Available online at www.sciencedirect.com
SCIENCE DIRECT®
Robotics and Autonomous Systems 49 (2004) 57–66
Robotics and Autonomous Systems
www.elsevier.com/locate/robot

Knowledge representation and planning for on-road driving

Stephen Balakirsky*, Chris Scrapper

National Institute of Standards and Technology, Intelligent Systems Division, 100 Bureau Drive, Gaithersburg, MD 20899, USA

Received 27 July 2004; accepted 27 July 2004

Abstract

This paper presents a cost-based adaptive planning agent and knowledge layers that is operating at one level of a deliberative hierarchical planning system for autonomous road driving. At this level, the planning agent is responsible for developing fundamental driving maneuvers that allow a vehicle to travel safely amongst moving and stationary objects. This is facilitated through the application of knowledge to the graph creation process and the use of dynamic cost function within the incrementally created planning graph. The cost function varies to comply with particular road, regional, or event driven situations, and when coupled with the incremental graph expansion allows for the agent to implement hard and soft system constraints. Further discussion will be provided that details one of the expert systems that is implemented to provide the planning system with this knowledge in real-time.

Keywords: Knowledge representation; Planning; Autonomous road driving; RCS

1. Introduction

In the mid-1990s, 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 [1]. This demonstration targeted a highly structured driving environment in the nations highway systems. Thorpe et al. [2] 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 autonomous vehicle planning systems attempt to create an agent function for goal-based autonomous vehicles. This function attempts to map the percepts from the vehicles sensors to possible actions the vehicle can take. Russell and Norvig [3] discuss deficiencies of table-driven agent functions, i.e. finite state machines or look-up tables. The authors claim that

* Corresponding author. Tel.: +1 301 975 4791; fax: +1 301 990 9688.
E-mail addresses: stephen.balakirsky@nist.gov, stephen@nist.gov (S. Balakirsky), christopher.scrapper@nist.gov (C. Scrapper).

0921-8890/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2004.07.016

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 $\sum_{t=1}^{T}|P|^t$, where T is the total number of percepts received by the agent, and P the set of possible percepts.

Since the generation of appropriate behaviors in stochastic environments may contain a countably infinite sequence of percepts, some deliberative planning systems use planning graphs that determine the optimal path through cost analysis. Guo et al. [4] discuss a typical motion planner that attempts to use this approach to tackle the global trajectory-planning problem. However, their approach has shown to be unable to handle the computational complexities in real-time. Balakirsky [5] has developed a real-time deliberative planning system that attempts to limit the computational complexity and has shown positive results in real-time. This approach is based on an A* graph search algorithm that limits the size of the graph through the application of knowledge throughout the graph creation and evaluation process. This knowledge is derived from a variety of sources that includes a priori world knowledge bases, sensor systems, and situation assessment systems.

This paper will provide an overview of this planning approach and the areas that knowledge has been applied to limit the systems graph complexity. Further discussion will be provided that details one of the expert systems that is implemented to provide the planning system with this knowledge in real-time. The layout of the rest of this paper is as follows. Section 2 contains a brief synopsis of the real-time control system reference architecture, *RCS*. In Section 3, the knowledge requirements of our general planning approach are discussed. Section 4 provides a detailed description of one of the systems knowledge generators and cost generators, and shows how this knowledge is derived. Finally, Section 5 will conclude the paper with a summary and discussion of future work.

2. 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].

Under RCS, each level of the hierarchy is composed of the same basic building blocks. 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 judgement (cost/benefit computation, goal priority).

For the case of on-road driving, the overall task may be decomposed into a five-level hierarchical system where each level refines its supervisor's plan output. An example of such a hierarchy is depicted in Fig. 1.

Each of these levels will have their own unique planning objective, re-planning rate, and knowledge requirements. The planning system presented in this paper is designed to fill the roles of behavior generation, world modeling, and value judgement for a single level of the hierarchy (level 3). 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 to account for changes in the planning horizon

Level	Planning Task	Example
5	Destination and ordering constraints	Go to supermarket and then bank
4	Specific route to get to a single destination	Drive First Street until Market; Turn Left on Market; ...
3	Lane and speed management to accomplish section of route	Drive left lane using acceleration profile 'a' until time=10; Change to right lane with lane change profile 'la'
2	In-lane control to follow lane and avoid small objects	Drive trajectory (parameters)
1	Actuator positions	Throttle 20%, steering5%

Fig. 1. Example hierarchical decomposition for road driving problem.

and uncertainty in the prediction of moving object locations and the success of task execution.

The world model utilized by the planning system for this level of the hierarchy is adapted from (7) to contain multiple knowledge layers within the WM, e.g. moving object prediction, road trajectory generator, a priori maps, etc. The knowledge layers discussed in this paper are expert systems that lie at the heart of the planning system for on-road driving. In order to describe the general knowledge requirements of these layers, the layer known as the Node Generator (NG) will be discussed in detail. The NG is composed of a simulator/predictor and a knowledge database (KD). The NGs KD contains a fused combination of sensed and a priori data that fully defines the road network (number of lanes, road curvature, lane markings, etc.), while the simulator/predictor simulates spatial/temporal transitions within the road network in order to provide predictions about future states and state transitions. These simulated transitions, along with the embedded road knowledge, are fused with the output of the other knowledge layers and evaluated by value judgement (VJ) to construct and analyze an incrementally created graph. The result of this graph evaluation produces a plan that exhibits the appropriate behaviors for navigating road networks.

3. Planning framework

The planning system utilized by level 3 of the road driving system is based on the incremental creation and evaluation of a planning graph as detailed by Balakirsky [5]. This system must apply various types of knowledge to accomplish the tasks of goal generation, graph construction, graph evaluation, and algorithm termination.

3.1. Goal generation

At the beginning of each planning cycle, the executor from the planner's supervisor must create a goal set that contains approximate locations that the vehicle will strive to achieve near the time of the time-based planning horizon. Future steps of the planning algorithm will cause termination when one of these goal states is achieved, or produce an error status report if none of the states may be achieved within a time margin of the planning horizon. Due to vehicle motion, the planning horizon will change with each planning cycle and the location of the members of the goal set will change. The actual vehicle will never execute more then 10% of the computed plan before the next planning cycle delivers a new plan for execution.

Fusing event driven command input (drive First Street until Market) and world knowledge (a traffic signal exists one block ahead on First Street) constructs the planner's goal set. For example, if traveling down a multi-lane road that contains a traffic signal, one potential goal location would be at the signal's intersection (the vehicle was stopped by a red light or congestion) and another would be some distance down the road that may be calculated by the vehicle's expected speed and the planning horizon. The executor currently uses a combination of a rule base and the NG (described in Section 4.2) to construct the goal set.

3.2. Graph construction

The plan at level 3 of the hierarchy is constructed through the use of an incrementally created graph (5). This approach utilizes knowledge generators to suggest potential next system states based on the current state being evaluated. Each of these knowledge generators is capable of determining possible state transitions for a restricted set of input states. For example, the current level 3 system utilizes one knowledge generator for on-road driving and a second, different generator, for off-road driving. Environmental knowledge about the location of roadways and information about the supervisor's desired goals (is the system trying to get off the road, or follow the road?) is utilized in making decisions about which generator (or combination of generators) to use.

The on-road knowledge generator, which is detailed in Section 4.2, utilizes the vehicles current state (location, velocity, etc.), environmental information (weather, visibility, etc.), road knowledge (number of lanes, road curvature, etc.), and supervisor constraints (aggressiveness, conformance to rules of the road, etc.) to algorithmically determine possible next states that lie along the roadway. The off-road knowledge generator incorporates similar information to construct a visibility graph around "no-go" region bounding boxes in order to find the shortest path that navigates less constrained off-road environments.

3.3. Graph evaluation

Once possible state transitions have been determined, these new states must be added into the graph structure. The transition to each new state is represented in the graph by a graph arc, and each transition is evaluated by all of the knowledge layers. This evaluation summarizes all of the state transition simulations that have been performed by the various knowledge layers and represents all of the events related to the state transition. This information includes knowledge about static and dynamic object interactions, conformance to the rules of the road, and general state variable changes (i.e. the vehicle lane changes). Section 4.3 provides more detail on this operation for the moving object prediction layer.

The information represented by the graph arc must now be summarized into a single value to be used during graph search. The planning system must combine all of the layer's simulation results and the supervisor's constraints to produce this single value. As in the graph generation, multiple cost systems that are specifically tuned to the different classes of terrain that the vehicle traverses exist to fill this function (see Section 4.3 for more information). The current system utilizes separate cost systems for determining the cost of road traversal, intersection traversal, general off-road traversal, and special off-road area traversal.

3.4. Algorithm termination

Once a value has been determined for each new graph arc, a standard graph search algorithm that is capable of producing a complete and optimal path (such as Dijkstra [8] or A* [9]) may be applied to find the next state that will be examined. If this state is a member of the goal set, then the planning function is complete. If not, elapsed computation time and minimum path cost will be examined to determine if a planning failure has occurred. If a planning failure has occurred, further knowledge must be applied to address the failure. It is possible that a sub-optimal solution exists or that rules in the knowledge generators may be relaxed to allow for additional states to be created and a solution to be found.

4. Application of knowledge

While knowledge is applied during all phases of the planning process, the heart of the system lies in the graph construction and evaluation. A sample knowledge generator for graph construction and a simulation engine for graph evaluation will now be presented in detail.

4.1. Detailed graph construction

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.

4.1.1. Attributed 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 object's attributes that are necessary

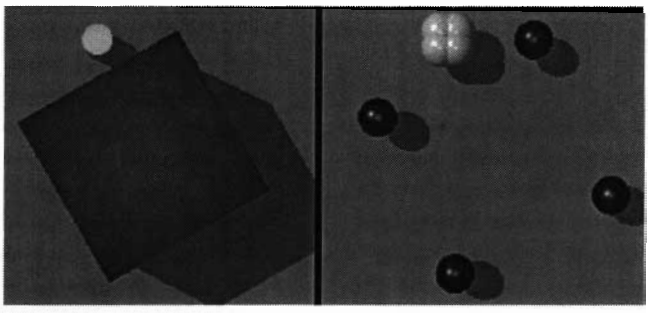


Fig. 2. Representative objects and resulting nodes.

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.

4.2. Road knowledge generator detail

A knowledge base that provides concise, accurate information about the road environment is essential for the successful operation of any deliberative autonomous on-road vehicle. Schlenoff et al. [11] have developed a road network 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 road network database captures the structure and features of the road network in a six level hierarchy that ranges in resolution from roads (bi-directional stretches of travel lanes bounded by proper names) to lane segments (a piece of a lane that consists of a constant curvature arc). Each level of the hierarchy encapsulates the minimum set of attributes needed to derive the appropriate knowledge about road networks for behavior generation at a particular level of abstraction.

The level 3 planning system described in this paper requires the highest obtainable fidelity from the on-road driving database, which is derived from the lower echelon of the decomposition hierarchy. This level of abstraction encodes lane segments and includes direct attributes that represent items such as end points, curvature, speed limit, and segment direction as well as indirect attributes (derived from higher levels of the database) that contain information about road markings, lane widths, lane barriers, and roadway composition.

4.2.1. Road state generator

In general, 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 set of discrete uniformly spaced attributed nodes known as road states. These road states carry with them specific attributes (e.g. lane markings, speed limit) of the lane segment from which they were derived and are each uniquely identifiable. 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. This error is less than half of the defined point spacing, and due to planning fidelity, is acceptable in this implementation of the NG.

The NG uses these road states, along with the vehicle's potential actions, to determine a given state's spanning set (the set RAND) for use in the graph expansion. This spanning set is made up of plausible next road states, known as goal states, and simulated spatial-temporal 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.

4.2.2. Derivation of road states

In order to derive the road states, the NG relies on an internal knowledge database that 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. This data structure, which is implemented as a connected graph of adjacent lane segments, is depicted in the UML logical model in Fig. 4. The road primitives are generalized in polymorphic structures in order to facilitate the extendibility and durability of the data structure.

Fig. 5 depicts the mathematical models that the NG uses to extract and elaborate road states from 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

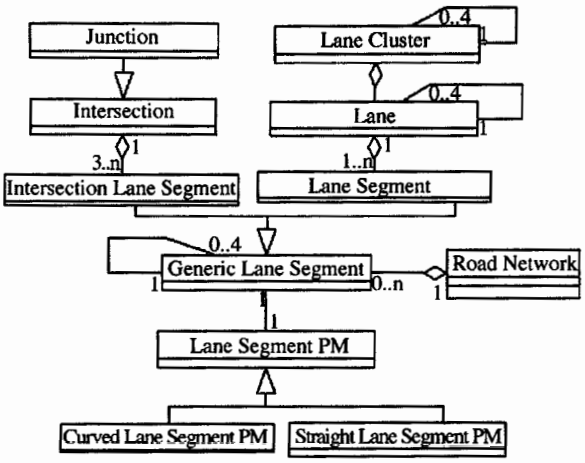


Fig. 4. UML logical model of NGs knowledge database.

the UTM location of the curvature center of the lane segment.

4.2.3. *Spanning set generation*

The constrained simulator of the NG is responsible for the determination of a given nodes spanning set.

Arc Length of the entire lane segment

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

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

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

$$UTM_s = \overline{B} + s_* \frac{\overline{BE}}{\left\| \overline{BE} \right\|} * i$$

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
$r = \left\| \overline{CB} \right\|$, Radius
i = Road node index into lane segment
Subscripts:
c=curved lane segment, s=straight lane segment, *=both

Fig. 5. Mathematical models for knowledge extraction.

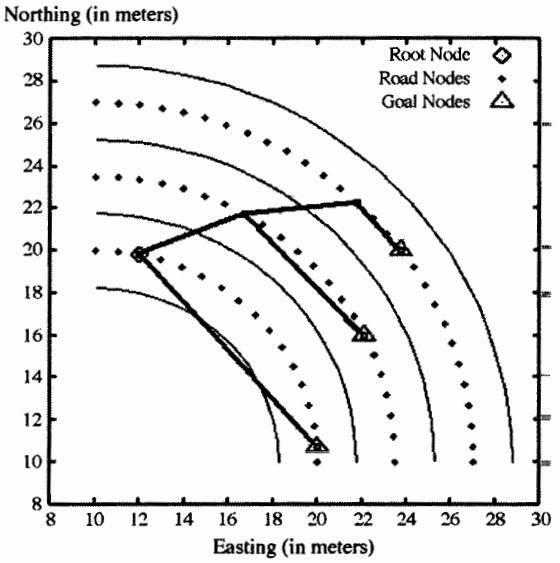
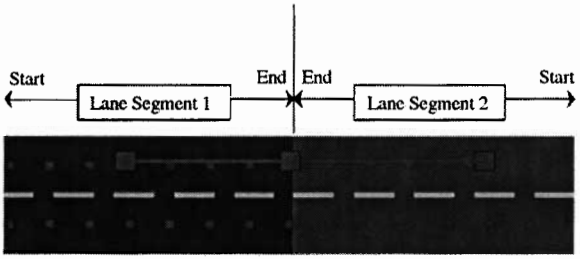


Fig. 6. Reachability graph.

The spanning set of a node is returned as a reachability graph as shown in Fig. 6. Each branch of the graph represents a separate trajectory that models the distance traveled and relevant actions 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 the reachability graph. The leaf nodes of the graph are the set of obtainable goal states for a given cycle.

In order to limit the potential size of the graph and restrict vehicle maneuvers, the simulator utilizes action and state relevance when creating the reachability graphs. 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 cost constraint, re-planning could take place with a reduced set of constraints (expanded RAND). 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 state being expanded. The state consists of the vehicle constraints,




```
movAlongLane(node, nodeDist, direction):
        NextNode = node_id + (direction * nodeDist)
        If NextNode < 0
                get Start_Link_LS
                nodeDist = nodeDist – node_id
        Else if NextNode >= nodeCount
                get End_Link_LS
                nodeDist = nodeDist – ( node.nodeCount – node_id)
        Else    return node in lane
        If Link_Lane_Segment_Side = END
                direction = -1
                node = get end node of linking lane segment
        Else
                direction = 1
                node = get first node of linking lane segment
        movAlongLane(node, nodeDist,direction)
```

Fig. 7. Illustration and pseudo-code for function used to move along a lane.

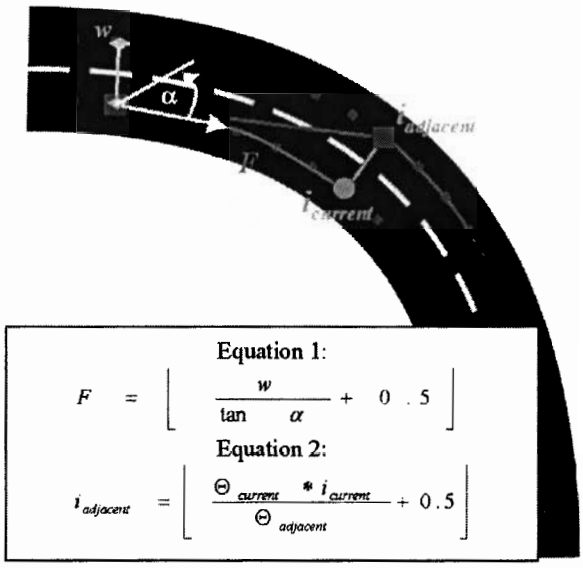


Fig. 8. Trigonometry-based equations used by the simulator to change lanes.

predicted 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 the lane is illustrated along with pseudo-code in Fig. 7. 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.The change lane function depicted in Fig. 8 uses a trigonometry-based equation to model a lane change while adhering to the lane change angle required for the maneuver. Eq. (1) from Fig. 8 shows how this function calculates the forward F component that is required to model the lane change angle a given the lane change width w. Eq. (2) shows the means in which the function ascertains the corresponding node index i in the adjacent lane segment.

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 the planning system is depicted in Fig. 6.

4.3. Cost evaluation detail

During graph construction, the knowledge generators that are detailed in the previous section 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 decision is made through the use of a graph search that finds the "cheapest" combination of nodes and arcs (a path through the graph) that achieves the system's goal. The actual cost of the path is determined by the value judgment module and is made up of the two components of node cost and arc cost. The node cost is computed through the application of rules to the state information that is available in each node. For example, additional cost is accrued for traveling the wrong direction in a lane. Arc cost consists of a combination of the action and effect costs of transitioning from one state to the next and the cost of occupying the area covered during the transition. It is the job of the value judgment module to summarize these result into a single value.

The traffic prediction layer presents an example of the simulations performed to determine the arc cost. This layer has the responsibility of predicting the future location of any moving object that is being tracked by sensory processing. The output of the layer is a probability map that shows the probability that a given object will be at a given location at a given time. This layer is specifically tuned to operate for either off-road or on-road driving. The layer may change modes of operation on an arc by arc basis. For off-road driving, a Kalman filter may be used to predict the near-term location of vehicles moving in the vicinity of our controlled vehicle. For on-road driving, knowledge of the road network, rules-of-the-road, and traffic conditions provides valuable information that aids in computing this location map.

For example, Fig. 9(a) depicts a vehicle about to cross an uncontrolled intersection with no traffic. It is likely that this vehicle will continue in its current lane at its current velocity as it traverses the intersection. If however, this intersection is controlled by a stop sign (Fig. 9(b)), will increase the probability of a lane change to avoid extra stopping time. Under the current implementation of the traffic lyer, a rule base is used to encode these probability changes. The probability changes due to rules firing have been determined in an ad hoc manor, and further research into properly, and automatically, setting these values is ongoing. Once the vehicle's expected actions have been established, locations may be computed. Bayesian theory is then applied to determine the final location probability map that combines all of this information. A detailed description of this layer's functions may be found in [12].

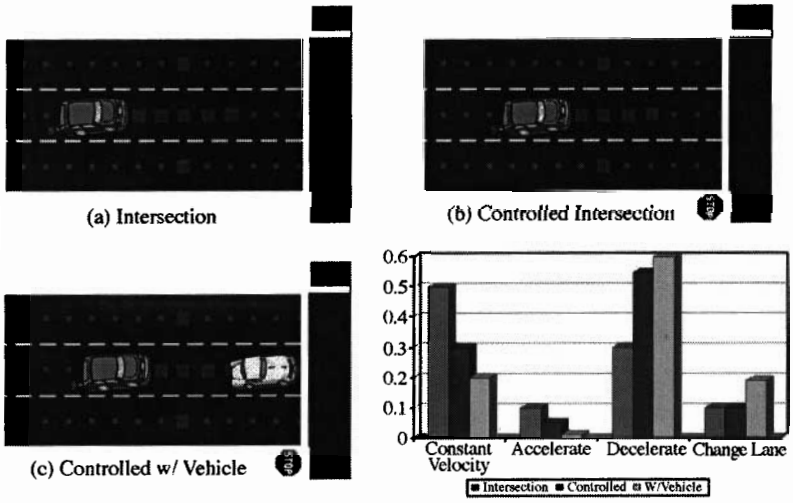


Fig. 9. Example of how the environment affects the predicted vehicle activity.

5. Summary and future work

Future developments for this system are scheduled to occur on the two fronts of real-system operation and theoretical development. While the system currently runs under NISTs simulation environment, it has never operated on any of our real robot systems amongst real traffic. We hope to address this issue this summer by demonstrating road driving amongst traffic on our campus.

On the theoretical side, development will continue on all of the knowledge generators that are part of the planning system. In particular, the NG will to be extended to handle more complex road structures, such as large clover-leaf intersections and parking lots, larger environments, and real-time sensor updates. This will require the further development of the on-road driving database [11] as well as the intelligence contained in the NG.

In order to work with larger environments, the world will be divided into multiple grids that can be cached into and out of memory. A statistical model will have to analyze how the planning system expands the planning graph in order to determine the appropriate way to prune the grids that are in memory. Real-time sensor data will replace a priori estimates of road curvature and lane markings. Significant research into data registration and fusion remains to be performed.

References

[1] D. Pomerleau, T. Jochem, Image processor drives across America, Photonics Spectra.
[2] C. Thorpe, M. Herbert, T. Kanade, S. Shafer, Toward autonomous driving: the cmu navlab. I. Perception, Expert IEEE [see also IEEE Intelligent Systems].
[3] S. Russell, P. Nonig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995.
[4] Y. Guo, Z. Qu, J. Wang, A new performance-based motion planner for nonholonomic mobile robots.
[5] S. Balakirsky, A Framework for Planning with Incrementally Created Graphs in Attributed Problem Space, IOS Press, 2003.
[6] J. Albus, Outline for a theory of intelligence, IEEE Transaction on Systems Man and Cybernetics.
[7] J. Albus, A. Meystel, L. Zadeh, Engineering of Mind: An Introduction to the Science of Intelligent Systems, 2001.
[8] E. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik.
[9] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transaction on Systems Science abd Cybernetics.
[10] A. Blum, M. Furst, Fast planning through planning graph analysis, Artificial Intelligence.
[11] C. Schlenoff, S. Balakirsky, A. Barbera, C. Scrapper, E. Hui, M. Pardes, J. Ajot, The Nist Road Network Database, Version 1.0, Technical Report, National Institute of Standards and Technology, 2004.
[12] C. Schlenoff, R. Madhaven, S. Balakirsky, An approach to predicting the location of moving objects during on-road navigation.

Stephen Balakirsky received the Ph.D. degree from the University of Bremen, Germany in 2003. He is currently a researcher in the Intelligent Systems Division of the National Institute of Standards and Technology. His research interests include planning systems, knowledge representations, world modeling, and architectures for autonomous systems.

Chris Scrapper works as a Computer Scientist in the Intelligent System Division of the National Institute of Standards and Technology. He is also a doctoral student attending George Washington University.