    do {s := 0;},
   nodata -> normal {} when precip <= 10
    do {var := 0.3*0.3; s := 1;},
   nodata -> problem {} when precip <= 60
    do {var := 0.9*0.9; s := 1;};
}
```

4. SIMULATION EXAMPLES

In this section, we will present the simulation results for the two sensor types we discussed in the previous section: range sensing and vehicle positioning. First, let us consider the scenario given in Figure 4: a vehicle equipped with a sidesensor is travelling in the left lane; two other vehicles travelling in the right lane with speeds greater than the equipped vehicle are slowly entering the side sensor’s range. For this scenario, the vehicle lengths and widths are assumed to be 5m and 2m respectively. The sensor’s maximum range is taken as 10m, while the number of rays is defined as 7 with a field of view of 40 degrees. The vehicles are travelling at the middle of 4-m wide lanes. The sampling rate for the sensor is 0.2sec (5Hz). The speed of the vehicles on the right is 1.5m/s faster than the equipped vehicle, and the distance between them is 7.5m. There are six pseudo-vertex points defined. We have assumed perfect sensor samples (without noise) in order to present the results clearly.

Figure 5 shows the results of the simulation runs for three different types of sensors we discussed earlier. The plots show the changes in the sensor range and azimuth readings for the time interval between t = 0.5sec and t = 12.5sec. At the beginning of this time interval, vehicle 2 approaching vehicle 1 enters the side sensor’s field of view. Around t = 6.0 sec, vehicle 2 slowly moves out of the field of view, while vehicle 3 is enters it. As seen in Figures 5(a) and 5(b), vehicle detection with the simple model occurs later than in other cases, since the computations are based on the vehicle center of gravity. The closest distance returned by the sensor is 3m, the minimum possible distance between the sensor and the center of gravity of the detected vehicle, instead of 2m— the minimal distance between two vehicles travelling in adjacent lanes (all according to the parameters given above). The maximum possible range reading, and the zero azimuth angle are returned by the sensors if there are no vehicles in the field of view. For this simplest case, the returned value of the angle is continuous to the resolution of the sensor sampling frequency.

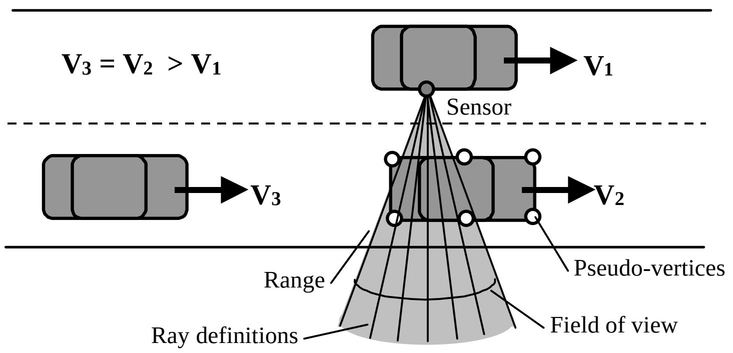


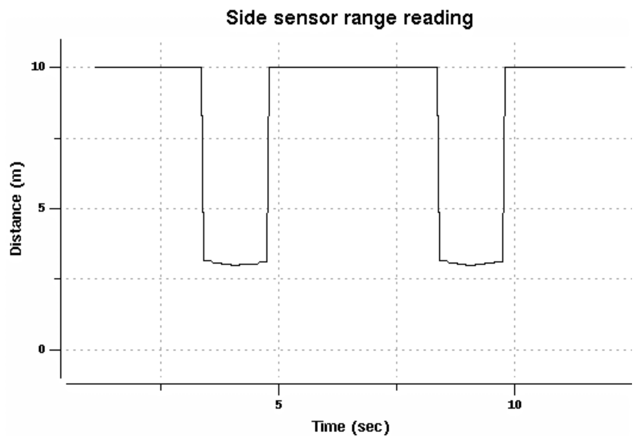
Figure 4. Simulation scenario for range sensor tests: two passing vehicles are detected by the right side sensor.

Figures 5(c) and 5(d) illustrate the effect of pseudo-vertex approach in close range readings. Although this approach returns better results than previous case, and is computationally less intensive than the ray-definitions approach, it may fail to capture the shape of the detected vehicle. In some situations, such as the one discussed here, the returned values are not always realistic. As seen in Figure 5(c), the distance to the vehicle in range “jumps” from approximately 2m to approximately 4m erroneously. This is due to the fact that the pseudo-vertex points in the closer side of the vehicle in range falls out of the field of view (See Figure 4). Furthermore, as indicated in Figure 5(d), the azimuth readings include large errors even if the vertex points on the correct side are tracked. This sensor type must be used only when the problem described here can be avoided.

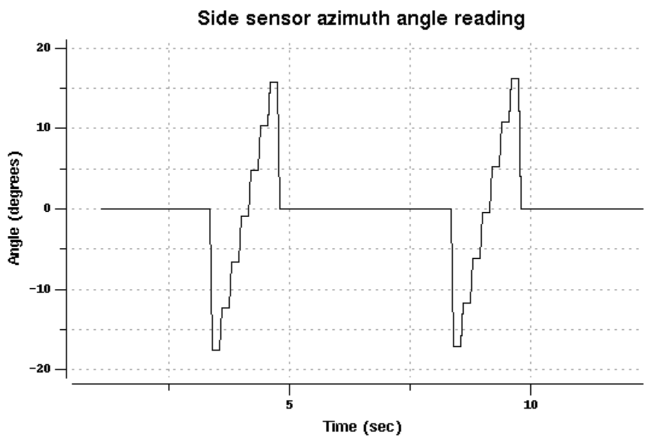
The most complex model we currently implement uses the ray definitions. Note that the plots in Figure 5 give the values for the closest intersection point between the rays and the edges of the rectangular vehicle definition. As seen in Figure 5(e), the range calculations are very realistic. The notch around t = 6.5sec indicates the moment where vehicle 2 started moving out of the sensor filed of view while vehicle 3 is entering. This can be seen more clearly from Figure 5(f) where the returned value of the azimuth angle to the closest point changes from –20 degrees to 20 degrees. Because the resolution for the sensor is defined by 7 rays, the azimuth angle is approximate for close ranges.

For the last two approaches, using more points/rays increases the accuracy of the sensor model while creating computational difficulty. The trade-off between resolution and computational effort is a function of the specific AHS scenario at hand. There will be applications where even the simplest sensor model is sufficient for design and/or analysis of a traffic, driver or vehicle controller model.

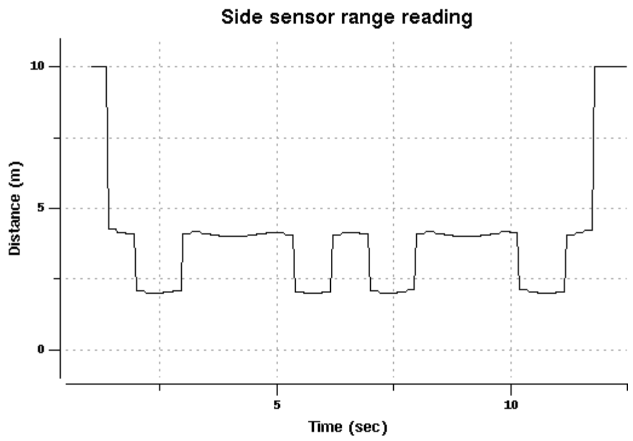
Our second simulation example is given in order to distinguish between different types of sensors used for the same purpose, and to indicate the similarities between their simulation models. Assume that a vehicle is equipped with a (differential) global positioning system, and an on-board sensor such as an encoder for dead reckoning. As mentioned above the same noise structure is used for both sensors by setting the mean and variance of the noise model according to empirical data. Although the real implementation is much more different, we use the same simple functional model with minor differences for both sensors. The main difference between the GPS and the encoder system is the noise characteristics; the rest of the information is provided by the simulation environment. Minor additions in the code given in Table 2 enables the user to emulate error accumulations for the dead-reckoning sensor. Assume that a vehicle is travelling along a straight segment of road. We assume that the position data can be sampled at a rate of 0.1sec (10Hz). The mean and variance for the measurement noise in the encoder is taken as 10^{-3}m and 4×10^{-6}m respectively, while these values for a differential GPS is approximately 0m and 9×10^{-2}m. We have also assumed that the variance of the GPS readings increase to 0.81m when there is a 10% or greater precipitation in the segment currently traveled by the vehicle. (The precipitation is defined as a percentage in current Smart-AHS platform[14].) The same fault-model descriptions can be used for other sensors as well.



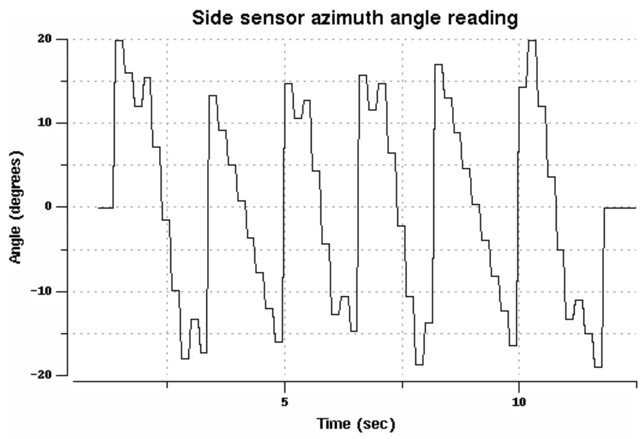
(a)



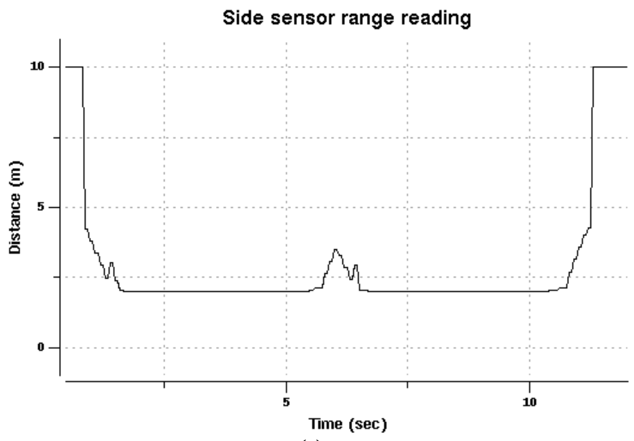
(b)



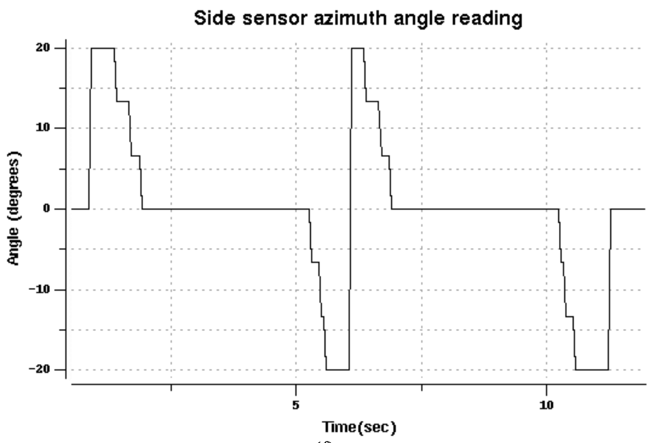
(c)



(d)



(e)



(f)

Figure 5. Distance and azimuth angle readings for different types of side sensors: (a-b) point vehicles with sensor field of view and orientation definitions, (c-d) pseudo-vertex approach, (e-f) ray definition approach (See text for details).

Figure 6 shows the data generated by the simulation runs in Shift. The plots show the sensor readings for an interval where the vehicle travels between 50m and 150 m down the road. The precipitation is 15% on the road segment between x = 50 and x = 100m. As seen from Figure 6a, the error in position reading with the encoder is accumulating while the sampled readings contain little deviations from the actual value. On the other hand, the GPS system returns independent readings that can be characterized as a Gaussian distribution around the actual position. The deviations from the actual value change with the transition of the sensor state to a different mode of operation due to the vehicle position on the road segment. The basic ideas described here will be extended to more complicated error models, and to other sensors.

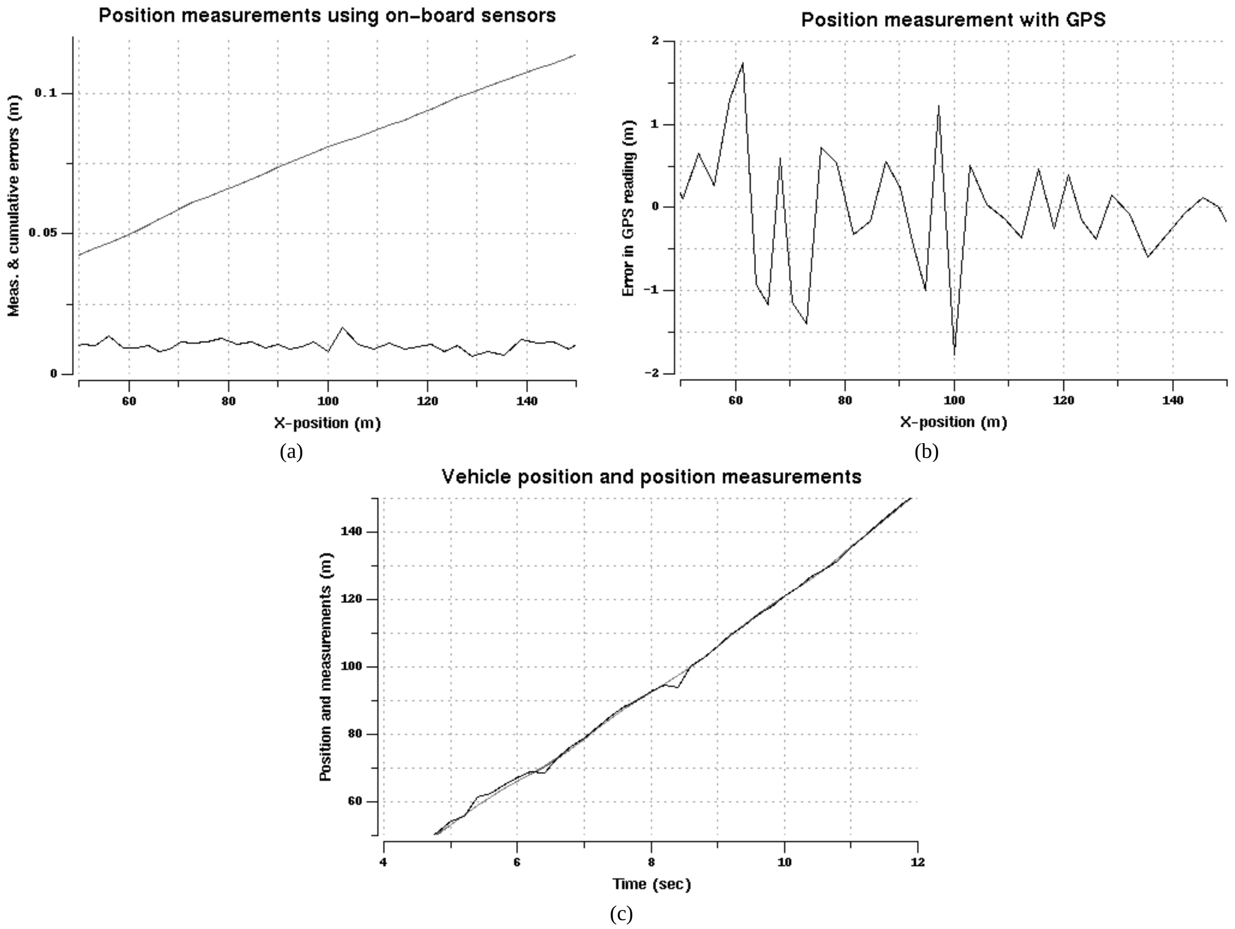


Figure 6. Vehicle position readings with (a) on-board sensors and (b) GPS.

6. CONCLUDING REMARKS

Functional sensor models, unlike most existing sensor models, are designed to bridge the gap between a driver model and its underlying simulation environment— providing several benefits. First, since functional sensors provide realistic sensor problems for simulated vehicles, algorithm designers are forced to recognize that perception should not be taken for granted in the real world. Thus, it is hoped that algorithms developed in simulations employing functional sensor models will require less modification when implemented on real intelligent vehicles. Second, functional sensors enable researchers to recreate existing sensor systems in simulation. For example, an intelligent vehicle driver model may be stated as using the ALVINN lane-tracking system[15]. By encapsulating the entire perception system as a functional sensor, the driver model can operate as if it were connected to a real ALVINN. The sensor model takes responsibility for providing the same interface as ALVINN without modeling details about its vision subsystem, underlying neural network, or training procedure[16]. Third, these sensor models can be used to develop intelligent vehicle algorithms that cannot currently be implemented on real robots due to the limitations of today's sensor technology. For instance, many tactical driving systems require accurate, long-term tracking of all the surrounding vehicles, with accurate estimates of their relative positions and velocities. Obviously, the functional sensor of this type is much easier to build than its real counterpart. By testing the driving systems in simulation using functional sensors, researchers gain a better understanding of their sensor requirements, and can thus focus sensor-building goals in a particular direction with more confidence.

This research is by definition quite open-ended. Functional sensors are refined as the needs of the driver models change, and as the underlying simulators mature. Our plans for future work in this area include:

- Extending existing functional sensors in response to developments in AHS microsimulator technology: For example, the current functional range sensors perform geometric calculations in 2-D. As full 3-D microsimulators become more common, the functional sensors will be updated. Similarly, more complex noise models will become practical with increases in processor speed. The implementation of different sensor types will be continued in conjunction with the sensor requirements and capabilities of the Navlab vehicles[11].

- Developing more detailed fault-modes for functional sensors: In some cases, this can require extensions to the simulation environment. For example, if a functional GPS is designed to fail inside tunnels, the microsimulator must explicitly model tunnels.

- Applying functional simulated sensors to other mobile robot domains: The current implementations of functional sensors have been restricted to highway simulators. However, functional sensors can be easily be applied to other simulated environments such as underwater robotics or space exploration.

Simulated runs and real-life testing are both essential in the development of a practical Automated Highway System. While realistic functional sensor models cannot eliminate the need for real sensors, our framework facilitates the productive investigation of AHS concepts and scenarios in simulation.

ACKNOWLEDGEMENTS

The authors would like to thank Alex Göllü, Jim Misener, and other members of the SHIFT Team at UC Berkeley/PATH for their valuable insights on sensor modeling and efforts in providing the simulation platform. This work was supported by USDOT under Cooperative Agreement Number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium.

REFERENCES

1. P. Varaiya, “Smart Cars on Smart Roads: Problems of Control,” *IEEE Transactions on Automatic Control*, Vol. 38, No. 2, pp. 195-207, Feb. 1993.
2. R. Sukthankar, *Situation Awareness for Tactical Driving*, Ph.D. Dissertation, CMU-RI-TR-97-08, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.
3. F. Eskafi, D. Khorramabadi, “SmartPath: An Automated Highway System simulator,” Technical Report UCB-ITS-94-3, University of California-Berkeley, 1994.
4. A. Göllü, *Object Management Systems*, Ph.D. Dissertation, University of California-Berkeley, CA, 1995.
5. R.C. Colgin, *Description and Analysis of a Bayesian CFAR Processor in a Nonhomogeneous Clutter Background*, Ph.D. Dissertation, Oakland University, Rochester, MI, 1995.
6. H.R. Everett, *Sensors for Mobile Robots: Theory and Application*, A.K. Peters, Wellesley, MA, 1995.
7. M.L. O'Connor, G.H. Elkaim, B.W. Parkinson, “ Carrier-phase DGPS for Closed-loop Control of Farm and Construction Vehicles,” *Navigation*, Vol. 43, No. 2, pp. 168-178, Summer 1996.
8. Ü, Özgüner, “Radar-based Convoying Using a Frequency Selective Surface Path,” *Proceedings of the 6th Annual Meeting of ITS America*, Vol. 2., pp. 750-755, 1996.
9. A. Andrews, “Theoretical and Empirical Analysis of PATH Magnetic Lane Tracking for the Intelligent Vehicle Highway System,” PATH Research Report UCB-ITS-PRR-92- 9, University of California-Berkeley, 1992.
10. P.N. Misra, M. Pratt, R. Muchnik, B. Burke, and T. Hall, “GLONASS Performance: Measurement Data Quality and System Upkeep,” *Proceedings of ION GPS-96*, pp. 261-270, The Institute of Navigation, 1996.
11. T. Jochem, D. Pomerleau, B. Kumar, and J. Armstrong, “PANS: A Portable Navigation Platform,” *IEEE Symposium on Intelligent Vehicles*, Detroit, MI, USA, September 25-26, 1995.
12. A. Deshpande, A. Göllü, and L. Semenzato, “The SHIFT Programming Language and Run-time System Dynamic Networks of Hybrid Automata,” PATH Research Report UCB-ITS-PRR-97-7, University of California-Berkeley, 1997.
13. SHIFT Team, “SHIFT, the Hybrid System Simulation Programming Language,” http://www.path.berkeley.edu/shift/ (September 9, 1997).
14. A. Deshpande, “AHS Components in SHIFT (Draft),” PATH Report, http://www.path.berkeley.edu/publications.html, (September 9, 1997).
15. D. Pomerlau, *Neural Network Perception for Mobile Robot Guidance*, Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, 1992.
16. R. Sukthankar, J. Hancock, C. Thorpe, “Tactical-level Simulation for Intelligent Transportation Systems,” *Journal on Mathematical and Computer Modeling, Special Issue on ITS*, 1997.