SCENARIO DEFINITION AND CONTROL FOR THE NATIONAL ADVANCED DRIVING SIMULATOR

Yiannis Papelis, Omar Ahmad, and Matt Schikore
The University of Iowa, National Advanced Driving Simulator, USA
Paper Number: 346

ABSTRACT

The National Advanced Driving Simulator (NADS) is a high fidelity simulator developed by the National Highway Traffic Safety Administration (NHTSA) and located at the University of Iowa. The NADS is a tool that allows fundamental research into the complex interaction between driver, vehicle, and roadway. To facilitate that goal, the NADS has been designed as a shared-use facility whose operating model allows researchers from laboratories, academia, and industry to design and test simulation scenarios on consumer class personal computers before using them on the NADS. A set of software components were designed and implemented to allow this off-line specification of scenarios. These software components are cumulatively referred to as Scenario Definition and Control (SDC). This paper overviews the NADS SDC software and how it can be used for developing driving simulation scenarios through the use of the Interactive Scenario Authoring Tool (ISAT).

SCENARIO PREPARATION FRAMEWORK

Numerous items must be specified while preparing to conduct research on driving simulators. In general, the higher the fidelity of the simulator, the more issues there are that must be specified by the user. Aspects that typically must be specified include the synthetic environment, the amount and density of traffic, the exposures to be used for the various research subjects, and the nature of binary information that must be collected for analysis after data collection. To manage the amount and complexity of information that must be specified by the user, the NADS has been designed to utilize graphical interactive tools that allow researchers to perform the majority of the specification on consumer class personal computers. The advantages of this approach are many. Researchers can fine-tune their specification on their own time, the NADS does not need to be occupied while testing takes place, and many research projects can be under development concurrently. Of course, not every aspect of a research project can be defined this way, but the goal of the NADS tools is to maximize what researchers can do and minimize the time it takes to prepare a simulator configuration.

In order to describe the operation and capabilities of the SDC software, it is important to define a process within which the software is used. At the same time, it is difficult to pinpoint an example process that is simple and easy to comprehend yet covers all potential models of NADS usage. We believe the example provided here covers a significant subset of NADS usage, but it should not be considered a limit on how the NADS can be used to investigate various problems. Rather, it is an example that is useful in explaining the nature of the SDC software. For example, the process does not indicate the potential of instrumenting new cabs or integrating new in-vehicle devices to current NADS cabs, yet any of these activities is possible. Figure 1 illustrates the example process.

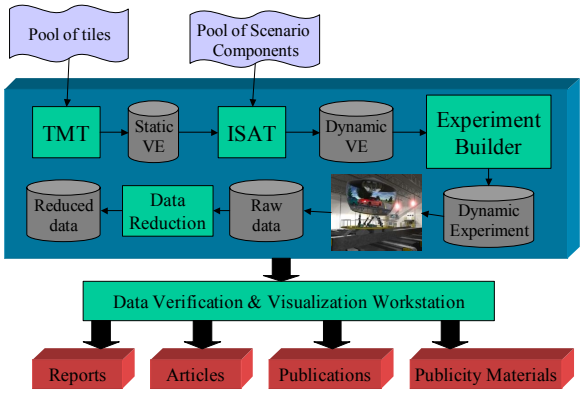

Figure 1 Example process.

The Tile Mosaic Tool (TMT) is a graphical tool that allows the creation of the road network that will be used in the simulator scenarios. The pool of tiles refers to a set of pre-fabricated components, each representing a small geographical area (e.g., a city block or a piece of road), that can be combined by the TMT into a larger virtual environment to be used in the NADS. The output of the TMT is called a *static virtual environment* because it contains the physical environment (e.g., the road network, buildings, terrain, features) but no active elements (e.g., traffic, pedestrians). The static virtual environment comprises several distinct but correlated databases, each of which is used by the various NADS subsystems while the NADS is running. For example, the visual representation of the virtual environment is displayed by the NADS Image Generator and

projectors, whereas the road network information is used by the various driver models populating the virtual environment. The ISAT depends on the existence of a static virtual environment on which to build scenarios. The pool of scenario components represents existing libraries of scenarios that can be used for building new scenarios. The ISAT produces a scenario file, which in combination with the static virtual environment represents a *dynamic virtual environment*. This information is fed to the experiment builder, a tool that allows the specification of the number of participants in a study and the conditions to which each participant will be exposed. For example, let us assume that the ISAT produces two scenarios, S1 and S2. If an upcoming study will use eight subjects, then each of the subjects can have any number of exposures using either S1 or S2 and the exposure conditions. The Experiment Builder allows the specification of such associations.

Upon completion of these associations, the cumulative set of various files and databases is transferred to the actual NADS computer systems where they can be used to execute the runs. After completion, the final output of the simulator is various raw data files that may include video, binary, or other data. Those are then fed to various post-processing tools (referred to as *data reduction*) and the resultant *reduced data* can be used by the researchers for statistical analysis. Generally, the ultimate goal of this process is the generation of various reports, articles, publications, or other publicity materials. This is facilitated through a Data Verification and Visualization Workstation (DVVW), which is a hardware-software custom-developed product that facilitates the integration, understanding, and dissemination of all available information

Note that no matter how complex the data that is produced in the various phases of the process, the goal of the data-handling tools is to hide this complexity from the user.

SCENARIO AUTHORING OVERVIEW

Scenario authoring involves the specification of both the static and dynamic elements of a virtual environment so that certain events appear to anyone who drives the simulator under the specific scenario independent of small differences in driving style or timing. As a simple example, consider testing a new in-vehicle warning device that provides various alarms to warn the driver of the potential of a collision. The researchers are interested in seeing how different warning signals perform as far as their ability to alert the driver and prevent a collision. To test such a device, one should be able to create potential collision situations. One way to do this is to have the subject drive along a road and through an intersection controlled by traffic lights while the light is green. As the driver nears the intersection, another car (car B), which is initially stopped by the red light on the other leg of the intersection, can begin moving, thus blocking the path of the subject. Let us consider the issues in building such a scenario.

By far the most critical aspect of such a scenario is timing the motion of car B so that it provides a consistent time-to-collision condition for all subjects, something that is necessary in evaluating the collision avoidance device. However, additional issues must be considered. When was car B created? Is car B the same make and model for all subjects, or does it change? How does one ensure that when the subject approaches the traffic light it is green as opposed to yellow or red? What if different subjects travel at different speeds toward the intersection?

One potential approach to building such a scenario in a way that addresses these concerns is to explicitly script everything that is dynamic about the virtual environment so that its timing is tied to the motion of the simulator driver. For example, time 0 is when the scenario starts, time 40 is when the driver reaches some fixed distance in front of the intersection, time 60 is when the driver's front bumper enters the intersection, and time 100 is when the driver reaches some point beyond the intersection. Scripting software can then determine the value of time based on how fast or slow the driver moves and then trigger every change in the virtual environment based on the parametric time scale. For example, the light will turn green at time 20, and car B will begin moving at time 40 and will reach the middle of the intersection at time 60. That way, everything is scripted and no matter how fast or slow the driver moves, the event will happen consistently.

Such techniques are often used in part task simulators that typically use relatively short scenarios or involve few traffic elements. In addition, the technique is simple, easy to understand, and provides deterministic results. However, when considering such a technique for use in high fidelity simulators, several problems arise. In high fidelity simulators, scenarios are longer and generally involve multiple entities. Coincidentally, similar problems can appear in part task simulators when using larger runs with scenarios requiring the coordination of multiple scenario elements. Coordinating one or two elements through explicit scripting for a 5-minute scenario is reasonable, but coordinating 500 vehicles for a 45-minute scenario is daunting and virtually impossible. Note that the 500 vehicles may not be active at the same time; they may represent traffic on the opposing lane that is not an integral part of the scenario.

Nevertheless, if scripting is the only tool available, it would have to be used for all vehicles. Another complication has to do with what happens after the event takes place. One may want to trigger different scenarios after the near-collision event, depending on the outcome of the first event. For example, if there was a near collision, a less severe event may need to take place on the next intersection, but if the subject veered away early enough, a more severe event may be necessary. Such decision-making cannot be programmed when scripting is the only tool.

An alternative approach is to use intelligent agents that populate the simulator's virtual environment and behave autonomously. For example, autonomous driver models can be used to control the various vehicles in the scene. An intelligent manager can control the traffic lights so their timing follows a specific pattern. Such an approach makes it easy to create scenarios involving multiple entities since the labor-intensive specification of the individual behavior of every entity is eliminated. In addition, the length of the scenario does not overly complicate the development of the scenario because autonomous entities can be created automatically throughout the scenario. A significant amount of work has focused on techniques for implementing such scenario control solutions [1,2,3].

However, other complexities surface. For example, timing and coordination become harder to achieve. Consider the example described earlier. If a traffic light manager controls the state of the traffic lights based on the pattern, there is no guarantee that when the simulator driver approaches the intersection the light will be green. Furthermore, if vehicles are autonomous, there is no guarantee that car B will be waiting on a red light since that car may have decided to pick a different route earlier. Even if a car is waiting at the red light, various subjects will travel at different speeds, thus reaching that point at different times so the proper time for car B to block the intersection is not known a-priori.

The approach used for scenario authoring in the NADS SDC software is a hybrid of the two techniques, with primary emphasis on using intelligent agents in conjunction with coordinators to ensure consistency [4]. Specifically, the SDC provides the user with the ability to easily create any number of entities that behave largely autonomously. In addition, the user can create coordinators that are *invisible* entities that exist in the virtual environment and whose responsibility is to orchestrate events by monitoring what happens and at key points modify the autonomous behavior of the remaining agents to achieve a pre-specified goal. Additional coordinators can be used to automate the generation of traffic, control the traffic lights, modify the environment conditions, and control numerous aspects of the simulator's operation. In addition, the ISAT allows the creation of purely scripted entities whose evolution is completely deterministic and whose timing is either independent or dependent on the simulator driver's actions.

Deterministic objects, or Deterministic Dynamic Objects (DDOs) as they are referred to in the SDC, are objects whose behavior is pre-scripted by the user. Specifically, the user can select a path and specify the velocity of the DDO at each point in the path. While the scenario is running, a DDO simply follows its path according to the user's specification.

Dependent DDOs, or DDDOs, are similar in that they follow a specific path, but their velocity adjusts so they reach a specific point of their path at the same time another entity reaches another target point. DDDOs allow vehicles that follow a scripted path to behave consistently as far as their relative position to another object (including the simulator driver), independent of the variation in the other object's speed.

Autonomous Dynamic Objects (ADOs) use a sophisticated autonomous driver model that controls their motion. One can think of an ADO as a human driver that is driving around the virtual environment trying to reach its destination. An ADO will follow the rules of the road, including following the speed limit and respecting traffic lights, and exhibit most behaviors exhibited by real drivers.

A unique feature of the driver model used in the NADS SDC, however, is the ability to have its default behavior modified at runtime. For example, an ADO can be told to follow a specific speed independent of the external conditions, or to change lanes, or the take a particular turn, etc. Such commands can be initiated by the researcher while the scenario is running through the ISAT or, most commonly, are issued by the various coordinators that can be authored a-priori. The actual paradigm used to represent the notion of commands in the SDC software is that of buttons and dials. Buttons and dials are conceptual models for inputs associated with each behavior type and operate much like physical push buttons and analog dials. Specifically, a button is an input mechanism that, when pressed, provides a one-time binary signal to the behavior. A dial is an input mechanism that, when set, provides a perpetual analog signal to the behavior. The actual effect of the delivered signal depends on the behavior itself and the state of the scenario element when the signal is received. For example, consider an imaginary button associated with ADOs whose name is “Turn Left.” Pressing the button has the effect of forcing the ADO to go left on the next intersection as opposed to the direction the ADO was originally going to take.

Note that buttons and dials provide a "suggestive" means of control, not an explicit one. This implies that a given behavior may decide to ignore a signal delivered though a button or dial if it cannot gracefully fulfill the request. For example, if the Turn Left button press came at a time when it was not possible to make a left turn, that request would not be fulfilled. At the same time, if a behavior defines that a given button or dial has a direct effect, the responsibility of reasonable behavior is passed to the user. Again, consider another example of a dial associated with an ADO named "ForceVelocity." If this dial is set, the ADO will travel at that velocity, no matter what. However, setting the speed to a fixed value could cause collisions or make the car run off the road.

The SDC software provides facilities that allow researchers to manually press buttons or set dials of any ADO in the simulation. However, this is not a reasonable approach for ensuring repeatable scenarios. Instead, the SDC contains additional virtual entities, called triggers, that can be programmed to perform such actions based on rules specified by the user.

The trigger is an entity that is given a series of conditions and a series of actions. The trigger continuously evaluates its conditions and when they are all true performs the actions. Some examples of conditions include another object reaching a specific point in the road network or a traffic light reaching a specific state (e.g., green). Various actions are available, including creating a new entity, deleting an existing entity, modifying the cycle of the traffic lights, issuing behavior modification commands to a set of autonomous entities through buttons and dials.

In addition to triggers, other high-level entities are included in the SDC software to ease certain tasks that are too labor intensive to perform manually.

The traffic manager is an entity that is tasked with generating autonomous traffic to populate the area around the driver. Because of a finite limit on the computational resources of the simulator, it is not possible to simulate an arbitrary high number of vehicles in real-time that populate the whole virtual environment. Therefore, the traffic manager creates traffic only in the vicinity of the driver.

A traffic source is another coordinator whose job is to create traffic. Unlike the traffic manager, however, the traffic source creates vehicles at a specific point in the database with a deterministic generation frequency.

Finally, the traffic light manager controls the state of traffic lights in the scene. The user can program the traffic light manager to achieve just about any necessary timing cycles. In addition, the traffic light manager, in conjunction with the triggers, allows the linking of traffic lights to achieve coordinated traffic light groups.

ISAT OVERVIEW

The ISAT is the primary front-end tool that allows researchers to interact with the SDC software. The ISAT utilizes a Graphical User Interface (GUI) that follows the typical GUI conventions of the Windows NT® and Windows 2000 ® operating systems. The ISAT uses a multiple document interface, where a document refers to a scenario, and allows the editing of any number of scenarios at the same time.

General tool layout

The ISAT uses toolbars to group similar function buttons together. Available menu and toolbar options when no document is loaded are a subset of all available ones. When initially loaded, the user can load an existing scenario file or create a new scenario file using the respective menu to tool bar button. Figure 2 illustrates how the ISAT looks after a document is loaded.

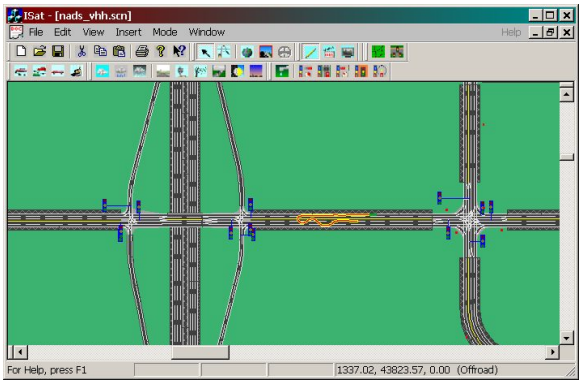

Figure 2 ISAT after loading of a scenario.

The tool is conceptually divided into three areas: the toolbar and menu area, the work area, and the status bar. The menu bar used in the ISAT is typical of most GUI programs. Various tool bars are available immediately below the menu. The ISAT toolbars are organized according to logical functions and can be "docked" at different places, including outside the main tool window. The work area is located below the toolbars and contains a top-down view of the synthetic environment. This view is focused on the features necessary for driving, i.e., it displays roads, lanes, lane markings, traffic lights, signs, etc., but does not display forests, buildings, or other visual features that do not affect driving. In addition, various features are displayed in iconic form as opposed to a geometrically correct rendering.

The status bar on the bottom of the window provides multiple dynamic messages that either reflect the potential effects of users' actions or provide detailed information about objects under the cursor.

Tool Operating Modes The actions that the user can perform when using the ISAT change depending on the tool operating mode. There are currently three operating modes, each focused on a different phase of the scenario development process. The three operating modes are authoring, rehearsal, and monitoring. The tool first starts in authoring mode. While in authoring mode, the user can create new scenario elements and modify existing elements. The rehearsal mode allows execution of the current scenario. Executing a scenario takes place by running the same scenario control software that would run on the simulator but on the PC on which the ISAT executes. While the scenario executes, the ISAT shows the evolution of the virtual environment on the main window. The monitoring mode is a special mode that can only be used when the computer running the ISAT is within the NADS computer network. While in this mode, the ISAT connects to the simulator's real-time system and obtains information about the position and state of all visible entities in the virtual environment and displays them in real time in the ISAT window.

There are numerous capabilities provided by the ISAT, however, a detailed description of these capabilities is beyond the scope of this article. The ISAT User's Guide [5] can be consulted for more information.

SCENARIO ELEMENTS IN SDC

Common Parameters

Developing NADS scenarios using the SDC software is achieved by using the ISAT to define a file containing all scenario elements such as ADOs, DDOs, and coordinators. A few key parameters can be associated with any scenario element and help coordinate the execution of scenario events. These parameters are the Creation Radius, the Activation Delay, the Lifetime, and the Scenario Object Library (SOL).

Creation Radius The creation radius is a threshold in the distance between the scenario element and the position of the simulator driver. At runtime, the element will not be created until the actual distance between the position of the simulator driver and the scenario element is less than or equal to that threshold. For example, consider a scenario element placed at position (0,0), a creation radius of 300, and an initial position for the simulator driver at location (500,0). When the scenario first starts, the actual distance will be 500, which is larger than 300, so the scenario element will not be created. What this means in the simulator is that there will be no visible entity at location (0,0). Furthermore, the computational load on the scenario caused by this element is minimal. Now consider the situation where the simulator driver is driving toward the location (0,0). The range between the scenario element and the driver is continuously monitored, and when that range gets below 300, the scenario element will be created. In the above example, this will take place when the driver reaches the location (300,0). The actual effect of the creation varies depending on the nature of the scenario element. For example, if the scenario element has a visual representation (it could be, for example, a stopped vehicle), then this is the time when that vehicle will appear in the simulator's virtual environment. The creation radius only affects when a scenario element is created, not when it is deleted.

The rationale for having a creation radius is twofold. First, it helps manage the computational load of the system by not creating scenario elements until near the time they will be used. Second, it can be used as a simple authoring mechanism where the rough timing of events is controlled by the motion of the simulator driver.

Activation Delay The activation delay is the number of seconds between the time a scenario element is created and the time its behavior is engaged. The default value for this parameter is 0, indicating that an element will begin acting as its behavior dictates immediately upon its creation. Note that upon creation, the visual representation of that element, if any, will appear in the simulator's virtual environment.

The activation delay is often useful in conjunction with a creation radius. Using a larger than needed creation radius can help minimize the chance that the simulator driver will observe an object popping out of nowhere, and the activation delay can ensure that the object does not begin its activity until the driver is closer. For example, consider a DDO vehicle that is programmed to begin traveling at 50 mph, has a creation radius of 500, and has an activation delay of 3. Once the driver gets within 500 feet of the DDO, the DDO will be created and the vehicle will appear in the virtual environment. However, another 3 seconds have to pass before the DDO will begin moving.

Lifetime The lifetime is the maximum number of seconds that a scenario element will exist. The time of existence begins when the scenario element is created. Once the lifetime is reached, the element is automatically deleted. Like the other parameters, the lifetime is useful in controlling the computational

load of a scenario. Having a finite lifetime ensures that a scenario element does not consume computational resources long after it has performed its task within the scenario.

SOL Model Most scenario elements have a visual representation associated with them. It is important to realize that in general, the visual representation is decoupled from the behavior of a particular object. Behavior refers to whether the object is a DDO, ADO, or static object. Visual representation refers to what the object looks like. The visual representation is selected among existing objects contained in the SOL, a library containing numerous objects, their visual representation, and additional properties such as their audio signature or terrain effects. Objects in the SOL are grouped in categories. Objects in the same category have similar properties and are conceptually similar. For example, all objects of category "Truck" have a parameter "weight"; however, each object can have its own value for that parameter.

Deterministic Dynamic Objects

Deterministic Dynamic Objects (DDOs) are objects with no autonomous behavior that follow a pre-scripted path in the virtual environment. DDOs come in two flavors, regular and dependent. The key difference is in how the velocity of the object is controlled while it follows the pre-scripted trajectory. In regular DDOs, the user specifies the velocity of the object at control points defining the trajectory, whereas in a dependent DDO the velocity is controlled relative to the closure rate of another object towards a target point.

The trajectory of a DDO is described by a series of user-specified control points. During scenario execution, the path of the DDO follows a geometric spline that visits all control points in order. Using a geometric spline has the effect of smoothing the motion of the DDO while it is crossing the control points.

The velocity of regular DDOs is explicitly specified by the user at each control point. The DDO is guaranteed to be traveling at the specified speed when crossing the control point. Interpolation is used for determining the speed between control points.

Because of the low computational cost, DDOs are convenient for manually creating large amounts of ambient traffic when that traffic is not near the driver and is not going to interact with the driver. For example, they can be used as traffic going over a bridge when the simulator driver is driving underneath the bridge, or vice versa. Also, in certain cases, DDOs can be used for sparse oncoming traffic, especially in divided highways. Since DDOs exhibit no reaction to the simulator driver or other ADOs, it is not recommended that they be used among other autonomous traffic elements.

DDOs can also be used for animating pedestrians on sidewalks. By enabling the creation radius parameter, DDOs can "come alive" around building corners just as the driver approaches an intersection. By using the ISAT to fine-tune their timing and trajectories, numerous pedestrians can be added to a scene with minimal effort.

Autonomous Dynamic Objects

The ADO behavior is derived from a sophisticated autonomous driver model [6] that exhibits several driving behaviors similar to human drivers. Once an ADO is placed in a scenario, it will travel along a random path in the road network while following all the rules of the road and interacting with other ADOs and the simulator driver. The ADO contains an extensive set of dials that allows the orchestration of complicated events, despite the highly autonomous nature of its behavior.

An ADO can be initialized as a random navigator or given a specific path to follow. When initialized as a random navigator, the ADO builds a path by using its start position as the beginning and adding intersections and roads. The direction taken when crossing an intersection is selected at random. When initialized with a pre-specified path, the ADO will follow that path and, once it reaches the path's end, will become a random navigator.

The ADO will use its turn signals as necessary when performing lane changes and turns. In addition, the brake lights will turn on when an ADO decelerates at a rate that would require the use of brakes. At this point, ADOs will not use their horns, although they can be forced to produce sound effects through their dials.

An ADO can be associated with various SOL objects, each of which have different physical properties including dimension or engine characteristics. In order to provide realistic movement when turning, braking, and accelerating, a multibody dynamic model is used to simulate the motion of the vehicle. Use of a realistic dynamic model also implies that objects controlled by an ADO cannot have supernatural performance, unlike DDOs whose speed is computed kinematically and will achieve any performance specified by the user.

The formalism used for modeling the ADO's behavior is designed to accommodate concurrent implementation of goal seeking. In effect, within the model, all behaviors are active at the same time, but only the most important ones are used for the actual guiding of the vehicle. This allows the ADO to

seamlessly react to new circumstances by exhibiting the behaviors as dictated by the current conditions. Currently, the behaviors included in this model include Following, Lane Tracking, Free Drive Speed Control, Lane Change, and Intersection Navigation.

Following The algorithm currently used by ADOs for following other vehicles is a simple controller that maintains some distance behind a lead vehicle. The actual value of this distance varies between ADOs or can vary from time to time within an ADO through a randomization function utilized to provide variation in traffic. Input parameters can also be used to modify the actual value of the target following distance. In addition, it can be overridden through buttons and dials.

The actual controller responsible for maintaining the distance is tuned fairly aggressively so that vehicles will react quickly when their desired distance is disturbed. This is to ensure that vehicles don't collide with each other or with the simulator driver. However, it is still possible to have collisions if vehicles are forced to drastically change their speed or if their path is obstructed in a way that makes stopping in time impossible. At the same time, the aggressiveness of the controller often leads to low-frequency oscillations when the speed of a car within traffic changes for whatever reason. To some degree, this oscillation is similar to spring-like oscillations observed in real traffic patterns.

Lane Tracking The lane tracking behavior is responsible for steering control. The ADO will track the current lane using a simple steering controller that is tuned to minimize lane incursions during sharp turns. In general, the ADO will track the center of the lane plus or minus a small random perturbation for variability. The controller attempts to keep the whole vehicle body within lane boundaries at all times.

Speed Control The speed at which an ADO travels through the network when not following another vehicle is controlled by many factors, including the type of road, the posted speed limit, the road's curvature, and external commands through buttons and dials. In general, if no speed limit is posted, the ADO first computes an estimate of the current speed limit based on the type of road. If a speed limit sign is passed, the ADO will obey the speed limit and remember it until the next intersection. Finally, the ADO will use the curvature of the road to compute an upper bound on its velocity. The upper bound is computed by calculating the maximum speed around the curve that would expose the vehicle to no more than a threshold lateral acceleration. The actual threshold varies depending on the input parameters, but in general, it cannot exceed the traction limit of the dynamic model. A typical value is 0.25 Gs.

Once all speeds have been computed (i.e., road default, speed limit, curvature), the lowest is used to guide the vehicle after taking into account the setting of the TargetVelocity Dial. Figure 3 illustrates the block diagram used for calculating speed control. The illustration is focused on the Free Driving Speed Control logic, but does include the logic used to combine velocity control signals from other behaviors and external inputs.

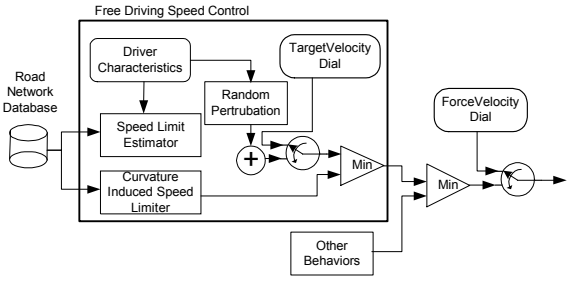

Figure 3 Diagram used to calculate speed control.

Lane Change The ADO looks for opportunities to perform lane change maneuvers. Currently, an ADO changes lanes for various reasons. It changes lanes to stay on its path. It changes lanes to the left to if it encounters a slower vehicle in front of it. An ADO changes lanes to the right as a means of traveling on the rightmost lane of multilane roads when no other conditions necessitate traveling on a lane other than the rightmost. When on a highway, an ADO will change lanes to the left to yield to merging traffic. An ADO will change lanes in response to advisory traffic signs warning of an upcoming lane closing. Finally, an ADO changes lanes due to an external command from the user. This command forces the ADO to change lanes regardless of its current path.

An ADO is relatively intelligent about how it performs lane changes. For example, if an ADO needs to turn right at the next intersection and there is a slow-moving vehicle in front of it, it will only change lanes to pass if there is enough distance before the intersection to move back into the right lane and make the right turn at the intersection. In addition, an appropriate gap must exist between vehicles on the target lane before a lane change maneuver is initiated. The lane change behavior acts dynamically; i.e., it continuously re-evaluates the current situation, and if conditions change enough, a lane change maneuver can be aborted.

Intersection Navigation The intersection navigation logic used by the ADO is designed to accommodate most types of intersections encountered in actual road networks. The general principle used for intersection navigation is to first

identify the intersection corridor that will be used to traverse the intersection and then find all vehicles whose corridors intersect the ADO's corridors. Having found the other conflicting vehicles, the ADO uses the rules of the road, including any traffic signs or lights, to prioritize all vehicles. At any given time, all vehicles other than the one with the highest priority will stop at the designated hold offset, typically indicated by a solid white line. Once the highest priority vehicle has cleared the part of the corridor that intersects the other corridors, the vehicle that was second in priority becomes the highest priority and proceeds.

It often happens that according to the rules of the road, two vehicles have similar priority; i.e., there is no clear differentiation on which one should go first. When this happens, a random selection is made among the vehicles. This coordination requires communication between vehicles to ensure that all have a consistent view of who has the right of way. The problem is that given N ADOs trying to negotiate an intersection, this approach requires N^2 messages. To avoid this explosion in communication, the intersection navigation logic uses a single controller implementing the same decision-making process to explicitly assign priorities to all ADOs. The single controller approach reduces the messages to 2*N, (N messages for the ADOs to communicate their path and another N messages for the controller to send back their priorities), thus minimizing the need for vehicles to communicate with each other. Although this is a centralized approach, the algorithm used still depends on simulating a distributed decision-making typical of real life.

Another challenging complication in intersection navigation is integrating the simulator driver in the traffic. Whereas ADOs can coordinate with each other by declaring their intended path and obeying the controller's priority assignment, the simulator driver does neither. To address this problem, there is a specialized coordinator, called the Driver Mirror, whose responsibility is to look at simulator driver placement and turn signals in order to predict the driver's path, which then is used by the remaining traffic to determine priorities. For priority assignment, because there is no way to tell the driver what to do, the ADOs have explicit logic to deal with the simulator driver. This involves computing the priority of the driver using a method similar to other ADOs, but then acting somewhat different:

- When the driver is the highest priority vehicle, ADOs yield to it as usual.
- When the driver has equal priority as other ADOs, the driver always wins the toss.
- When the driver has lower priority than other ADOs, the highest priority ADO proceeds as usual but monitors the driver to detect motion; if motion is detected, priorities are re-evaluated using the updated placement of the vehicles.

The potential for deadlocks always exists, and the central controller can detect deadlocks and resolve them. However, to ensure that there are no deadlocks between the scenario cars and the driver, as in the case where ADOs think the driver is the highest priority but the driver thinks that another ADO should proceed, the ADOs will use a time-out value for how long to wait. If the time-out expires, the ADOs get new priorities that have the driver at a lower priority level.

To ensure that there are no collisions if the driver decides to move after an ADO has begun crossing the intersection, in addition to monitoring the driver's motion, each ADO uses a forward-looking cone to detect vehicles immediately ahead of it. If the driver appears in that cone, the vehicle stops until the cone area clears.

The intersection navigation logic runs concurrently with other behaviors, and if other behaviors dictate a slower speed, then that speed is selected. That way, when multiple vehicles are queued on an intersection, the follow logic of all vehicles but the first would override any intersection navigation commands, thus stopping the vehicle behind the queue leader.

Buttons & Dials In addition to the default autonomous behaviors, the ADOs support these buttons & dials to help coordinate scenarios:

ChangeLaneLeft – Causes the ADO to change lanes to the left.

ChangeLaneRight – Causes the ADO to change lanes to the right.

TurnLeft – Causes the ADO to turn to the leftmost road at the next intersection.

TurnRight – Causes the ADO to turn to the rightmost road at the next intersection.

TargetVelocity – Causes the ADO to try and achieve this velocity whenever possible. The ADO ignores speed limits but does not ignore road curvatures.

ForcedVelocity – Causes the ADO to blindly achieve this velocity as quickly as possible. The ADO ignores speed limits and road curvatures.

AudioState – When set, the ADO continuously generates the specified sound(s). The sound will only be audible on the NADS.

VisualState – When set, the ADO continuously activates the specified light(s).

The Traffic Manager

The Traffic Manager (TM) is responsible for generating ambient traffic around the simulator driver. In this context, ambient traffic consists of one or more ADOs that are created and deleted by the TM at runtime to ensure that a pre-specified amount of traffic is surrounding the simulator driver at all times. The composition of traffic, the traffic density, and other parameters are grouped into Traffic Manager Input Sets (TMIS). To provide maximum flexibility, multiple TMIS can be specified and the active one can be dynamically selected at runtime through triggers. This allows different ambient traffic properties at different places of the simulation.

Conceptually, the operation of the TM is rather simple. At runtime, the TM counts the number of vehicles around the driver and computes the traffic density. If the density is lower than specified, the TM creates a few ADOs to increase the density. If the density is higher than specified, the TM deletes a few ADOs to lower the density. If the density is roughly equal to the value specified, no action is performed.

In practice, the actual operation of the TM is somewhat more complicated than described above. This complication is caused by two factors, namely the need to hide the creation or deletion of ADOs from the simulator driver and the limited number of ADOs that can be simulated at each time step due to the computational load on the scenario control processor. Both factors necessitate a more complicated algorithm that uses additional user-specified parameters whose purpose is to help minimize the computational load, minimize visibility of creations and deletions, and maximize the observability of any traffic elements by the simulator driver once they are created.

Following are the parameters specified in conjunction with the traffic manager. Creation Distance indicates how far from the driver new ADOs are created. Deletion Distance is a threshold distance that, once crossed by an ADO, would trigger its deletion. The desired density is a range between the Minimum Density and the Maximum Density parameters. To minimize load on the system, the TM runs at a regular interval, whose duration is specified by the Run Frequency parameter. Finally, to avoid overloading the scenario processor, the user can specify an absolute limit on the number of ADOs produced by the traffic manager.

The Traffic Light Manager

The Traffic Light Manager (TLM) is responsible for controlling the sequencing of traffic lights in the virtual environment. Note that by default, traffic lights are set to the off state. The TLM must be activated and timing should be defined for all intersections whose traffic lights need to be operating during the simulation.

Timing is defined for groups of related traffic. These groups are called Coordinated Light Groups (CLGs). Currently the grouping of lights into CLGs is done automatically by the software that generates the road network and cannot be modified by the user. In general, all traffic lights on an intersection are grouped together in a CLG.

Within each CLG, timing is specified by providing an arbitrary number of states and then specifying the color of each traffic light during that state. The duration of each state is also specified. At runtime, the TLM implements a simple state machine that visits all states in the order they were specified. In general, the TLM utilizes a small, but not negligible, part of the computing capacity of the scenario computer. However, when the number of traffic lights is very large, the I/O time necessary to communicate the state of each traffic light to the IG can get quite large. As a result, the TLM provides the ability to load balance the system by only operating traffic lights that are within a certain range of the simulator driver. Use of this load balancing is almost mandatory when a scene contains more than one hundred traffic lights.

Triggers

Triggers are flexible coordinators that allow the development of complicated scenarios. Simply put, triggers are predicate-action pairs. The predicates dictate conditions that, once satisfied, cause the actions to take place. At runtime, triggers continuously evaluate their predicate conditions and, if satisfied, perform the actions.

There are several types of triggers. Each type is characterized by the type of conditions that can cause it to fire, but the types of actions that can be performed upon firing are consistent across all trigger types. A variety of parameters associated with a trigger provide increased flexibility in creating scenarios. These parameters include a fire delay duration and a de-bounce duration. The fire delay parameter introduces a delay between the satisfaction of the trigger predicates and the actual firing. The de-bounce duration limits the minimum amount of elapsed time between successive trigger firings. The following are the types of triggers available in SDC.

Global Time Trigger This trigger fires after a fixed amount of time has elapsed since the beginning of the simulation.

Road Pad Trigger This trigger fires when scenario elements travel over a specified section of

the road network. It can be programmed to filter the elements that can trigger it by specifying a list of names, a list of SOL categories, and similar criteria.

Time To Arrival Trigger This trigger fires when a moving scenario element is within a certain time of arriving at a target point. The primary goal of this trigger is to allow authoring of near-collision events where the time to collision must be precisely controllable independently of the driver's actions.

Traffic Light Trigger This trigger fires when a traffic light reaches a specified state. For example, the trigger can be programmed to fire when a traffic light becomes red.

Geometry Position Trigger This trigger fires when any traffic element enters an arbitrary geometrical region. Conceptually, this trigger is similar to the road pad trigger, but the trigger polygonal region can be on or off the road network. This allows elements that move outside the road network, such as train carts and pedestrians, to also trigger scenario events.

Trigger Actions

The flexibility of building scenarios in the SDC is largely controlled by the capabilities of triggers. All trigger types have the ability to perform any number of the following actions upon firing:

- Create a new scenario element
- Delete an existing scenario element
- Set the dial or press the button of a scenario element
- Modify the traffic manager parameters
- Play an audio special effect
- Cause a simulated failure to the NADS vehicle subsystems (i.e., tire failures, brake failures, engine noises etc.)
- Force a coordinated traffic light group into a specific state
- Log time-stamped data in the NADS collection file
- Terminate the simulation
- Pre-position the motion base to optimize performance when the upcoming maneuver is known
- Adjust the tuning parameters of the motion system to better match the upcoming road

Additional actions can easily be incorporated in the system. An example of an action that is currently integrated into the system is the ability of triggers to initiate phone calls to an arbitrary number. This capability requires some minimal hardware to be interfaced with the simulator. The goal of this trigger action is to help investigate the effect of various types of cell-phones on driver distraction.

CONCLUSION

This paper provided an overview of the SDC software for the NADS. The goal of the SDC software is to provide a powerful environment for creating realistic yet repeatable scenarios that facilitate the conduct of research at the NADS.

REFERENCES

[1] P. C. Van Wolffelaar, W. Van Winsum, "Traffic Modeling and Driving Simulation – An Integrated Approach," *Proceedings of the Driving Simulation Conference, 1995*, Sophi Antipolis, France, September 12-13, 1995, pp. 236-244.

[2] J. G. Kuhl, D. Evans, Y. Papelis, R. Romano, G. Watson, "The Iowa Driving Simulator: An Immersive Environment for Driving-Related Research and Development," *IEEE Computer* 28(7), pp. 35-41.

[3] M. H. Strobl, J. H. Bernasch, J. P. Lowenau, "Generation of Complex traffic Scenarios in the BMW Driving Simulator," *Proceedings of the Driving Simulation Conference, 2000*, Paris, France, September 6-8, 2000, pp.245-256.

[4] J. Cremer, J. Kearney, Y. Papelis, "HCSM: A Framework for Behavior and Scenario Control in Virtual Environments," *ACM Trans. Modeling Comp. Simul*. 5(3), pp. 242-267.

[5] National Advanced Driving Simulator, *ISAT User's Guide and Functional Reference* (NADS Report N01-004), Iowa City, IA: NADS, 2001.

[6] Y. Papelis, O. Ahmad, "A Comprehensive Microscopic Autonomous Driver Model for Use in High-Fidelity Driving Simulation Environments," *Proceedings of the Annual Transportation Research Board Meeting*, Washington, DC: TRB, 2001 (to appear).