Proceedings of the AAAI Sprint Symposium Series on Knowledge Representation and Ontology for Autonomous Systems, Palo Alto, CA, March 22-24, 2004.

Knowledge Representation for On-Road Driving

Chris Scrapper and Stephen Balakirsky
National Institute of Standards and Technology
Gaithersburg, MD 20899
{scrapper, stephen}@nist.gov

Abstract

This paper will present a knowledge layer used by a hierarchical on-road driving planner that represents a road network as a discrete set of attributed road states. This knowledge layer facilitates the construction of a planning graph by providing simulation and prediction services to the planning system. These services allow the determination of possible spatial transitions along a road network that a vehicle may take from its current location given its current state.

I. Introduction

In the mid 90's, a vision based machine-learning system known as RALPH controlled the lateral movements of a vehicle over the majority of the highways across United States of America (Pomerleau and Jochem 1994). This demonstration targeted a highly structured driving environment in the nation's highway systems. (Thorpe et al. 1991) comments that the development of reliable algorithms for autonomous driving requires an environment that is unstructured and realistic. The development of a truly autonomous agent capable of handling partial observable stochastic environments that contain multiple agents operating in proximity with each other will require innovative ideas.

Deliberative planning systems attempt to create an agent function for goal-based autonomous vehicles. This function attempts to map the percepts from the vehicle's sensors to possible actions the vehicle can take. (Russell and Norvig 2003) discuss deficiencies of table-driven agent functions, i.e., finite state machines or look-up tables. The authors claim that using a table driven approach to solving such a problem is unrealistic and hypothesize the total number of entries in such a table can be estimated by

$$\sum_{t=1}^{T} \left| P \right|^{t},$$

where T = total number of percepts received by the agent, and P = the set of possible percepts.

In order to deal with a countably infinite sequence of percepts in stochastic environments to generate appropriate behaviors, some deliberative planning systems use planning graphs that determine the optimal path through cost analysis. (Guo, Qu, and Wang 2003) discusses a motion planner that attempts to tackle the global trajectory-planning problem but has shown to be unable to handle the computational complexities in real time. (Balakirsky 2003) has developed a real-time deliberative planning system that has shown positive results in real time. This approach is based on an A* graph search algorithm that attempts to limit the size of the graph through the application of knowledge to the graph creation process.

An on-road driving planner such as (Balakirsky 2003) requires a knowledge layer that is capable of extracting knowledge from the a priori world knowledge, imperfect sensor knowledge and situational awareness. This paper will discuss a mini-expert system that is implemented to provide the planning system with the appropriate knowledge in real time. The layout of the rest of this paper is as follows. Section II contains a brief synopsis of the Real-time Control System reference architecture, *RCS*. In Section III, the on-road driving database that decomposes a road network into useful chunks of information that are easily accessible will be discussed. Section IV will discuss the incremental graph approach and sample results. In Section V, the knowledge layer used to support the planning system is discussed. Finally, Section VI will conclude the paper with a discussion of the expansion of the knowledge layer used by the on-road planning system.

II. Reference Model Architecture

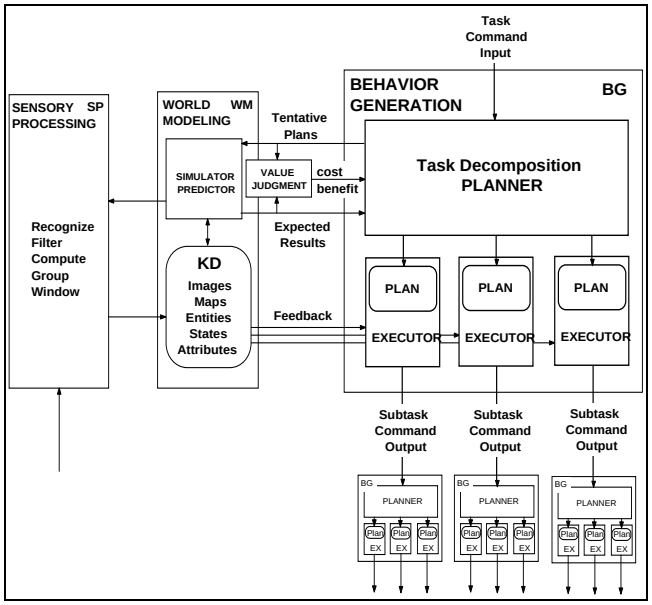

Figure 1. RCS computational Node (Albus and Meystel 2001)

In order to guarantee real-time operation and decompose the problem into manageable pieces, it is necessary to embed the planning framework into a hierarchical architecture that was specifically designed to accommodate real-time deliberative systems. The Real-Time Control System (RCS) reference model architecture is a hierarchical, distributed, real-time control system architecture that meets this need while providing clear interfaces and roles for a variety of functional elements (Albus 1991) .

Under RCS, each level of the hierarchy is composed of the same basic building blocks illustrated in Figure 1 (Albus and Meystel 2001). These building blocks include behavior generation, *BG*, (task decomposition and control), sensory processing, *SP*, (filtering, detection, recognition, grouping), world modeling, *WM*, (knowledge storage, retrieval, and prediction), and value judgment, *VJ*, (cost/benefit computation, goal priority).

This implementation of the WM is adapted from (Albus and Meystel 2001) to contain multiple knowledge layers within the WM, e.g. moving-object prediction, road state generator, maps, etc. The knowledge layer discussed in this paper is a mini-expert system that lies at the heart of the planning system for on-road driving, known as the Node Generator (NG). As shown in Figure 1, it is composed of two parts: a simulator/predictor and a knowledge database, *KD*. The NG maintains a knowledge base, provides state prediction, supplies data to planners and executors, and simulates spatial/temporal transitions within the road network. The simulated transition in the road network is evaluated by Value Judgment (VJ) to construct and analyze an incrementally created graph in order to produce the appropriate behavior for navigating road networks (Albus et al. 2001).

III. Planner

The planning system presented in this paper is designed to fill the roles of behavior generation, world modeling, and value judgment for a single level of the RCS hierarchy. The system receives a set of intersections that must be traversed, and one or more final goal locations from its supervising level. The system then refines this plan for specific lane locations and vehicle velocities while taking into account dynamic and static objects as well as user objectives and constraints (all of which must be in the knowledge representation of the system). The results of this plan refinement are then passed to the next lower level of the hierarchy for further refinement and execution. This process is periodically repeated for systems that include uncertainty in the prediction of moving object locations and the success of task execution.

1. Planning Framework

The incremental graph-planning framework is based on a combination of traditional graph-based planning techniques (Nilsson 1998; Russell and Norvig 2003) and logic-based planning techniques (Blum and Furst 1997). A detailed description of the planning system may be found in (Balakirsky 2003). The main difference between the incremental graph planning framework and traditional graph planning approaches is in the way states are mapped to nodes and the way that the nodes are connected.

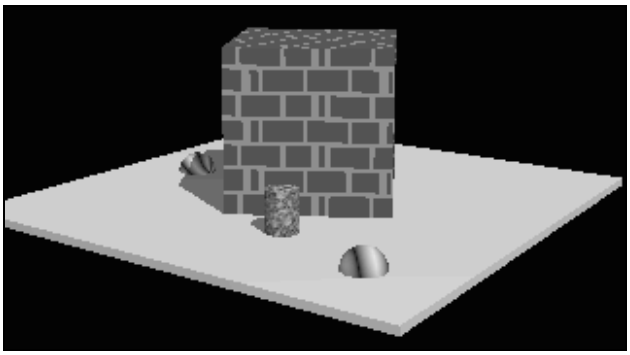

Figure 2. Representative objects from World Model

i. Attributed Graph Nodes

Figure 2 depicts several static objects that may be in an autonomous driving planning system's world model (a brick wall and an empty soda can) and the start and end points of a potential plan (the two spheres). In addition to knowledge about an object's location, the world model tracks all of the object's attributes that are necessary for making a cost/benefit decision that involves the object. For the above case of autonomous driving, this decision is about whether to drive over or around the object, and if over the object, at what speed. It should be noted that combinations of attributes are necessary for making this determination. Sample attributes that appear in the world model include such information as the corners of an object's bounding box, the object density, and the object's intrinsic value.

Having a varied set of object attributes allows the system to make intelligent trade-offs in its plans. For example, the system should run over a soda can in order to avoid driving into a brick wall. However, the system should be willing to incur substantial damage (i.e., hit the brick wall) rather than running over a small child (which would not damage the vehicle).

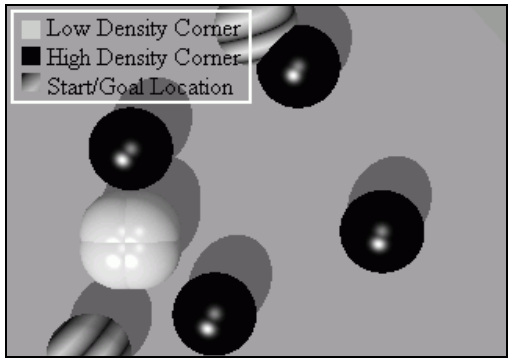

Figure 3. Overhead view of nodes derived from objects in Fig. 2

In order for the planning system to utilize these attributes as decision points, a node in the planning space must represent their location. As shown in Figure 3, these nodes may be densely or sparsely populated, and there are no "wasted" nodes placed in locations that are not relevant to planning decisions. In addition, the existence of these nodes may depend on the particular planning application. For example, if path planning for a large all-terrain vehicle, the low density corner nodes depicted in Figure 3 (the white nodes) would not be included in the planning graph. Since nodes are only instantiated where relevant attributes exist, the set of all nodes available for use by the planning system is referred to as the set of Relevant Annotated Nodes (RAND).

Relevant nodes must exist wherever a planning decision point exists. In the above example, there is no time dimension. The placement of nodes must simply allow the system to decide whether to go over or around static objects.

ii. Logical Arc Connections

Once the set of potential graph nodes has been established, a technique for creating a specific node's spanning set must be developed. The graph arcs that are utilized by the incremental framework build upon concepts developed by Blum and Furst in their GraphPlan work (Blum and Furst 1997). Blum and Furst have created a logical planning graph that expands levels by examining the results of the execution of all valid actions.

For the purposes of the incremental graph approach, the intersection of the result of applying a node's valid actions and the set RAND will be considered as the spanning set of the current node. For the static example presented above, the only valid action is for the vehicle to drive in a straight line (in any direction). By constructing a visibility graph from the current node to all other nodes, the node's spanning set may be found.

In other planning domains, time is an important factor and decisions must be made that match a pre-specified control cycle. It should be noted that for these cases the spanning set of a node is time-dependent, and therefore the graph topology may change on a planning cycle by planning cycle basis. In addition, the spanning set for a particular node at time t may not be equal to the spanning set for node at time t+1.

This is true for the case of on-road driving at the vehicle level of the hierarchy, where a decision on vehicle velocity and lane control (which lane to occupy) must be made every second. The planning space is now four-dimensional (x, y, time, velocity) and graph constraints through the use of valid actions and attribute relevance becomes even more important. For the over-simplified case shown in Figure 4, the valid actions that may occur are accelerate (A), decelerate (D), maintain velocity (M), and change lane. It is also possible to simultaneously change lane and velocity. The full set of valid actions is shown as the white and gray circles in the figure. However, the size of the spanning set will be constrained by membership in the set RAND. In this case, only the white circles are legal lane positions and members of RAND. Therefore, the spanning set will consist of the white circles.

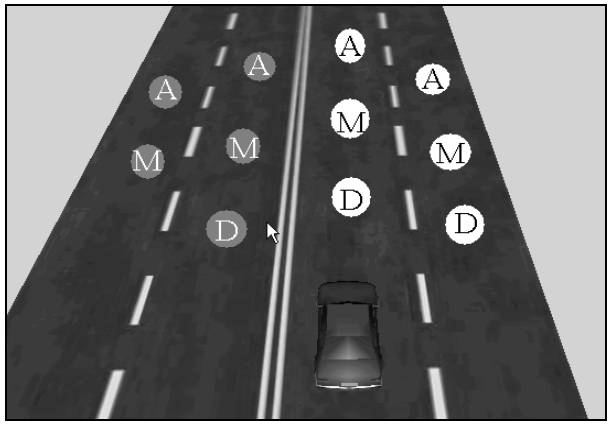

Figure 4. Spanning set of road states for in-road planning system

This technique of excluding possible actions from a node's spanning set is used to implement hard constraints on the plan. Since no graph nodes exist in the oncoming traffic's lane, it is impossible for the system to plan to enter this lane. It should be noted that if a planning failure occurs (no path below a certain cost threshold exists), then the constraints may be relaxed and the set RAND expanded to include all of the circles. The system must then use soft constraints in the form of the system cost function in order to discourage the vehicle from driving in the oncoming traffic's lane.

The focus of the remainder of this paper is the generation of the nodes that are members of RAND. The techniques that are utilized to generate the nodes and the knowledge representation of the underlying data will be examined.

IV. On-Road Driving Database

A knowledge base that provides concise, accurate information about the environment and the objects that reside in the environment is essential for the successful operation of any deliberative autonomous vehicle. For on-road driving, this knowledge must include the precise location of where the road lies, what type of marking or barriers are present along the roadway, the presence of traffic signs or signals, etc. (Schlenoff et al. 2004) have developed an on-road driving database that accurately conveys the appropriate information about road networks for the various fidelities of planning systems that are used in the RCS hierarchy.

The structure and features of the road network are captured in a six level decomposition hierarchy. This six level hierarchy was chosen for the on-road driving database to allow for the efficient representation of the road network at various levels of abstraction and to allow the knowledge to be more easily accessible to the different levels of BG within the system. Each level of the hierarchy encapsulates the minimum set of attributes needed to

derive the appropriate knowledge about road networks for behavior generation at that particular level of abstraction.

1. Database Definitions

The definitions in the on-road database that classify where two paths converge are:

Level 1: *Junctions* –two or more paths that converge or diverge, or a controlled point in a roadway.
Level 2: *Intersections* - type of junction in which two or more separate roads come together.

The levels of the decomposition hierarchy are defined as:

Level 1: *Road* – bi-directional stretch of travel lanes bounded by its proper name
Level 2: *Road Segment* - uni-directional stretch of a road bounded by intersections
Level 3: *Road Element* - stretch of a road segment bounded by any type of junction
Level 4: *Lane Cluster* – stretch of a road element bounded by change in attributes or road feature
Level 5: *Lane* - single pathway of travel in a lane cluster that is bounded by lane markings
Level 6: *Generic Lane Segment* – piece of a lane that consists of constant curvature arcs
- *Lane Segments* – lane segments that traverse road segments
- *Intersections Lane Segments* – lane segments that traverse intersections

As mentioned previously, the on-road driving BG system contains a planner at each level of the RCS hierarchy. The RCS architecture is design to uniformly distribute the computational complexities over the levels of the hierarchy by creating a hierarchy where each level is responsible for smaller planning horizons at higher plan fidelities than their supervisor. This architecture is mirrored in the on-road driving database to satisfy the different knowledge requirements needed at each level of the RCS hierarchy. Below is one implementation of an on-road planning system under the RCS architecture. The planning system discussed in the previous section is referred to as the Vehicle Level Planner, which is mapped into the Drive Behavior Planner and Elemental Maneuver Planner.

Level 1: *Destination Planner* – Coarse plan consisting of roads and intersections
- Planning horizon: 1-2 hours and up to 100 km
- Knowledge Requirements: Roads, Road Segments, Intersections

Level 2: *Route Segment Planner* – Negotiating intersection and road segments
- Planning horizon: 10 minutes and up to 10km
- Knowledge Requirements: Road Segments, Road Elements, Intersections

Level 3: *Drive Behavior Planner* – Low-level behaviors and lane changes
- Planning horizon: 100 seconds and up to 500 meters
- Knowledge Requirements: Lane Clusters, Lane, Intersection

Level 4: *Elemental Maneuver Planner* – real-time maneuver, i.e. acceleration
- Planning horizon: 10 seconds and up to 50 meters
- Knowledge Requirements: Lanes, Lane Segments

Level 5: *Goal Path Trajectory Planner* – trajectory within lane segments
- Planning horizon: 1 second and up to 5 meters
- Knowledge Requirements: Lane Segments

For further details concerning the on-road driving database refer to (Schlenoff et al. 2004).

V. Road State Generator

As mentioned in Section II, the Node Generator is one knowledge layer in the World Model of an RCS computational node that is composed of a simulator/predictor and a knowledge database. The implementation of the NG discussed in this paper is for the on-road driving knowledge layer at the vehicle level of the RCS architecture and is responsible for (Albus et al. 2001):

1. Creation and maintenance of on-road driving knowledge base
2. State elaboration and knowledge extraction
3. Constrained simulation and state prediction

Road Networks present a continuous, complex, unstructured environment containing static and dynamic features. To decrease the computational complexities of planning in a continuous domain, the NG maps the continuous environment into a spanning set of discrete uniformly spaced attributed nodes known as road states. The NG uses these road states and the vehicle's current state to predict plausible road states, known as goal states, and creates simulated temporal-spatial transitions of the vehicle along a road network, known as trajectories. The set of goal states and trajectories defines a reachability graph that is used by the planner to find a cost optimal path through the road networks.

1. Knowledge Database

The knowledge database consists of a data structure that stores a priori knowledge of the road networks as well as in situ knowledge received from the sensory data. The data structure used in this knowledge layer of the WM is a composition of road objects received from the on-road driving database discussed in Section IV. The data structure, which is implemented as a connected graph of adjacent lane segments, is depicted in the UML logical

model in Figure 5. The road primitives are generalized in polymorphic structures in order to facilitate the extendibility and durability of the data structure.

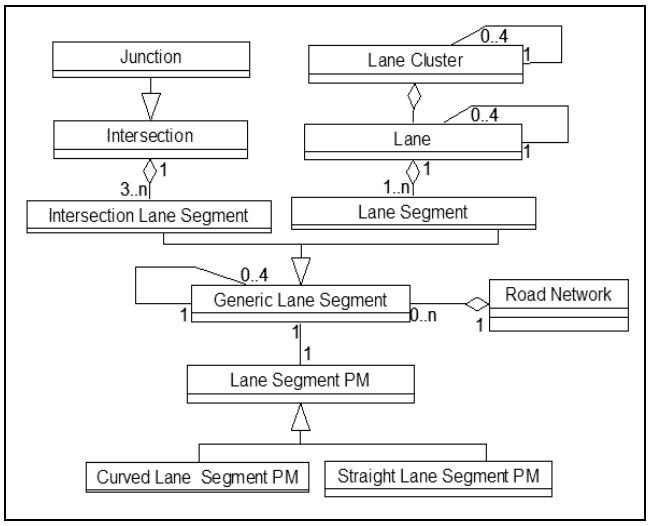

Figure 5. UML logical model of knowledge database

2. State Elaboration and Knowledge Extraction

As mentioned previously, road states are discrete uniformly spaced nodes that carry with them road feature attributes (e.g., lane markings, speed limit). Each road state is uniquely identifiable by a tuple that includes the road object id and node id that can be seen as an offset from the beginning of a lane segment. Note that the mapping from the continuous domain to a discrete domain inherently introduces an error in the spacing of the last two nodes of every lane segment which is no greater than half of the defined node spacing. This error is acceptable in this implementation of the NG because of the fidelity at this planning level.

StateElab(nodeName) → UTM Location
StateElab(UTM) → nodeName
StateElab(laneSegment_id, offset) → nodeName
StateElab(nodeName, orientation) → directionInLane
StateElab(nodeName, directionInLane) → orientation

Legend:
StateElab() - overloaded function for state elaboration
nodeName - unique road state identifier
directionInLane - direction of travel with respect to how the lane segment is rendered
UTM – Universal Transverse Mercator

Figure 6. Psuedo-code prototypes for state elaboration function

The NG provides the planner with the ability to extract and elaborate road states from the KD using logical and mathematical models. Figure 6 illustrates the knowledge that can be derived from partial state knowledge by the state elaboration function in the NG. For example, given a road state unique identifier, known as a nodeName, and the orientation of the vehicle in the current state, the NG is able to derive the direction the vehicle is traveling in a lane segment with respect to how the lane is rendered in the knowledge base.

Arc Length of the entire lane segment

$$L_c = \left[\overrightarrow{CB} \angle \overrightarrow{CE} \right] * r \qquad L_S = \left\| \overrightarrow{BE} \right\|$$

Number of nodes in a given segment

$$N_* = \left\lfloor \frac{L_*}{s} + 0.5 \right\rfloor$$

Angle corresponding to *s* for a particular lane segment

$$\Theta_c = \frac{s}{r} \qquad \Theta_s = 1.0$$

Determining UTM locations of individual nodes

$$UTM_c = \begin{vmatrix} \cos(\Theta * i) & \sin(\Theta * i) \\ -\sin(\Theta * i) & \cos(\Theta * i) \end{vmatrix} \overrightarrow{CB} + \vec{C}$$

$$UTM_s = \vec{B} + s * \frac{\overrightarrow{BE}}{\left\| \overrightarrow{BE} \right\|} * i$$

Legend:
UTM vectors that define a lane segment
C=Curvature Center, B=Start Point, E=end Point
w = width of lane segment
s=uniform arc length separating each node in lane segment
$r = \left\| \overrightarrow{CB} \right\|$, Radius
i = Road node index into lane segment
Subscripts:
c=curved lane segment, *s*=straight lane segment, *=both

Figure 7. Mathematical models for knowledge extraction

To efficiently handle numerous road states without the implementation of a large complex data structure, the road states are derived from knowledge stored in the KD. Figure 7 depicts the mathematical models that the NG uses to extract and elaborate road states form the knowledge residing in the KD. For instance, the UTM location of a road state in a curved lane segment is derived by utilizing a rotational matrix, whose angle corresponds to the angular offset of the road state in the lane segment. The matrix is used to rotate the radial vector that runs from the curvature center to the starting point of the lane segment. The rotated vector is then translated to the proper UTM locations by adding the UTM location of the curvature center of the lane segment.

3. Constrained Simulator and State Predictor

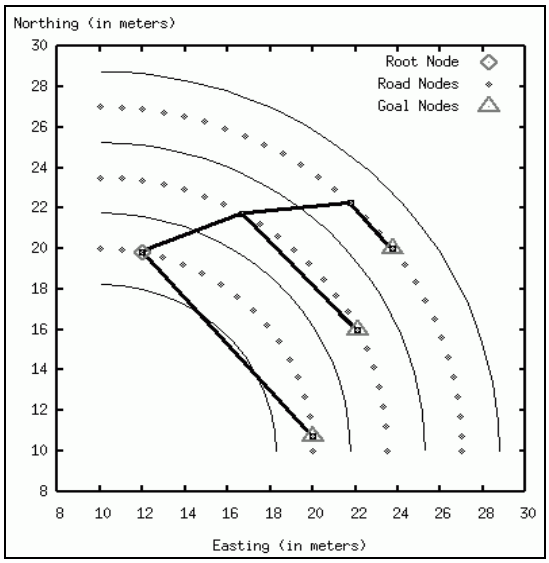

Figure 8. Reachability graph

The constrained simulator of the NG is responsible for the creation of the reachability graphs shown in Figure 8. Each branch of the graph represents a separate trajectory that models the distance traveled and the maneuvers performed by the vehicle (i.e., turn, change lane, maintain lane) per cycle. A trajectory is represented by the NG as an ordered subset of road states that are connected to a given root node to form a reachability graph. The leaf nodes of the graph are the set of obtainable goal states for a given cycle.

The simulator utilizes the constraints applied to the system when creating the reachability graphs to limit or dictate the maneuvers that a vehicle can perform. For example, on a first planning pass the NG may be constrained from returning any road states that violate a driving law or would produce uncomfortable vehicle movement. However, if no plan is found that satisfies the planner's maximum cost value, replanning could take place with a reduced set of constraints. The constraints could now allow for emergency maneuvers by altering the angle in which lane changes are performed or allow nodes deemed illegal by the set of road rules.

The simulator creates the reachability graphs by applying equation-based algorithms to the internal state of the vehicle. The internal state of the vehicle consists of the vehicle constraints, current road state, orientation, and velocity. The current base set of functions used in the simulator allows for the vehicle to move along the lane segment or to change lanes.

The function that moves along a lane is depicted by a UML flow-chart in Figure 9. Note that this figure assumes that the arc length between the uniformly spaced road states is one meter. When the function is called, the number of road states that can be traversed (*deltaNode)* is initially computed. If *deltaNode* is greater than or equal to the number of road states in the lane segment (*nodeCount*) then the simulator must get a handle to the lane segment linking to the end of the current lane segment and the appropriate road state of the adjacent lane segment. Once the link to the adjacent road state is found, the simulator determines how many nodes it may still traverse (*rDist*). The function is recursively called to move along the lane in order find a leaf node. During the recursive process, the function maintains an ordered set of road states that consists of a given start node, a leaf node, and the first node of every lane segment traversed during the process.

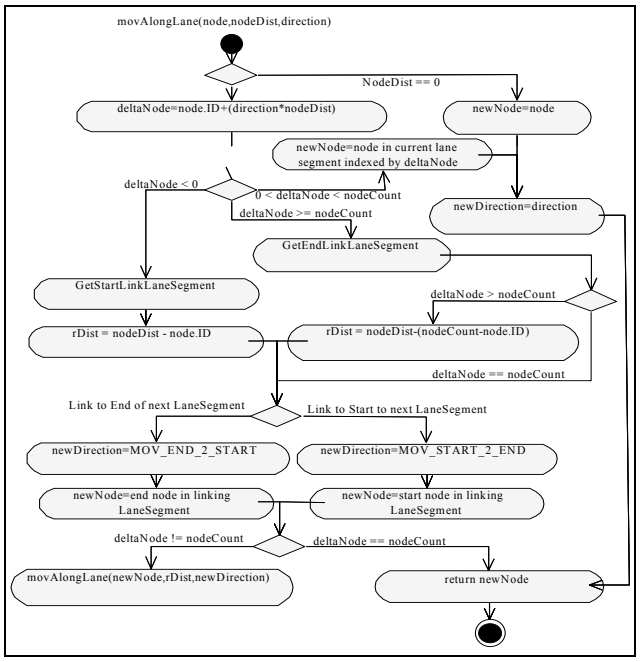

Figure 9. UML flowchart of function for moving along a lane

The change lane function uses a trigonometry-based equation to model a lane change while adhering to the lane change angle required for the maneuver. Equation 1 shows how this function calculates the forward, *F*, component that is required to model the lane change angle α given the lane change width *w*. Equation 2 shows the means in which the function ascertains the corresponding node index *i* in the adjacent lane segment.

$$\text{Equation 1: } F = \left\lfloor \frac{w}{\tan \alpha} + 0.5 \right\rfloor$$

$$\text{Equation 2: } i_{adjacent} = \left\lfloor \frac{\Theta_{current} * i_{current}}{\Theta_{adjacent}} + 0.5 \right\rfloor$$

Figure 10. Trigonometry-based equations used by the simulator to change lanes. See nomincature for Θ in Figure 7.

The simulator builds the reachability graph one trajectory at a time using the two functions discussed previously. When constructing a reachability graph, the simulator attempts to find additional arms of the reachability graph (representing multiple lane changes) by recursively searching adjacent lanes using the two base equations mentioned above. This is accomplished by first performing a lane change maneuver to the adjacent lanes of the root node. If an adjacent lane segment exists, and a node in this lane segment is obtainable at the given vehicle's speed, then the function forms a trajectory to this adjacent lane. This trajectory is created by connecting the root node of the reachability graph to the node found during the lane change maneuver with nodes that model the vehicle's traversal to this lane as intermediate nodes in the trajectory. A completed reachability graph that is used by BG is depicted in Figure 8.

VI. Further Work

The NG will to be extended in the future to handle more complex road structures, such as large cloverleaf intersections and parking lots. This will require the further development of the on-road driving database (Schlenoff et al. 2004) as well as the intelligence contained in the NG.

The NG will also be expanded to handle larger environments by dividing the world into multiple grids that can be cached in to and out of memory. A statistical model will have to analyze how BG expands the planning graph in order to determine the appropriate way to prune the grids that are in memory.

It is predicted that the incorporation of this knowledge layer into VJ will improve the performance of BG. By increasing the intelligence in the knowledge layer, the NG will be able to assist the heuristic search to make a more accurate estimation of the cost that will be incurred in reaching the goal. This will reduce number of nodes in the planning graph, which in turn will lead to better performance by the behavior generation.

References

1. Pomerleau, D. and Jochem, T. 1994. Image Processor Drives Across America. *Photonics Spectra* 30:80-85.
2. Thorpe, C. et al. 1991. Toward Autonomous Driving: the CMU Navlab. I. Perception. *Expert, IEEE [see also IEEE Intelligent Systems]* 6:31-42.
3. Russell, S. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall.
4. Guo, Y., Qu, Z., and Wang, J. 2003. A New Performance-Based Motion Planner for NonHolonomic Mobile Robots. In the Proceedings of the 2003 Performance Metrics for Intelligent Systems Workshop
5. Balakirsky, S. 2003. *A Framework for Planning With Incrementally Created Graphs in Attributed Problem Spaces*. Berlin, Germany: Akademische Verlagshesellschaft Aka GmbH.
6. Albus, J. and Meystel, A. 2001. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. John Wiley & Sons, Inc.
7. Albus, J. 1991. Outline for a Theory of Intelligence. *IEEE Transactions on Systems Man and Cybernetics* 21:473-509.
8. Albus, J. et al. 2001. *The RCS Handbook: Tools for Real-Time Control Systems Software Development*. John Wiley & Sons, Inc.
9. Nilsson, N. J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco: Morgan Kaufmann Publishers, Inc.
10. Blum, A. L. and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90:281-300.
11. Schlenoff, C. et al. 2004. The NIST Road Network Database. NISTIR. Forthcoming