Planning for On-Road Driving Through Incrementally Created Graphs

Stephen Balakirsky, *Member, IEEE* and Chris Scrapper

***Abstract*— A general-purpose hierarchical planning framework that allows for cost-optimal, logic-constrained plans will be presented. This framework will be applied to planning an on-road path for an autonomous vehicle traveling amongst moving and stationary objects. Through this application, the ability to implement both hard and soft constraints, as well as cope with dynamic environments and user objectives will be demonstrated.**

I. Introduction

Many road-driving planning systems exist in the current literature [1-4]. Researchers such as Dickmanns have demonstrated the ability to drive at high-speeds amongst moving objects, while Pomerleau and Jochem have driven across the United States in a mostly autonomous fashion. Still others have taken advantage of specially modified roadways for vehicle guidance [4] or created systems with all-terrain mobility [3]. While these systems are capable of low-level lane following, and even in some cases of changing lanes and navigating exit ramps, they lack the ability to plan for when a lane-change or passing maneuver should take place. The planning system that is described in this paper is designed to work in a hierarchical architecture as the supervisor to one of these low-level planning/control systems in order to construct plans for mixed traffic (autonomous and manual) driving on non-instrumented roadways. This system plans out to the next 10 seconds of travel and is capable of planning on the driving maneuver level (i.e. passing maneuvers, turning maneuvers).

II. The Reference Model Architecture

In order to guarantee real-time operation and decompose the problem into manageable pieces, it was 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 [6,7].

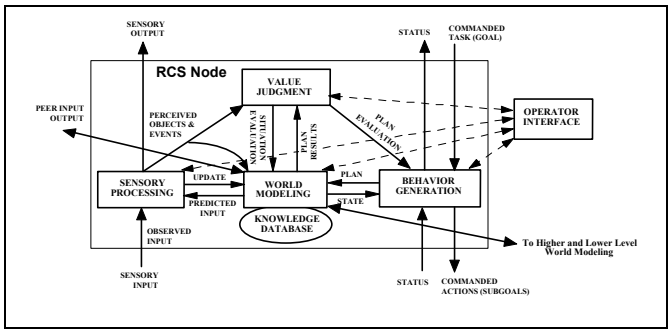

Figure 1. Internal structure of a RCS_NODE (from [6] p. 28).

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

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 hierarchy. The system receives a set of intersections that must be traversed and a final goal location 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. 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.

III. Planning Framework

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.

A. Attributed Graph Nodes

The left side of Fig. 2 depicts several static objects that may be in an planning system's world model (a high density black object (i.e. a brick wall) and a low density white object (i.e. an empty can). In addition to knowledge about an object's location, the world model tracks all of the

This work was supported in part by the Defense Advanced Research Project Agency's Mobile Autonomous Robot Software program and the Army Research Laboratory's Robotics Demo III program

Both authors are with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA (phone:301-975-3418; fax: 301-990-9688; e-mail:{stephen, scrapper}@nist.gov).

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, 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 its 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 over a brick wall. However, the system should be willing to incur substantial damage (i.e. hit the wall) rather then running over a small child (which would not damage the vehicle).

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 the right had side of Fig. 2, 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 Fig. 2 (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.

B. 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 [8]. 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.

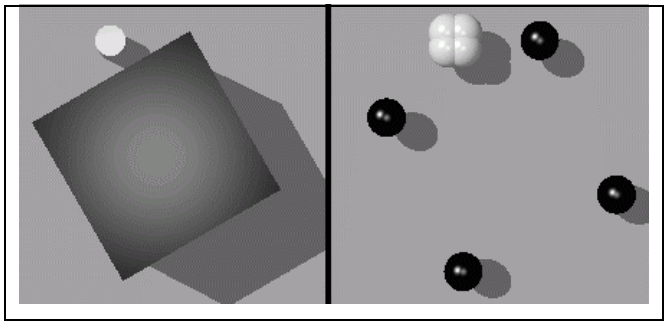

Figure 2. Representative objects and resulting nodes.

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 where a decision on vehicle velocity and lane control (which lane to occupy) must be made every second (for this level of the hierarchy). 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 Fig. 3 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.

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. This may be accomplished in two different ways. The first is to assign a high arc cost to the action of crossing a double yellow line on the road. This will discourage the vehicle from entering the opposing lane. The second is to assign a high state cost to occupying a state whose direction of travel is opposite of the vehicles. This will assure that once in the opposing lane, the stay is as short as possible.

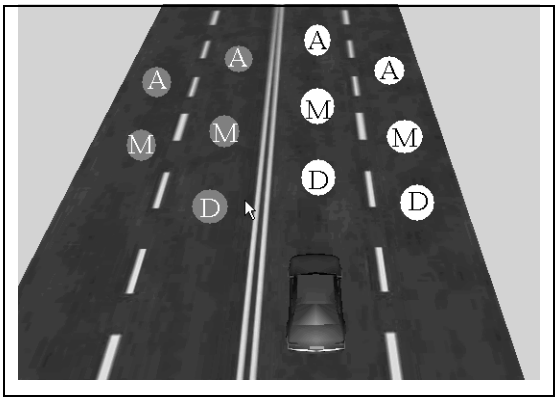

Figure 3. Spanning set for on-road planning system.

C. Planning Algorithm

The graph node creation and connection strategies outlined above may be used to create a formal algorithm that resides in the behavior generation module of RCS and is outlined as follows (see [5] for a more detailed analysis):

1) Insert $\text{rand}_{\overrightarrow{\text{start}},t_0}$ into open list O. The notation $\text{rand}_{\overrightarrow{\text{start}},t_0}$ specifies a member of the set **RAND** with state specified by the vector "start" at time t_0.

2) Insert one or more goals $\text{rand}_{\overrightarrow{\text{goal}},t_g}$ into goal list G.

3) Graph search selects and removes $\text{rand}_{\vec{k},t_i}$ from O.

4) Evaluate $\text{rand}_{\vec{k},t_i}$.

   a) If $\text{rand}_{\vec{k},t_i} \in$ G, and re-planning desired goto (1),

   b) else if $\text{rand}_{\vec{k},t_i} \in$ G finished,

   c) else if $\alpha_{\vec{l},t_{i+1}} >$ MaxCost, then no path found,

   d) else continue.

5) Define the set to be visited TBV = $SP_{\text{rand}\vec{k},t_i}$

6) While TBV $\neq \varnothing$, create arc ($\text{rand}_{\vec{k},t_i}$, $\text{rand}_{\vec{l},t_{i+1}}$) and evaluate $\beta = pc(\text{prand}(\text{rand}_{\overrightarrow{\text{start}},t_0}, \text{rand}_{\vec{l},t_{i+1}}))$ where $\text{rand}_{\vec{l},t_{i+1}} \in$ TBV.

   a) Remove $\text{rand}_{\vec{l},t_{i+1}}$ from the set TBV.

   b) If $\text{rand}_{\vec{l},t_{i+1}}$ VISITED and $\beta < \alpha_{\vec{l},t_{i+1}}$, or $\text{rand}_{\vec{l},t_{i+1}}$ NOT VISITED, set $\alpha_{\vec{l},t_{i+1}} = \beta$, set $\text{rand}_{\vec{l},t_{i+1}}$ backpointer to $\text{rand}_{\vec{k},t_i}$, add $\text{rand}_{\vec{l},t_{i+1}}$ to O.

   c) Remove arc($\text{rand}_{\vec{k},t_i}$, $\text{rand}_{\vec{l},t_{i+1}}$).

   d) End while.

7) Go to 3.

Step 1 and 2 of the algorithm begin the search process by placing the node that contains the current system state into the list O of open nodes and one or more goal states into the goal set G. The algorithm will terminate when a minimum cost path has been found to any of the nodes in the set G or some maximum cost has been exceeded. It should be noted that the feature of being a start or goal node is a relevant attribute and therefore the start and goal nodes are always relevant and elements of **RAND**. This assures that the graph will contain the start and goal states even if these states have no other relevant attributes associated with them.

In step 3, it is assumed that a search technique that utilizes an open list is used. Many such techniques exist in the current literature including searches such as breadth first [9], depth first [9], and A* [10]. A search methodology dependent criterion is used to select the node from the list O that should be expanded. If the list O is empty, then a plan failure has occurred. At this point, the system may either return a failed status indicating that no plan was found, or may relax the selection criteria for relevant attributes (i.e. increase the size of NRAND) and attempt to find a new plan. Through this technique, a hierarchy of hard constraints may be established that are gradually relaxed as a result of plan failures.

Step 4 performs a test to determine if the current node is a member of the goal set G or if its cost exceeds the maximum allowable cost. The goal set is defined as the set of nodes in **RAND** that will satisfy the objectives of the system. This set may contain more then one node (e.g. a mobile robot should arrive at a certain location as soon as possible, with no specific time information given), but should never be empty.

One integral feature of this algorithm is dynamic replanning. By returning to step 1 after a successful plan has been generated, dynamics in the hard constraints, soft constraints, and environment are easily handled.

Steps 5 and 6 incrementally expand and evaluate the graph structure. The spanning set of the node being expanded is constructed, and the cost of achieving these nodes from the current node is examined. During the spanning set construction, possible actions and relevant attributes are examined, and nodes that correspond to these attributes are either located (they have been previously created) or are created. Created nodes always begin with a status of NOT VISITED. This indicates that no path from the graph origin to the node has been explored and evaluated.

Arcs are temporarily created to link the current node to all of its successors, and the path cost function pc evaluates the new potential path to the successor node. Step 6b assures that the least expensive path from a given node to the start of the search may be found by following a series of backpointers, and the least expensive cost from a node to the start of the search is maintained in the variable $\alpha_{\bar{l},t_i}$. Since only a single path from each node back to the start is maintained, the acyclic property of the graph is guaranteed. An error flag is returned if step 4 finds that the value of $\alpha_{\bar{l},t_i}$ for the node being expanded exceeds a maximum cost. This flag may be used by different search techniques in different ways. For example, an iterative deepening search may use this transition to a new graph branch while an A* search would need to relax constraints on the generation of relevant attributes and replan.

IV. Knowledge Layers

The heart of the algorithm presented above lies in the determination of a node's spanning set and the evaluation of the node and arc costs. For the case of on-road driving, a mini-expert system has been developed for the determination of spanning sets and a rule base is used for cost evaluation.

A. Spanning Set Generation – Road Nodes

The spanning set mini-expert system is referred to as the Node Generator (NG). The NG is part of the RCS world model and facilitates the simulation and prediction of possible spatial transitions along a road network that a vehicle may take from its current state. The NG consists of a knowledge engine that creates and maintains the data structure, a Knowledge Database (KD) known as the On-Road Driving Database [11] that contains both a priori and sensed road feature data, a constraint system that limits the graphs branching factor, and a simulation system that is used to predict node locations.

The knowledge engine creates a polymorphic internal data structure that is used to handle the various elements contained in the task decomposition hierarchy of the On-Road Driving Database. From this internal data structure, the knowledge engine is able to construct reachability graphs. Each branch of the graph represents a separate vehicle trajectory that models the distance traveled and vehicle action performed (i.e. turn, change lane, maintain lane) per graph expansion cycle. A trajectory is represented by the NG as a set of road states that are connected to a given root node to form the reachability graph. The leaf nodes of the graph represent the set of obtainable road states for a given cycle. These leaf nodes are used in step 5 of the planning algorithm to expand the incrementally created graph.

The internal knowledge representation of the NG is based on the road data hierarchy described by Schlenoff et. al in [11]. In this hierarchy, the lowest level of knowledge is a lane segment that represents a constant curvature section of a lane. The NG must map these real-world lane segments into a finite set of discrete uniformly spaced attributed nodes. These nodes are uniquely identified by a tuple that includes the lane segment information and a node id that may be seen as an offset from the beginning of the lane segment. Due to the nature of the nodes, an inherent error is introduced between the two last nodes in every lane segment that is defined by the max node displacement. The reason for this is the uniform nature of the node displacement.

The NG utilizes system constraints when creating the reachability graph to limit the vehicle's possible actions. 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 is called in step 5 of the planning algorithm to determine the direction and distance that the vehicle will travel from a given state in one graph expansion cycle. The simulator relies on three equation-governed algorithms to create a reachability graph per cycle. First simulating a trajectory in the current lane and then searching for possible trajectories in adjacent lanes creates this graph. As shown in Fig. 5 the search for additional trajectories always move away from the root node.

In creating the reachability graph, the NG uses state information about the vehicle's direction of travel, velocity, and the node generation constraints. The current base set of functions used in the knowledge engine allow for the vehicle to move along the lane segment or to change lanes. The function that moves along a lane, depicted by a UML flow-chart in Fig. 6, is a recursive function that moves along a lane a given distance to find the leaf node. 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.

The change lane function models a lane change while adhering to the lane change angle required for the maneuver. Equation 1 shows how this function calculates the forward 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.

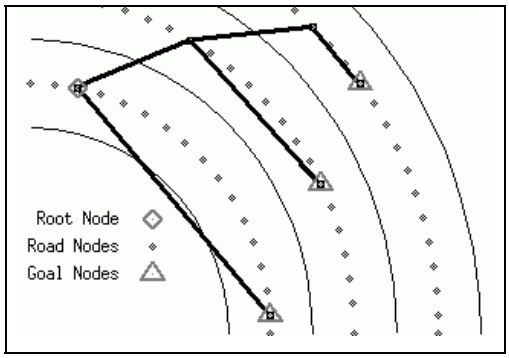

Figure 4. Trajectory generation.

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

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

The knowledge engine 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 vehicles 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 vehicles traversal to this lane as intermediate nodes in the trajectory.

B. Cost Evaluation

The results from the node generator formulate a set of next possible states that the system may achieve. They do not however, tell the system which state it *should* achieve. This job is performed by RCS's value judgment through the use of a cost function. As previously mentioned, the cost function may be broken down into the two evaluations of node occupancy cost and arc traversal cost.

The node occupancy cost may be determined by a weighted sum of a logical combination of individual node attributes. For example, if the cost of occupying the rightmost lane of a multi-lane roadway is said to have unit cost, then the cost of occupying the leftmost lane of the roadway (in the proper direction of travel) may have a cost that is slightly above unit cost while occupying a lane in the opposing direction of traffic would have a cost significantly above unit cost. The combination of opposing traffic direction **AND** no passing zone could be made to have an even greater cost. This cost assignment would encourage the vehicle to travel in the rightmost lane except to pass. Currently, these weights are hand tuned. Automatic determination of the planning weights is an area of future research.

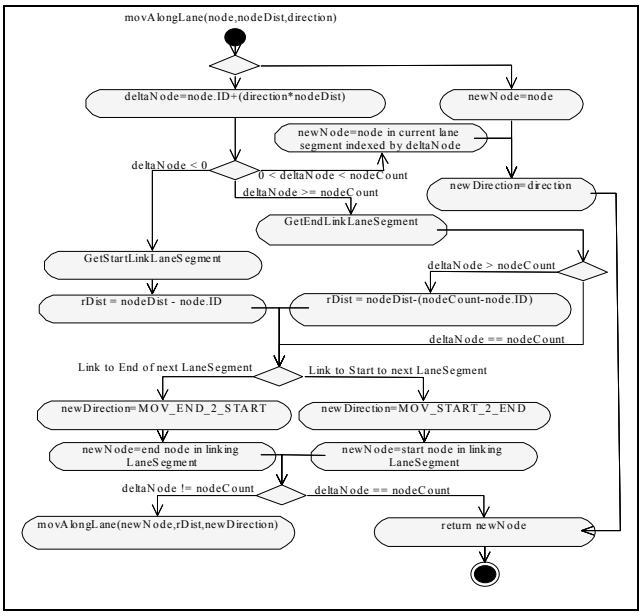

Figure 5. Activity diagram for lane traversal.

The arc traversal cost assigns a cost to performing a particular action. Applying a cost to acceleration and deceleration may be used to encourage consistent velocity control. Additionally, the starting and ending states may be examined in the context of the action to assign additional penalties. For example, changing lanes when the current state is a no-passing zone would be penalized.

C. Object Collision Determination

In addition to determining a path that does not violate any rules-of-the-road, it is desired to control which objects the vehicle is allowed to run over. As static objects are detected, their location and properties are stored in the world model hashed on the lane segment and nearest node. When arc costs are computed, any arc that traverses a node tagged with an object causes further evaluation to be performed.

This evaluation takes to form of a query into an external ontology that determines the predicted damage to the vehicle, vehicle's payload, and object based on the object type, vehicle type, and closing velocity [12]. Moving objects are handled in a similar fashion. In this case the predicted trajectory of the object is stored in the world model, and when a collision or near miss is predicted the ontology may be consulted for collision consequence information. This collision consequence information is combined with the mission objectives (i.e. an collision that causes no damage is allowed) to formulate an additional component of the final arc cost.

V. Experimental Setup and Results

The framework described in the preceding sections has been coded as a C++ program running in a Linux

environment. The system is capable of distinguishing between two classes of static objects (object class is determined by external sensor processing) and a single class of moving objects. The detected location of static objects is assumed to be accurate, and the class difference specifies whether it is possible to run over the object without vehicle damage. Fig. 7 displays a sample computed path where traversable objects are represented as boxes and untraversable objects are represented as cones. For moving objects, the exact trajectory of the object is known a priori. The traffic vehicles in the figure are typical moving objects.

The framework accepts a goal vector that specifies a single spatial location, a range of goal times (may be "any time"), and a range of goal velocities (may be "any speed"). The output of the planner is a set of road states that defines a cost optimal path spaced at one-second intervals in time. This is shown in Fig. 7 as the series of spheres that depicts the vehicle accelerating, changing lanes and driving through traversable objects, changing back to the right-hand lane and decelerating to a stop. The current planning graph contains three separate acceleration and three separate deceleration profiles. The road-node generator is currently able to generate nodes for any curvature of road without intersections.

One of the major areas that affects the system's performance is the search heuristic utilized by the search in step 3 of the framework's algorithm. Currently, an A* search is being utilized with a heuristic that assumes that the remaining distance to the goal is traversed by accelerating at the maximum acceleration rate to the speed limit and then maintaining that speed until the goal is achieved. This heuristic works well for straight road segments and shows performance degradation for paths that involve curves or turns. The reason for this is that the heuristic assumes a straight-line distance from the current location to the goal, and this may significantly underestimate the true distance.

VI. Conclusions and Future Work

An algorithm for on–road planning in a hierarchical architecture has been presented. This algorithm has been implemented in a general purpose planning framework and utilizes a "mini" expert system on graph node placement for on-road driving that has also been developed for this effort. This system has been shown to work on a variety of road-types, and in the presence of several different classes of static and dynamic objects.

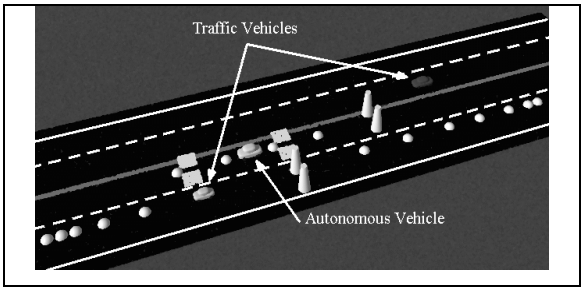

Figure 6. Actual plan amongst various class of static and moving objects.

This system is under constant development and refinement with 3 major areas of concentration. The first area of work is the expert system for on-road graph node placement. This system is currently being expanded to include proper node placement for traversal through complex intersections.

The system's behavior and abilities are greatly affected by the complexity and richness of the system cost function. Additional environmental factors need to be encoded in this function and additional behaviors need to be examined.

Finally, additional components of the RCS system are being developed to allow for this system to operate in real-time on a real vehicle. An executor for plan control and execution is being developed and real-time constraint issues are being investigated.

References

1. Dickmanns, E., "The seeing Passenger Car VaMoRs-P," *International Symposium on Intelligent Vehicles*, 1994.
2. Pomerleau, D. and Jochem, T., "Image Processor Drives Across America," *Photonics Spectra*, 1996, pp. 80-85.
3. Shoemaker, C. and Bornstein, J. A., "Overview of the Demo III UGV Program," *Proceedings of the SPIE Robotic and Semi-Robotic Ground Vehicle Technology Conference*, Vol. 3366, 1998, pp. 202-211.
4. Thorpe, C., Jochem, T., and Pomerleau, D., "The 1997 Automated Highway Free Agent Demonstration," *IEEE Conference on Intelligent Transportation System*, Boston, MA, 1997, pp. 496-501.
5. Balakirsky, S., *A Framework for Planning with Incrementally Created Graphs in Attrributed Problem Spaces*, Akademische Verlagshesellschaft Aka GmbH, Berlin, Germany, 2003.
6. Albus, J., et. al, "4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems," NISTIR 6910, Gaithersburg, MD, 2002.
7. Albus, J., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 21, 1991, pp. 473-509.
8. Blum, A. L. and Furst, M. L., "Fast Planning Through Planning Graph Analysis," *Artificial Intelligence*, Vol. 90, No. 1-2, 1997, pp. 281-300.
9. Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice-Hall 1995.
10. Hart, P. E., Nilsson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100-107.
11. Schlenoff, C., Balakirsky, S., Barbera, A., Scrapper, C., Hui, E., Paredes, M., and Ajot, J., "The NIST Road Network Database: Version 1.0," National Institute of Standards and Technology, NISTIR, 2003.
12. Uschold, M., Provine, R., Smith, S., Schlenoff, C., and Balakirsky, S., "Ontologies for World Modeling in Autonomous Vehicles," *Proceedings of the 18th International Joint Conference on Artificial Intelligence: Ontologies and Distributed Systems Workshop*, 2003.