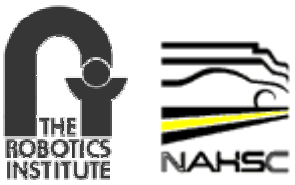

Human Driver Models for AHS Simulations

Cem Ünsal

CMU-RI-TR-98-02

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania
15213-3890

February 1998

This work was supported by US Department of Transportation under Cooperative Agreement number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expresses or implied, of the U.S. government.

Table of Contents

1 Introduction

This document describes the human driver models designed for automated highway-vehicle systems simulations in SHIFT/Smart-AHS. The models given in this document are defined using SHIFT programming language, and some of the Automated Highway System components in Smart-AHS platform [2,3]. Therefore, SHIFT and Smart-AHS packages provided at simulation web site [1] and files described in [4] are required. Human driver models given here include simple two-dimensional vehicle, vehicle controller, and sensor descriptions. All necessary files are available on-line at `http://www.cs.cmu.edu/~unsal/research/shift`; the source code is also given here and in the document describing the sensors [4].

The next section describes the general structure of the human driver model in SHIFT. In order to make decisions about vehicle motion, the driver model requires information about its lane position, speed, relative positions of other vehicles, etc. These are provided by the sensor modules as well as the *vehicle roadway environment processor* (*VREP*), a module defined in Smart-AHS library. Sections 3, 4 and 5 introduce three different driver models, and their discrete states (of mind) and decision structures. Additional models required for the simulation are described in Section 7.

2 General Structure

The general structure for human driver models in SHIFT/Smart-AHS is given in Figure 1. A basic simulation includes the decision and pursuit modules as well as additional definitions for vehicle kinematics, vehicle controllers, sensor(s), vehicle roadway environment processor (*VREP*), and/or *sensor environment processor* (*SEP*). The structures of different human driver models are very similar, except a few differences in the sub-module definitions and, consequently, their connections.

The vehicle model is 2-D, and therefore some of the VREP inputs are zero. Information about the vehicle and road orientation, lane position and road curvature is sent to the controller in order to generate a steering command based on the road curvature and the relative position of the vehicle with respect to the lane/road. Global and/or road positions as well as the current lane are connected to appropriate sensor inputs for distance and azimuth calculations (See [4] for details). Global positions are also used for *Cell* definitions in sensor environment processor (*SEP*).

The vehicle controller takes desired speed, desired lane position, and current values of the lane position, vehicle/road orientation, wheel speed and road curvature to generate longitudinal acceleration and steering. This module can be visualized as a driver's control actions following his decisions, or as an automated vehicle controller following manual speed and lane change commands.

Vehicle model is kinematic; it is a standard bicycle model with wheel base definition. The center of gravity of the vehicle is assumed to be at the middle of rear axle. The longitudinal and lateral displacement as well as the vehicle rotation around its axis normal to the ground are defined as outputs, and connected to vehicle roadway environment processor (*VREP*).

The sensor module uses the information provided by *VREP*, *SEP* and/or global set of vehicles to evaluate one or more of the following (depending on its type), and sends these to the decision module:

- distance to the closest vehicle
- azimuth angle to the closest vehicle
- headway distance
- tailway distance
- longitudinal distance to the closest vehicles in the side lanes, front and back
- longitudinal distance to the second closest vehicle in the same lane or adjoining lanes (front and back)
- rate of change in headway distance

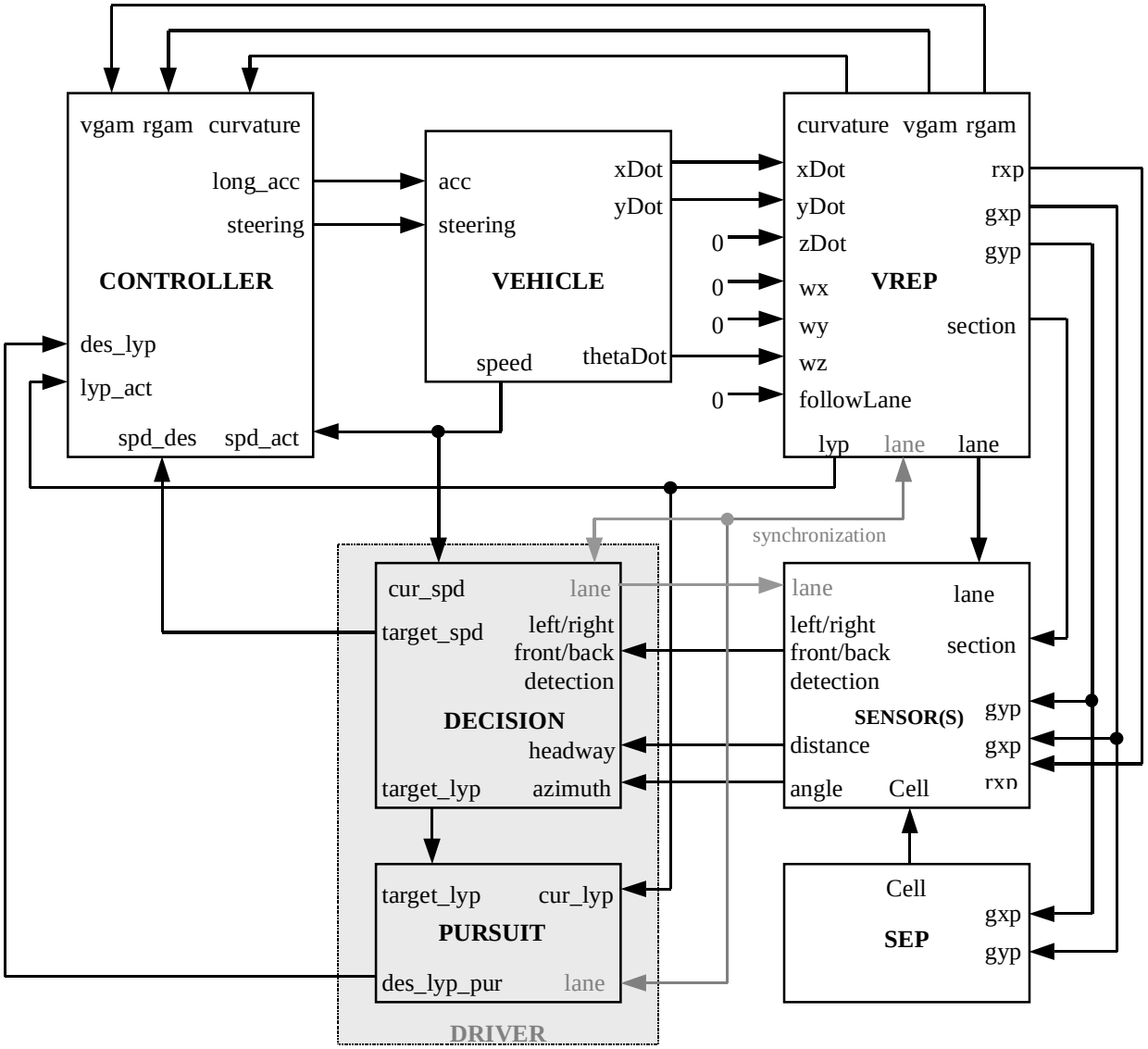

Figure 1. General structure of the Human Driver Model in SHIFT.

If the number of vehicles in the simulation is large, then geometric hashing methods can be used for sensor measurements and/or vehicle detection. The two alternative are (a) the cell structure defined as part of the roadway description in Smart-AHS [2] and (b) the sensor environment processor and the global grid of cells [4].

For the description of cell structure in roadway definitions, see Smart-AHS manual pages [5]. The sensor environment processor (*SEP*) uses the global position of its associated vehicle, and the global set of *Cells* to evaluate the current cell the vehicle is travelling in. This information is then sent to the sensor module to be used in the definition of the subset of vehicles to be checked by the sensor.

The decision module (main part of the driver model) takes current distance (headway), azimuth, left/right detection and/or current speed information to make an intelligent decision on the target speed and lane position. Target speed is directly sent to the vehicle controller model while the target lane position is passed on to the pure pursuit point model.

Pure pursuit point model evaluates the desired pursuit point location on the lane based on the target lane position indicated by the driver module and the current lane position of the vehicle provided by VREP. Pursuit and decision modules are synchronized to lane change transition of associated VREP.

The driver models are described in Sections 3, 4, and 5. All other modules and the main vehicle description mentioned above are introduced in Section 7.

3 Human Driver Model A

This first model is designed to keep a predefined headway distance. The target speed, which is sent to the vehicle controller module, is adjusted so that the headway distance is close to desired headway, which is a function of the current vehicle speed. If the headway distance is less than a minimum acceptable value, the target speed is set to zero. If the distance is larger than a predefined value, the driver transitions to cruise state where it tries to match a preset value.

The driver also takes random actions by a predetermined frequency. Possible actions are changing lanes –if there are no vehicles in the side sensor range- and a random change in the cruise speed.

The behavior of this driver model can be adjusted by using the parameters defined in Section 3.2.

3.1 Inputs, Outputs, and Discrete States

Human driver model A has the following inputs:

- current speed (*current_spd*)
- current headway (*current_hdwy*)
- detection information from side sensors (*sensor_lf*, *sensor_lb*, *sensor_rf*, *sensor_rb*)

The speed input is connected directly to the vehicle model, but an additional sensor model can be placed between vehicle and driver modules. The headway and side detection data are provided by the sensor modules described in [4].

The following are the outputs of the driver module:

- target speed (*target_spd*)
- target lane position (*target_lyp*)
- desired headway (*des_hdwy*; not connected)

Target speed is directly sent to the vehicle controller. Target lane position is sent to the pursuit module, which is used to implement lane-changing behavior of a human or automated driver. Detailed information on pursuit model can be found in Section 7.2. The details of the connections are shown in Figure 1.

There are five different discrete states for the driver model A:

- Cruise (*cruise*)
- Follow (*follow*)
- Stop (*stop*)
- Change left (*change_left*)
- Change right (*change_right*)

The behavior of the driver in each discrete state is defined using flow equations in SHIFT [1]. In state *cruise*, the driver tries to match its desired cruise speed. The change in the target speed is proportional to the difference in the target and desired speeds.

In *follow* mode, the driver adjust her speed to match the desired headway distance, which is defined as a function of the current vehicle speed. Current headway and desired headway distance are compared, and the desired change in the target speed is calculated. If the calculated target speed is

larger than the maximum acceptable speed, then the change is reset to zero. (See source code in Section 6 for details.)

In *stop* mode, the target speed is set to zero. *Change left* and *change right* modes forces the driver to keep its last target speed definition before transition, and change the target lane position value to +/- *lane width* for lane change. This target lane value is kept until lane change is indicated by an external event of *VREP*.

3.2 Model Parameters and Variables

The model parameters and variables used in this simple driver model are:

- *speed_change* Evaluated change in target speed (m/s^2).
- *speed_change2* Change in target speed; input to the limiting function (m/s^2).
- *aa* Preset acceleration and deceleration values for *follow* mode (m/s^2);
- *dd* used to change target speed.
- *max_speed* Maximum possible speed for the driver (m/s).

- *min_hdwy* Headway distance limit for transition between states *stop* and *follow* (m).
- *des_hdwy_limn* Parameters defining the comfort region around the desired headway value (m).
- *des_hdwy_limp*
- *hlim* Limiting value of the headway distance for transition between states *follow* and *cruise* (m).

- *lwi* Lane width (m).

- *cruise_ speed* Desired speed for *cruise* mode (m/s).
- *ad* Proportional gain constant for tracking *cruise_speed* (1/s).

- *t* Time (s).
- *action* Value indicating the random action to be taken at *t = actiontime*.
- *actiontime* Random time interval for actions; defined at the setup phase (s).

- *myrrep* Associated VREP.

3.3 Discrete State Transition Rules

Table 1 on page 6 gives the conditions under which the discrete state transitions take place. The parameters, model variables and the system inputs are described in the previous section.

3.4 Other Issues

Certain sets of parameters may result in oscillatory behavior in vehicle speed (and consequently headway distance) in follow mode. The parameters are to be adjusted carefully. If necessary, the follow behavior can be slightly changed to “do not follow vehicle when the rate of change in headway is positive.”

Table 1. State transitions.

#	From	To	When
1	cruise	stop	current_hdwy ≤ min_hdwy
2	follow	stop	current_hdwy ≤ min_hdwy
3	stop	follow	current_hdwy ≤ min_hdwy * 1.5
4	cruise	follow	current_hdwy < hlim
5	follow	cruise	current_hdwy ≥ hlim
6	follow	change_left	t ≥ actiontime AND action < 1 AND sensor_lb = -1 AND sensor_lf = -1
7	follow	change_right	t ≥ actiontime AND action ≥ 1 AND action < 2 AND sensor_rb = -1 AND sensor_rf = -1
8	cruise	change_left	t ≥ actiontime AND action < 1 AND sensor_lb = -1 AND sensor_lf = -1
9	cruise	change_right	t ≥ actiontime AND action ≥ 1 AND action < 2 AND sensor_rb = -1 AND sensor_rf = -1
10	cruise	cruise	t ≥ actiontime AND action ≥ 2
11	change_left	cruise	External event synchronization myvrep:UpdateLaneLeft
12	change_right	cruise	External event synchronization myvrep:UpdateLaneRight

4 Human Driver Model B

The second human driver model is designed to keep a desired headway distance when there is a vehicle in predefined sensor range, and to keep a desired speed otherwise. If the headway distance is too close or its rate of change is a large negative value, the driver changes lanes to avoid a collision. If lane change is not possible, the driver tries to stop. Details of the driver behavior are given in the following sections. The behavior of this driver model can be adjusted by using the parameters defined in Section 4.2.

4.1 Inputs, Outputs and Discrete States

Human driver model B takes the following as inputs:

- current speed (*current_spd*)
- current headway (*current_hdwy*)
- detection information from side sensors (*sensor_lf*, *sensor_lb*, *sensor_rf*, *sensor_rb*)

The speed input is connected directly to the vehicle model, but an additional sensor model can be placed between vehicle and driver modules. The headway and side detection data are provided by the sensor modules described in [4].

The following are the outputs of the driver module:

- target speed (*target_spd*)
- target lane position (*target_lyp*)
- desired headway (*des_hdwy*; not connected)

Target speed is directly sent to the vehicle controller. Target lane position is sent to the pursuit module, which is used to implement lane-changing behavior of a human or automated driver. Detailed information on pursuit model can be found in Section 7.2. The connections are shown in Figure 1.

There are six different discrete states for the driver model B. These are:

- Cruise (*cruise*)
- Follow (*follow*)
- Follow closely (*followc*)
- Change left (*change_left*)
- Change right (*change_right*)
- Stop (*stop*)

The behavior of the driver in each discrete state is defined in flow equations in SHIFT [1]. In state *cruise*, the driver tries to match its desired cruise speed. The change in the target speed is proportional to the difference in the target and desired speeds.

In *follow* and *follow closely* modes, the driver adjusts its speed according to available headway distance. Desired headway distance where the driver is comfortable is defined by a function of the current speed. Current headway and desired headway distance are compared, and the desired change in the target speed is calculated using headway rate as an additional constraint. The behavior is summarized in Table 2 and Table 3. If the current headway distance is close to the desired headway value, then no change in the target speed is made. If the current headway value differs from the desired value by a large amount (define *by lim2p* and *lim2n*) then preset acceleration/deceleration values are used to adjust target speed. For smaller deviations (defined by *lim1n* and *lim1p*) target speed is adjusted if the rate of change of the headway is appropriate.

The only difference between two follow modes is the definition of the parameters. These and the transition rules between states are discussed in the following sections.

Change left and *change right* modes force the driver to keep its last target speed definition before transition, and change the target lane position value to *+/- lane width* for lane change. This target lane value is kept until lane change is indicated by a *VREP* event.

In discrete state *stop*, the driver's target speed is set to zero.

Table 2. Calculation of change in target speed in follow modes.

	Current headway region					
	-	*dh – lim2n*	*dh – lim1n*	*dh*	*dh + lim1p*	*dh + lim2p*	+
Value of a_1	*-dd*	If *h'* < 0 then *–dd* else 0	0	0	if *h'* > 0 then *aa* else 0	*Aa*	
dh = f(*current speed*) : desired headway *limxn*, *limxp* (*x = 1, 2*) : limiting values defining the regions *aa*, *dd* : preset acceleration, deceleration values							

Table 3. Limiting function on the change in target speed.

	Current target_speed region		
	- 1	max. speed	+
Value of target_speed'	max (a_1, 0)	a_1	min (a_1, 0)

4.2 Model Parameters and Variables

This section describes the model parameters and variables used in the simulations. These are:

- *hdwy_delay_t* Delay in seconds for headway rate calculations (sec).
- *min_hdwy* Headway distance limit for transition from state *follow(c)* to *stop* (m).
- *min_hdwy2* Headway distance limit for transition from state *cruise* to *stop* (m).

- *des_hdwy_lim1p*
- *des_hdwy_lim2p*
- *des_hdwy_lim1n*
- *des_hdwy_lim2n*

  Parameters defining the regions around the desired headway value (in meters; See Table 2 and Table 3).

- *aa*
- *dd*

  Preset acceleration and deceleration values for follow modes (m/s^2); used to change target speed.

- *lane_shift_hdwy* Limiting value of headway distance for lane changing decision (m).
- *lane_shift_hrate* Limiting value of headway rate for lane changing action; used with *lane_shift_hdwy* (m).
- *lane_shift_hrate2* Limiting value of headway rate for lane changing action (m).

- *cruise_speed* Desired speed for *cruise* mode (m/s).
- *ad* Proportional gain constant for tracking *cruise_speed* (1/s).

- *hlim* Limiting value of headway distance for transition between states *follow* and *cruise* (m).
- *hlim_delay* Limiting value of headway distance for transition between states *follow* and *followc*; also used to adjust command delay according to mode (m).

- *lwi* Lane width (m).

- *cmd_delay* Delay in speed command for *follow* mode (sec).
- *cmd_delay2* Delay in speed command for *folllowc* mode (sec).

- *max_speed* Maximum possible speed for the driver (m/s).
- *target_speed* Calculated target speed; input to the command delayers, i.e., *target_spd*(*t* + *cmd_delay*) (m/s).
- *speed_change* Evaluated change in target speed; a_1 in Table 2 (m/s^2).
- *speed_change2* Change in target speed; input to the limiting function (m/s^2).

- *myvrep* Associated VREP.

- *delayer_hdwy* Delayer for headway measurements.
- *delayer_cmd* Delayer for speed command (*follow* mode).
- *delayer_cmd2* Delayer for speed command (*followc* mode).
- *delayed_hdwy* Headway measurement delayed by *hdwy_delay_t* using *delay_hdwy* (m).
- *headway_rate* Rate of change in headway distance (m/s).

Some of the above parameters must be defined carefully in order obtain an intelligent driver model. Figure 2. shows some of the parameters and variables that are related to headway control structure. Desired headway and limiting value of the headway distance for lane changes are functions of the current vehicle speed, and therefore continuous variables. While the current headway measurement is compared with the desired headway, its value is also checked against other constant parameters (such as *min_hdwy*, *min_hdwy2*, *hlim*, *hlim_delay*) to adjust driver state. There are two separate functions for the limiting value of the headway distance for lane shifting actions (*lane_shift_hdwy*); one for each follow modes.

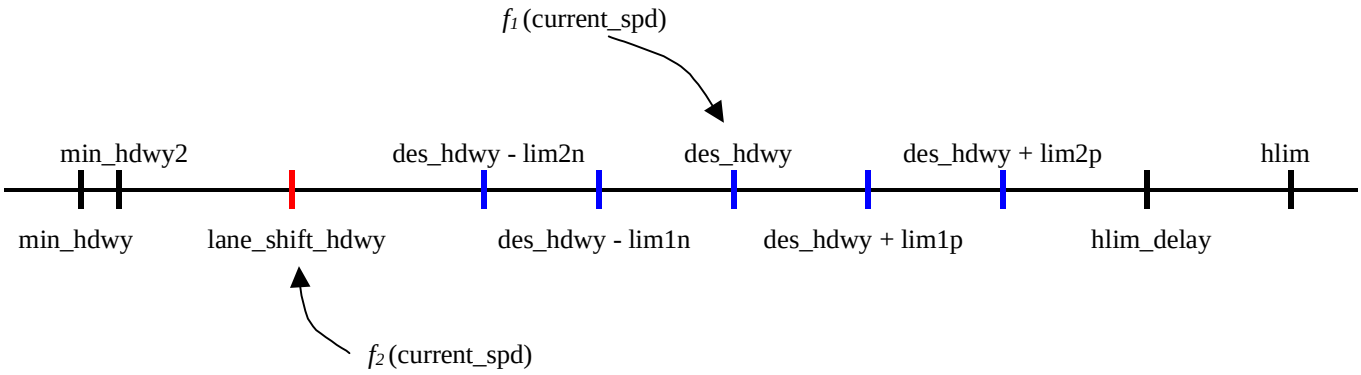

Figure 2. Relation between headway control parameters.

4.3 Discrete State Transition Rules

Table 4 on page 10 gives the conditions under which the discrete state transitions take place. The parameters, model variables and the system inputs are described in the previous sections.

4.4 Other Issues

For each discrete driver state, target speed is defined either algebraically or by using differential equations. In both cases, the evaluated value is fed into a command delayer that emulates driver's reaction to events. The evaluated target speed is sent to two separate delayers operating with different delay times. One of the delayer outputs is chosen depending on the current headway: for closer headway distances the reaction time is shorter. Delay times are defined in the setup phase.

The headway and side sensor measurements are not delayed. Adding delay to, say, headway measurements will provide a simple model for driver perception lag. Adding delay to sensor measurements (especially to the headway), would create a new set of programming problems, especially during lane changing maneuvers. Since a driver is more attentive during lane changes, it may be possible to remove perception delay during the lane changing maneuvers, and then switch back to normal operation mode (in terms of perception delay) once the associated *VREP* indicates lane change.

More complicated lane changing decision mechanisms is possible based on longitudinal distance measurements provided by the side sensors; the sensors are capable of providing distance and rate information. The details are given in [4].

Table 4. State Transitions

#	From	To	When
1	cruise	stop	current_hdwy < min_hdwy2
2	follow	stop	current_hdwy < min_hdwy
3	followc	stop	(current_hdwy < min_hdwy AND hdwy_rate ≤ 0) OR current_hdwy ≤ min_hdwy2
4	stop	followc	current_hdwy > min_hdwy AND hdwy_rate > 0
5	follow	followc	current_hdwy ≤ hlim_delay
6	followc	follow	current_hdwy > hlim_delay
7	cruise	follow	current_hdwy < hlim
8	follow	cruise	current_hdwy ≥ hlim
9	follow	change_left	((current_hdwy < lane_shift_hdwy AND hdwy_rate < lane_shift_hrate) OR hdwy_rate < lane_shift_hrate2) AND sensor_lb = -1 AND sensor_lf = -1
10	follow	change_right	((current_hdwy < lane_shift_hdwy AND hdwy_rate < lane_shift_hrate) OR hdwy_rate < lane_shift_hrate2) AND sensor_rb = -1 AND sensor_rf = -1
11	followc	change_left	((current_hdwy < lane_shift_hdwy AND hdwy_rate < lane_shift_hrate) OR hdwy_rate < lane_shift_hrate2) AND sensor_lb = -1 AND sensor_lf = -1
12	followc	change_right	((current_hdwy < lane_shift_hdwy AND hdwy_rate < lane_shift_hrate) OR hdwy_rate < lane_shift_hrate2) AND sensor_rb = -1 AND sensor_rf = -1
13	change_left	cruise	External event synchronization myrep: updateLaneLeft
14	change_right	cruise	External event synchronization myrep: updateLaneRight

5 Human Driver Model C

The third human driver model also keeps a desired headway distance when there is a vehicle in range, and otherwise tries to keep a desired cruise speed otherwise. Lane changing behavior is parameterized using a set of time flags. If the current vehicle speed differs from the desired cruise speed by a certain amount for a predefined time interval, the driver decides to change lanes. At this point, the lane change is arbitrary if both left and right lanes are open. If both adjacent lanes (if they exist) are occupied by other vehicles (for a certain longitudinal range), and the flag for lane change is set for a predefined time interval, then the driver decides to slow down hoping that this action will provide an opening in

adjacent lanes. The decrease in vehicle speed is provided by changing the evaluation function for the headway. (See following sections and the source code given in Section 6 for details.)

It may also be possible to use additional information from the sensor modules and define more complex behavior rules using the same driver structure given in this section.

5.1 Inputs, Outputs and Discrete States

Human driver model C has the same inputs, outputs and discrete states as the driver model B given in Section 4. The behavior of the driver in each discrete state is also similar. The only difference between driver models B and C is in the decision to make lane changes.

In *follow* and *follow closely* modes, the model tracks the time passed since the last lane change. If the vehicle is not traveling at the desired cruise speed, a flag is set after a predefined time interval (driver's patience has expired). After the flag is set, the driver changes lanes if the adjacent lane is not occupied by another vehicle. At this point, the model checks the whole range of side sensors, front and back (See sensor definitions in [4]) for vehicles; more complex decision rules are possible. If both left and right adjacent lanes (if they exist) are occupied, the driver decides to decrease her speed after a specified time interval hoping for an opening. Forcing a speed increase or a sequence of actions are possible by minor changes in the code.

5.2 Model Parameters and Variables

The parameters/variables for drive model C is very similar to the previous model in Section 4, except that the following parameters are added instead of *lane_shift_hrate, lane_shift_hrate2*:

- *lsft* Time passed since last transition to follow mode (s).
- *lsiflagt* Time interval for setting the flag for lane shift (s).
- *lsflagt2* Time interval for starting the speed decrease for a possible lane opening.
- *lsfspeedlim* Acceptable speed difference beyond which the flag timer starts counting (m/s).

The parameter *lane_shift_hdwy* is kept in order to guarantee safe operation. If the headway distance drop below this value, the lane changing decision is made. The relative position of this parameter on the real axis with respect to parameters *min_hdwy* and *min_hdwy2* affects the behavior of the driver: the transition to state *stop* from follow states may never occur.

5.3 Discrete State Transition Rules

Human driver model C differs from the previous model given in Section 4 only in its rules for transition to lane changing states. Therefore, only the transition with new rules are given in Table 5 on page 12. All other transitions are the same as those given in Table 4. Also, see the source code given in Section 6 for use of the second flag *lsflagt2* for speed adjustment.

5.4 Other Issues

Any combination of the models given in Sections 3, 4, and 5can be created with minor changes in the source codes. Although the models given here are designed for populating the highway during simulations and therefore, detailed behavior definitions may not be necessary, nor significant for the simulation, additional rules/transition for aborting lane changing actions can be added for more realistic behaviors.

Table 5. State transitions.

#	From	To	When
9	follow	change_left	(lsft ≥ lsflagt OR current_hdwy < lane_hift_hdwy) AND sensor_lb = -1 AND sensor_lf = -1
10	follow	change_right	(lsft ≥ lsflagt OR current_hdwy < lane_hift_hdwy) AND sensor_rb = -1 AND sensor_rf = -1
11	followc	change_left	(lsft ≥ lsflagt OR current_hdwy < lane_hift_hdwy) AND sensor_lb = -1 AND sensor_lf = -1
12	followc	change_right	(lsft ≥ lsflagt OR current_hdwy < lane_hift_hdwy) AND sensor_rb = -1 AND sensor_rf = -1

6 Source Code

Source code for human driver model discussed in Sections 3, 4, and 5 are given below:

<humandrivers.hs>

7 Additional Files for HDM Simulations

This section briefly describes the files used in HDM simulations. Some of these files/modules are described in [4] as indicated; other types such as the pure-pursuit point follower model, simple driver model and an example definition of the type *vehicle* are defined here.

7.1 Range Sensor Models

The following range sensors models used with the driver models given in this document can be found in [4]:

- frontsensor.hs
- leftsensor.hs
- rightsensor.hs
- backsensor.hs
- backsensor2.hs
- frontsensor2.hs
- leftsensor2.hs
- rightsensor2.hs
- frontsensor2_rate.hs

7.2 Pursuit Point Model

SHIFT type *Pursuit* is used to take the target lane position information from the driver (or the higher decision level in the control hierarchy), and relays it to the (lateral) vehicle controller with a predefined rate of change. This subroutine is required in order to make sure that the lateral controller is able to track the pursuit point.

A second task for this type is to guarantee smooth transitions for the desired lane position (lateral deviation) input to the lateral controller during lane changes. As seen in the highway and vehicle environment processor *VREP* descriptions in Smart-AHS [3], the lateral lane deviation value *lyp* "jumps" $-lw/2$ to $lw/2$ during left lane changes, or vice versa (*lw* indicates the lane width). The driver (or higher control level in the hierarchy) model currently uses a lane deviation value of $\pm lw$ to

indicate a desired lane change. Due to the change in the actual value at transition from one lane to the other, the value of the desired deviation also needs to be changed to guarantee smooth operation. The type *Pursuit* uses a conditional transition waiting for the lane change to take care of this problem.
This module takes target lane position from driver module, current lane position and current lane for *VREP* as inputs, and generates desired pursuit point as output. The rate of change for the desired lane position and its maximum value are user-defined parameters. See the source code given below for details.

7.2.1 Source Code

The SHIFT code for pursuit point evaluation is given below:

<pursuit.hs>

7.3 Two-dimensional Vehicle Model

Description of the two-dimensional kinematic model of the vehicle as well as the source code can be found in [4].

7.4 Vehicle Controllers

Descriptions of the lateral and longitudinal controllers used with the 2-D kinematic vehicle model as well as the source code are given in [4].

7.5 Simplest Driver Model

SHIFT type *Driver0* is designed to follow a predefined trajectory. It uses the same input output structure as the driver models described in Sections 3, 4, and 5. This model does not make any intelligent decisions on its speed, nor lateral position, except slowing down with predefined parameters when there is a vehicle in front of it. This model is used to test other driver models under different highway conditions.

The SHIFT code for the simplest driver model is given in Section 6.

7.6 Main Vehicle Description File

An example file (`ExVehicle.hs`) is included here to illustrate subtype definitions for a manually controlled vehicle simulation in Smart-AHS/SHIFT platform. As seen in the SHIFT code below, the set of vehicles is defined globally. Subtypes defined for the vehicle require inclusion of additional files for the SHIFT compiler. A generic type Vehicle includes a two-dimensional vehicle model (subtype *Vehicle_Kinematics*), a lateral and longitudinal control models (*Controller*), the pursuit subroutine (*Pursuit*), a simple driver model (e.g., *DriverA*), and several sensor models (e.g., *RangeSensor_PV*) as well as vehicle roadway environment processor (*VREP*), a source (*Source*), a sink (*Sink*), and/or a sensor environment processor (*SEP*). Global position, the velocity and the width and length of the vehicle can be defined as outputs for the vehicle.

In the setup phase, sensor, controller, and other model parameters are defined; the newly created vehicle is added to the set of vehicles. The initial location of the vehicle is inherited from the associated source. Figure 1. in Section 2 illustrates possible links between the subtypes constituting the vehicle.

The SHIFT code for the example vehicle description is below:

<ExVehicle.hs>

8 Scenario Description Files

Figure 3. illustrates the 4-section racetrack given in file `racetrack.hs`. The file includes four sources in the lower left section. For details, see the source code below and the web page http://www.cs.cmu.edu/~unsal/research/shift/track.html.

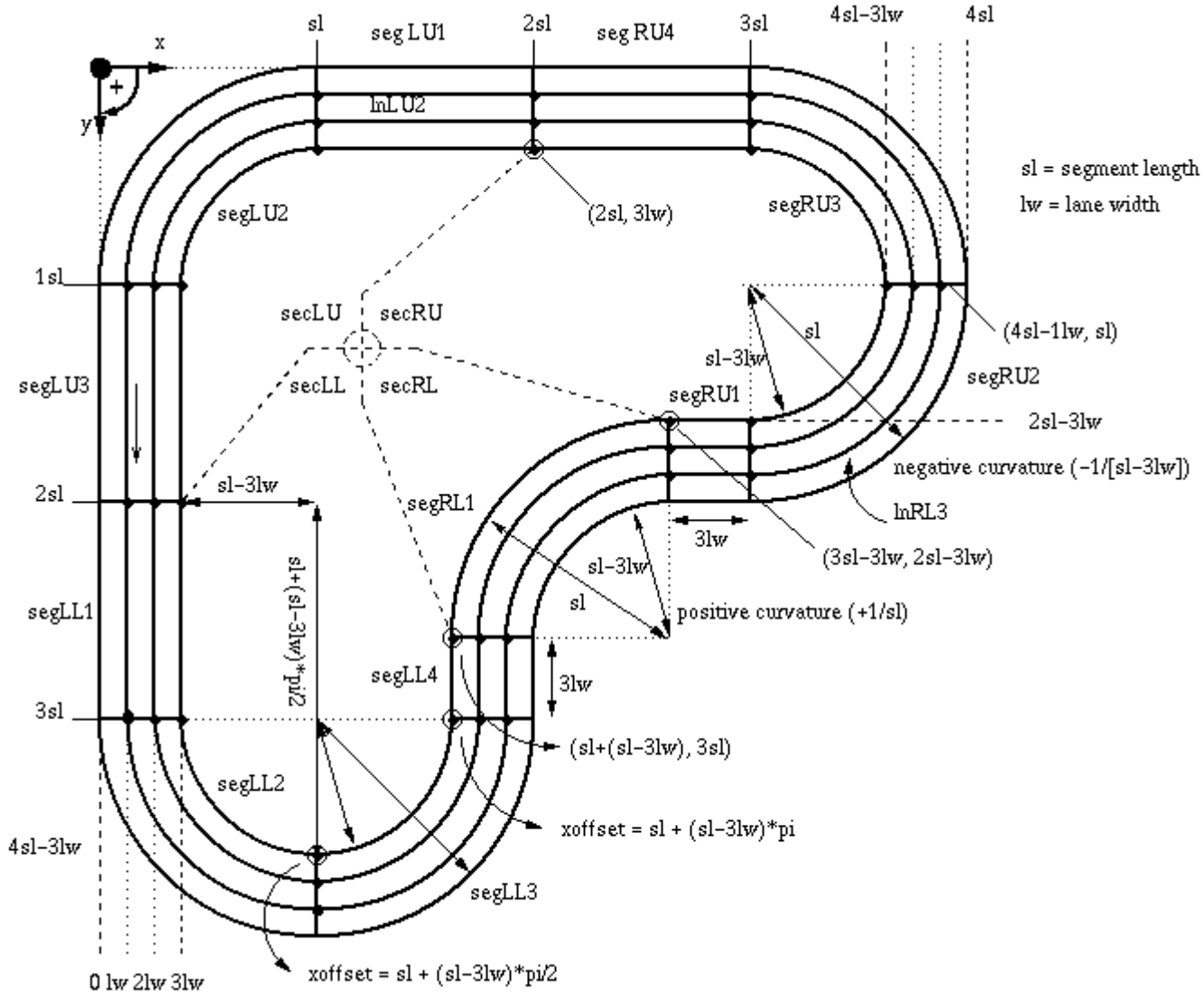

Figure 3. Racetrack with 4 sections, 3+4+1+4 =12 segments, and 3 lanes.

The SHIFT code for the racetrack description is given below:

<racetrack.hs>

9 Additional Information

9.1 Using Multiple Vehicle/Driver Types in a Single Simulation

As indicated in this document and in [4], several modules (i.e., SHIFT types) use the global set of vehicles for sensing, detecting, and making decisions. It is also possible to define different types of vehicles (e.g., one source creating vehicles taking random actions, another creating vehicle with decision-making drivers) and combine these into a single global set of vehicles using SHIFT class and hierarchy definitions.

However, our attempts to remove ego-vehicle from this heterogeneous set of vehicles for sensor evaluations resulted in run-time errors. A possible explanation of this condition is incomplete implementation of class hierarchy in simulation platform. It is possible to add different types of vehicle to the global set provided that all subtypes of vehicles belonging to the same class in the set; but extracting a specific vehicle from the set is not.

On the other hand, it is always feasible to define multiple sets for different types of vehicles. This solution will of course introduce additional calculations and/or repeated loops in implementation of every SHIFT type directly or indirectly related to range sensing and detection algorithms.

9.2 Files

All source code files listed below as well as this documentation are provided at `http://www.cs.cmu.edu/~unsal/research/shift/index.html`

- 2dkinectrl.hs
- 2dkineveh.hs
- backsensor.hs
- backsensor2.hs
- cell.hs
- frontsensor.hs
- frontsensor2.hs
- frontsensor_rate.hs
- gps.hs
- grid.hs
- grid_simpler.hs
- humandrivers.hs
- leftsensor.hs
- leftsensor.hs
- noise.hs
- pursuit.hs
- racetrack.hs
- rightsensor.hs
- sep.hs
- source_grid.hs

Sensor models and related files described here are compatible with Smart-AHS versions 0.45 and 0.60 [2], and were tested under SHIFT ver. 2.12 using Sun SPARC© Station 4 running SunOS 4.1.4.

The following text must included in the directory where the source files are located, under the name CONDITIONS:

```
/*************************************************************************\
* The files in this directory are distributed under the following    *
* conditions:                                                        *
*                                                                    *
* 1. The recipient shall refrain from disclosing the software,       *
*    in any form, to third parties without prior written             *
*    authorization from Carnegie-Mellon University.  The             *
*    recipient shall have the right to use and copy the              *
*    software on, or in connection with the operation of, any        *
*    computer system owned or operated by it.  In addition,          *
*    the recipient shall have the right to modify or merge           *
*    the software to form updated works.                             *
*                                                                    *
* 2. If the recipient receives a request from any third party        *
*    to furnish all or a portion of the software to any third        *
*    party,  it will refer such a request to Carnegie-Mellon         *
*    University.                                                     *
```

```
*                                                                     *
* 3. Carnegie-Mellon University shall not be held liable for any     *
*    damages resulting from the use or misuse of the software        *
*    provided by it.  Furthermore, Carnegie-Mellon University        *
*    remains without obligation to assist in its installation        *
*    or maintenance.                                                 *
*                                                                    *
* 4. The recipient agrees to acknowledge Carnegie-Mellon             *
*    University in appropriate citations appearing in public         *
*    literature when reference is made to the software provided      *
*    above.                                                          *
*                                                                    *
* 5. If the recipient develops any enhancements to the software      *
*    which materially improves its operation, the recipient          *
*    agrees to make such enhancements available to Carnegie-         *
*    Mellon University without charge, provided Carnegie-            *
*    Mellon University agrees in writing to receive such             *
*    enhancements in confidence, if requested to do so.              *
*                                                                    *
* 6. This header comment must remain attached to the source          *
*    code of the provided software.                                  *
*                                                                    *
*  Bug reports and suggestions can be mailed to Cem Unsal by         *
*  electronic mail addressed to: "unsal@ri.cmu.edu". As mentioned    *
*  in condition 3 above, the author is not obligated to fix any      *
*  such bugs, or even to acknowledge receipt of the bug report.      *
*                                                                    *
\********************************************************************/
```

10 Contact Information

The author of this document can be contacted at:

Cem Ünsal

Robotics Institute	(412) 268-5594
Carnegie Mellon University	(412) 268-5571 (fax)
5000 Forbes Avenue	unsal@ri.cmu.edu
Pittsburgh, PA 15213-3890	http://www.cs.cmu.edu/~unsal/

11 References

[1]The SHIFT Team, “SHIFT, the Hybrid System Simulation Programming Language,” California PATH/University of California-Berkeley, http://www.path.berkeley.edu/shift (September 19, 1997).

[2] The SHIFT Team, “California PATH Smart-AHS,” California PATH/University of California-Berkeley, http://www.path.berkeley.edu/smart-ahs (September 19, 1997).

[3] A. Deshpande, “AHS Components in SHIFT,” California PATH/University of California-Berkeley Report, http://www.path.berkeley.edu/shift/doc/ahs.ps.gz (September 19, 1997).

[4] C. Ünsal , “Sensor Models for AHS Simulations,” Technical Report CMU-RI-TR-98-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 1998, http://www.cs.cmu.edu/~unsal/research/shift/sensor.pdf.

[5] M. Antoniotti, and A. Deshpande, “SmartAHS 1.x User’s Manual,” California PATH/University of California-Berkeley, http://www.path.berkeley.edu/smart-ahs/smart-ahs-user-man.html (January 27, 1998).