A Hierarchical World Model for an Autonomous Scout Vehicle

Tsai Hong[*], Stephen Balakirsky, Elena Messina, Tommy Chang, Michael Shneier
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

ABSTRACT

This paper describes a world model that combines a variety of sensed inputs and *a priori* information and is used to generate on-road and off-road autonomous driving behaviors. The system is designed in accordance with the principles of the 4D/RCS architecture. The world model is hierarchical, with the resolution and scope at each level designed to minimize computational resource requirements and to support planning functions for that level of the control hierarchy. The sensory processing system that populates the world model fuses inputs from multiple sensors and extracts feature information, such as terrain elevation, cover, road edges, and obstacles. Feature information from digital maps, such as road networks, elevation, and hydrology, is also incorporated into this rich world model. The various features are maintained in different layers that are registered together to provide maximum flexibility in generation of vehicle plans depending on mission requirements. The paper includes discussion of how the maps are built and how the objects and features of the world are represented. Functions for maintaining the world model are discussed. The world model described herein is being developed for the Army Research Laboratory's Demo III Autonomous Scout Vehicle experiment.

Keywords: Autonomous vehicle, path planning, sensory processing, 4D/RCS, hierarchical world model, Demo III

1. INTRODUCTION

The Experimental Unmanned Vehicle (XUV) developed for Demo III uses a control system designed in accordance with the 4D-Real-time Control System (4D/RCS)[1] hierarchical architecture which divides the system into perception, world modeling and behavior generation subsystems. This paper focuses on the design and functionality of the world modeling subsystem at two levels in the hierarchy—the Vehicle level and the Autonomous Mobility (AM) level. At each level in the hierarchy, the world model acts as a bridge between multiple sensory inputs and a behavior generation (path planning) subsystem for autonomous driving. The world model includes occupancy grids, or maps, and symbolic object representations at each level of the hierarchy. In addition, a number of functions are described that maintain the model. To understand the application more fully, the sensors and sensor-processing algorithms are presented briefly, and the information extracted from them is described. The map at each level of the hierarchy is used by a path planner module to compute safe and task-appropriate routes for the vehicle to travel to the limit of the terrain represented at that level[2].

The Autonomous Mobility (AM) level maintains a model of the area around the vehicle (a map) to 50 m from the vehicle, at a resolution of 40 cm per map cell. The sensory processing that takes place at this level combines information from multiple imaging sensors to construct and maintain the map. The map is used by a path planner to develop plans for vehicle motion out to about 50 m, with an update rate of about 5 Hz. The level above the AM level in the 4D/RCS hierarchy is the Vehicle Level. This level of the hierarchy contains a world model and behavior generation system that operates over a larger extent and lower resolution than the AM level. The level has a cycle time on the order of seconds instead of milliseconds, and operates on a map with a cell size of 5 m and an extent of 500 m. The Vehicle Level world model contains feature and elevation data from *a priori* digital terrain maps. In order to take advantage of these features, the world model at this level utilizes a different structure than that of the AM level and has different responsibilities. The Vehicle-Level world model (also known as the Layered World Model or LWM) is responsible for maintaining a model of the environment and for working with Behavior Generation (BG) to create and evaluate a planning graph. This cooperation includes determining the states that will become the nodes of the planning graph, determining how the

[*] Authors' email addresses: firstname.lastname@nist.gov.

planning graph is connected, and determining the cost of the graph edges. In order to accomplish these tasks, the LWM is equipped with a Knowledge Database (KD), a Plan Simulator (PS), and a Value Judgment (VJ) module.

Section 2 provides an overview of the hierarchical world model. Section 3 briefly discusses the sensors and algorithms used to provide information to the world model. Section 4 describes the map and object representations used by the world model and the functions used to maintain the model. Section 5 describes future work and discusses issues related to the world model representation. Finally, conclusions are presented in Section 6.

2. WORLD MODEL

The world model is the system's internal representation of the external world. It provides a central repository for storing sensory data in a unified representation, and decouples the real-time sensory updates from the rest of the system. The world model process has two primary functions:

1. To create a knowledge database (map) and keep it current and consistent. In this role, it updates existing data in accordance with inputs from the sensors, and deletes information no longer believed to be representative of the world. It also assigns confidence factors to all map data and adjusts these factors as new data are sensed. The types of information included in the map are state variables (e.g., time, position, orientation), system parameters (e.g., coordinate transforms, sensor to vehicle offsets, etc.), and lists or classes of sensed objects. The world model process also provides functions to update and fuse data and to manage the map (e.g. scrolling and grouping objects.)
2. To generate predictions of expected sensory input based on the current state of the world and estimated future states of the world. For the Demo III autonomous driving application, very little *a priori* information is available at the highest resolution in the hierarchy to support path planning between the vehicle's position and a final goal position. The world model therefore constructs and maintains all the information necessary for intelligent path planning[2]. At lower resolution levels, *a priori* information is supplied from maps of the area being modeled.

At the Autonomous Mobility level in the 4D/RCS hierarchy, the world model implementation fuses information from multiple sensors, including navigation sensors, ladar, and stereo vision. The navigation system provides information about the vehicle's current position, orientation, speed, velocity, etc. Data from the Ladar sensor (Section 3.1) includes a range image (32 rows x 180 columns) processed to provide an array in which each element contains the distance to a point in space. The vehicle is equipped with two pairs of stereo cameras. One provides color imagery, while the other provides infrared (FLIR) data. The information obtained by processing the stereo range information includes range and color (or intensity) at each point.

The world model incorporates a set of maps at multiple resolutions. Each map fuses sensory information and *a priori* knowledge into its occupancy-grid representation. Information at different hierarchical levels has different spatial and temporal resolution. The map is north oriented and scrolls as the vehicle moves. The various features are integrated over time, computing confidence and filtering out spurious false detections. On the Demo III vehicle, data from the various terrain sensors are fused in this mapping process. For each planning cycle, a copy of the obstacle map is rotated from a north-oriented into a vehicle-oriented map and is sent to the planner.

The primary use of the model data is to plan safe and efficient paths at each level of resolution. The path-planning module uses the map to select a locally optimal path from the current position to the commanded goal. For example, the highest resolution planner heuristically selects a path by starting with a web of potential path segments that extend out 20 m. The path is updated (replanned) approximately ten times per second[3]. There are two types of path segments: straight and curved. Curved segments extend 20 m from the vehicle. Each is a series of clothoid segments which are kinematically feasible based on the turn rate of the steering wheel. These paths are generated offline for different initial speeds and steering wheel positions. Straight path segments are used for the next (lower) resolution level, from 20 m to 50 m from the vehicle. Although not kinematically feasible, they are computationally simpler. Given that the path is recomputed frequently, an exact solution for more distant segments is not necessary. The planner selects the best combination of segments leading to the goal, using a cost function depending on its task. It does this by first pruning all segments blocked by obstacles. Then it searches through the remaining segments to find the path with lowest cost. Among the factors used in the cost function are terrain elevation, the presence or absence of obstacles, tree cover, the presence of tall grass areas, the relative distance between different path options, and commanded speed.

3. SENSORS AND SENSORY PROCESSING

Sensor processing algorithms use sensor data to compute vehicle position, range, obstacle lists, obstacle positions, and terrain information. The suite of sensors used in the mobility system include a Schwartz Electro-Optics (SEO) Scanning Laser Rangefinder (Ladar), a pair of color cameras for stereo vision, a stereo pair of Forward-Looking Infra-Red (FLIR) cameras, a stereo pair of monochrome cameras, a pan-tilt platform, a Global Positioning System (GPS) sensor, a force bumper that alerts the system to obstacles in the vehicle's immediate path, and an Inertial Navigation System (INS) sensor[†]. The Ladar and stereo camera sensors are described in Sections 3.1 and 3.2. All sensors are mounted on the vehicle, which is equipped with electric actuators on the steering, brake, transmission, transfer case, and parking brake. Feedback from the sensors provides the controller with engine rotations per minute, speed, temperature, fuel level, etc. Multiple navigation sensors are used. A Kalman filter [4] computes vehicle position and orientation using data from the inertial dead reckoning system and the carrier phase differential GPS unit.

3.1. Ladar Sensor

The Ladar sensor is mounted on a pan/tilt platform to increase its rather narrow 20° field of view (FOV). The range of the tilt motion is ± 30° resulting in an effective Ladar field of view of about 80°. Range data are read into an image array containing 32 rows and 180 columns. Using *a priori* knowledge about the location and orientation of the Ladar mounting on the vehicle, calibration factors, and vehicle position data, we transform the range information into position and orientation values in a world coordinate frame.

Obstacles are defined as objects that project more than some distance *d* above or below the ground plane (defined as the plane on which the wheels of the vehicle lie). Positive obstacles, which extend above the ground plane, are detected in the range images, while negative obstacles are detected in the world model map[5]. The positive obstacle detection algorithm scans column by column in the image, starting with a point known to be on the ground. An initial ground value is assigned at the location where the front wheels of the vehicle touch the ground, known from INS and GPS sensors. A pixel is labeled a positive obstacle if it rises high enough and abruptly enough from the ground plane. The results of the positive obstacle detection are shown in Figure 1. The figure on the left is a Ladar scene of a wall obstructed by a truck on the far right. The objects in the foreground are low poles. The figure on the right shows the objects detected as positive obstacles in this scene.

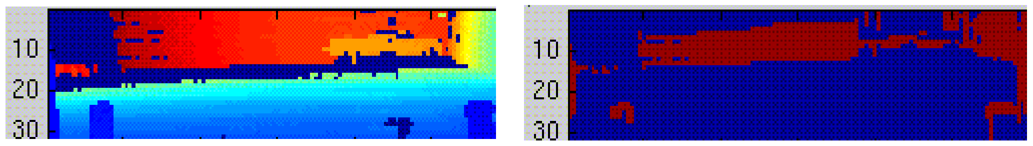


Figure 1 *Left: Raw ladar image. Right: Detected positive obstacles*

The negative obstacle detection algorithm maintains its own high-resolution ground map centered on the vehicle. This ground map contains all the projected ground pixels detected by the positive obstacle detection module. The algorithm first identifies groups of pixels in the range image that potentially correspond to a negative obstacle because they are below the ground level and are large enough[6] to prevent the vehicle from driving over them. For efficiency, the algorithm detects only the borders of negative obstacles.

After a group of pixels has been labeled as an obstacle, additional processing is performed to classify the obstacle type. The quality of the range data precludes more than a coarse classification, which currently identifies vegetation and ground (i.e., not vegetation). The approach is based on the method of Ojala, Pietikännen, and Harwood[7]. It makes use of two texture measures, Local Binary Patterns (LBP), and Contrast. The LBP values are combined with a contrast measure at each point, computed over the same 3_3 window. The contrast measure is computed as the difference between the

[†] Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the materials or equipment identified is necessarily best for the purpose.

averages of the range values of the pixels with ranges greater than the center pixel and those with ranges less than the center pixel value. This range contrast can be viewed as a measure of porosity of a volume. If the volume is sparsely filled, the contrast will be large, as in the case of grass. If the foliage is denser, such as for brush or thick shrubs, the contrast will be smaller, since the expected distance the laser will travel before hitting a surface will decrease. If the surface is solid, the contrast should be close to zero. The texture measures are computed after the obstacles are computed. First, the connected components of the obstacle image are found. For each obstacle, a two-dimensional histogram is computed from the texture measures applied to the pixels in the component. The histogram is used to determine the class to which the obstacle belongs. Classes are defined by models, created in a learning phase. The upper image in Figure 2 shows a raw Ladar image of a scene taken at Ft. Knox with tall grass and obstacles. The lower image shows the classification results.

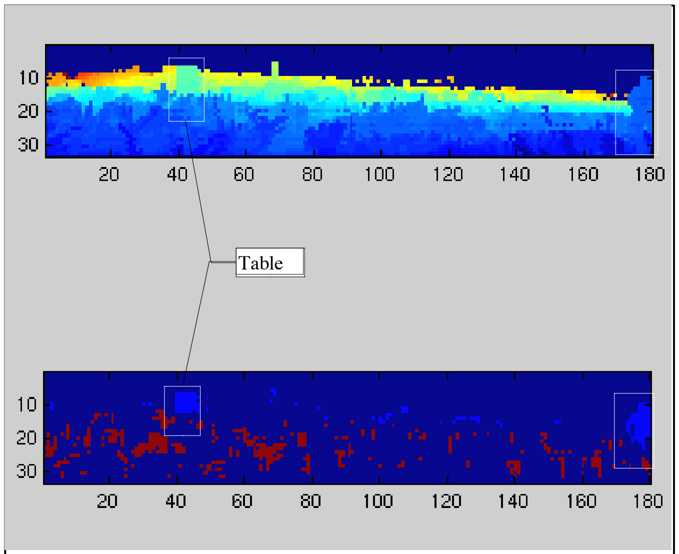


Figure 2 *Top: A range image from Fort Knox, with a table and a person in the image. Bottom: The results of classifying the obstacles in the scene. Red represents vegetation, while Blue represents non-vegetation objects that are still labeled as obstacles.*

3.2. Stereo Vision Sensors

Stereo vision provides another way of computing range information. The system is equipped with a color camera pair with a 60° FOV and a FLIR camera pair with a 40° FOV for night vision. The stereo system includes an iris controller; an image acquisition unit; a stereo range algorithm; positive and negative obstacle detection algorithms[8]; and a terrain classification algorithm[9].

A multiresolution, coarse to fine, approach is taken to determine correspondence between the left and right images. First, the two images are rectified to align their scanlines in order to increase processing efficiency. A difference of Gaussian image pyramid is then constructed, and image similarity is computed using a sum of squared difference measure for 7×7 windows over a fixed disparity range. The disparity is estimated, and bad matches are removed using consistency constraints. After smoothing, the pixels are transformed to three-dimensional points by triangulation. For each range image column, a set of obstacle detectors is applied to extract gaps and discontinuities in the range data that indicate non-traversable regions. Non-traversable regions are classified into either negative or positive obstacles. Negative obstacles are detected by checking for gaps in the range data followed by a range jump. This usually occurs when a ditch or hole exists. Positive obstacles are detected by checking for upward slanted edges in the range data, i.e., any upward protrusion out of the ground plane steep enough to be non-traversable or to cause a tip-over hazard.

Terrain classification is performed on color images taken from one of the stereo images. Consequently, the classification label is registered with the range image. Classification types currently include green vegetation, dry vegetation, soil/rock, ruts, tall grass, and outliers. The classification algorithm relies on color, and is based on Bayesian assignment. The class likelihood's are represented using a mixture-of-Gaussian model. The parameters of the model are estimated by training data using the Expectation Maximization algorithm.

4. WORLD MODEL: MAPS, OBJECTS AND FUNCTIONS

This section describes the organization of the world model in detail. Section 4.1 describes the map structure at the Autonomous Mobility level as well as the terminology used to define and classify objects and terrain in the map. The following section describes the structure at the Vehicle level. Section 4.3 describes the computational functions performed by the world model to maintain its representation. Section 4.4 describes the way the system evaluates the cost of a proposed plan, while section 4.5 explains how the plan simulator can be used for generating the costs of hypothetical paths.

4.1. Maps and objects at the Autonomous Mobility level

A modified form of Hebert's[10] grid obstacle map was adopted for representing obstacles in a way suitable for path planning and vehicle control. The map consists of a header and a square, two-dimensional array of cells (Figure 3). The map header contains information that is shared by all map grids. Table 1 describes the data fields.

The map is a two-dimensional array (301 x 301 cells) containing information extracted from processed sensor data. Figure 4a shows an image of the XUV on an unpaved road at Fort Knox; Figure 4b displays this scene as an elevation

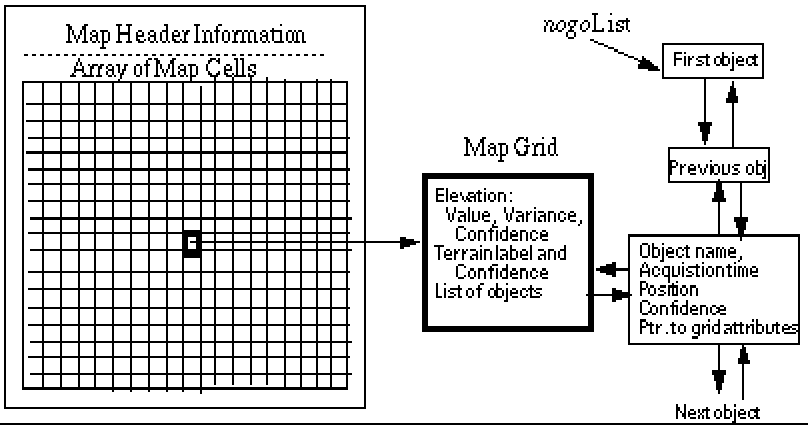
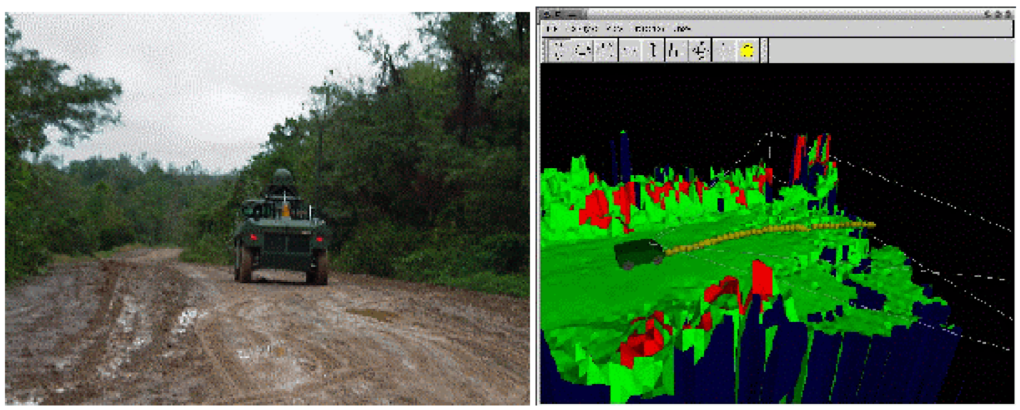


Figure 3 *World Model Map*

map constructed from the information in the world model. The position of the XUV is shown as an overlay on the map; the yellow path in front of the XUV represents the vehicle's planned path, and red areas represent unclassified obstacles.



(a) (b)

Figure 4 *(a) XUV at Ft. Knox (b) World Model elevation map of (a)*

Datum	Description
Grid size	The size of each map grid (cell) in meters.
Map size	The number of grids in each dimension of the map.
Map center	Grid center in NIU[‡] coordinates (north, east)
Grid center location	Grid center location in the array data structure of the Map
Vehicle position	Current (x,y,z) position of vehicle in NIU coordinates
Vehicle orientation	Current (roll, pitch, yaw) angles of vehicle in NIU coordinates
Predicted position	Predicted vehicle position at next computational cycle
Predicted rotation	Predicted vehicle orientation at next computational cycle
Predicted curvature	Predicted steering curvature at next computational cycle
Predicted speed	Predicted vehicle speed at the next computational cycle
Offset position	Offset between vehicle position UTM[§] coordinates and NIU coordinates

Table 1 *World Model Map Header*

Each cell in the map grid represents an area defined by the header grid size (currently 0.4 m × 0.4 m) and is marked with the time it was last updated. The total extent of the map is 120 m x 120 m. The information stored in a cell includes:

1. The average ground elevation height; the variance of the height; and a confidence measure reflecting the "goodness" of the elevation data.
2. A data structure describing the terrain covered by the cell. This includes a terrain label (e.g., tall grass, water, ruts, etc.), and a cost factor for determining the relative safety of traversing the cell. The terrain label represents the best estimate of the terrain type based on information fused over time. Each terrain type has an associated confidence.
3. A linked list structure describing the type of object viewed by the sensor. Examples of different types of objects include roads, buildings, fences, positive or negative obstacles, traversable or non-traversable regions, etc. Each object has a name, a position, a time stamp and a confidence measure. For each cell, objects are stored in a linked list containing pointers to the previous object in the list, the next object in the list, and a pointer back to the map grid describing the object attributes. Certain object labels are explicitly declared, e.g. a non-traversable region is flagged as a "*no go*" area.

4.2. Vehicle Level World Model and Knowledge Database

The Knowledge Database (KD) contains a very rich representation of the features of the outside world at a resolution and extent that is dictated by the level of the architecture where it resides. This information is stored in attribute layers, where each group of related features is represented as an independent layer. For example, in the Demo III application,

[‡] Navigation Interface Unit. The inertial navigation system measures orientation and distance traveled and combines the odometer and heading data to generate vehicle position data.

[§] UTM (Universal Transverse Mercater) coordinates are the earth's coordinates read from a Global Positioning System (GPS).

layers include an *a priori* layer that contains static knowledge about the environment and an obstacle layer that contains dynamic knowledge. The basic form of the layer is a combination of a regular, *n*-dimensional grid of cells that represents the system's discrete state space with regard to the layer's features and a database of specific feature instantiations. The dimensionality of the grid is set to appropriately capture the features being modeled. For example, in Demo III, the robot is constrained to travel on the ground plane, and the world positions are represented by an (x, y) pair. Therefore, the *a priori* layer is two-dimensional (position in the world), and the obstacle layer is three-dimensional (position in the world and time).

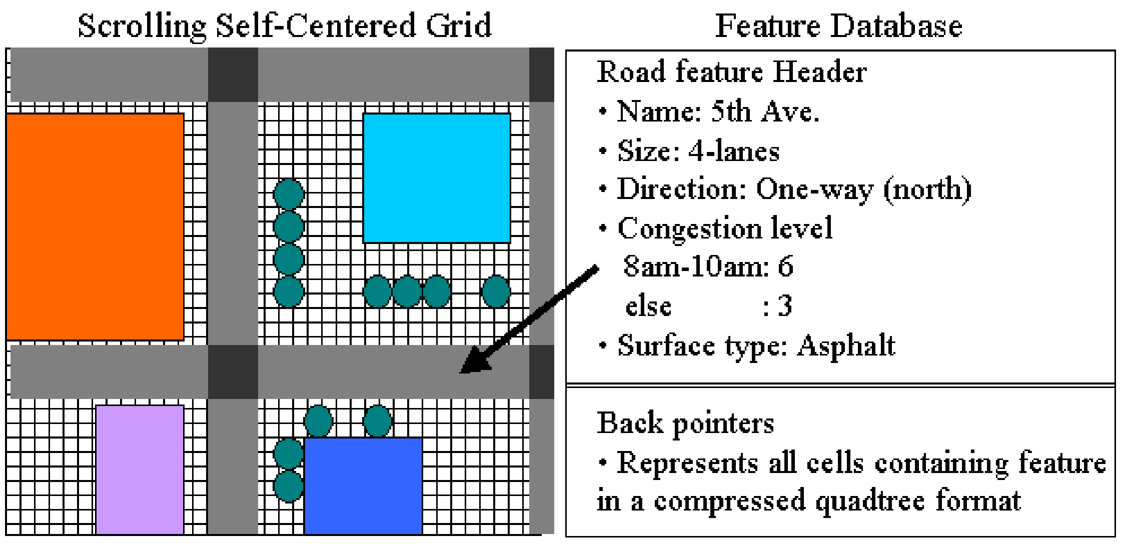


Figure 5 *Structure of a world model layer for the Vehicle Level.*

Each cell of the grid structure contains a set of flags that denotes which of that layer's possible features are contained in the cell, pointers to specific instantiations of each contained feature, and a planner cache list. Figure 5 shows a representation of the *a priori* layer for the Demo III system. This layer may be viewed as a conventional two-dimensional map, where each grid cell is denoted as some combination of road, forest, river, etc. Underlying this map is an object oriented database where an individual cell that includes the label “road'” would have a pointer to a specific road object that contains information about the road (speed limit, number of lanes, etc.). All cells that contain a piece of that specific road share this specific road object. In addition, as shown in Figure 5, the road object contains back-pointers to each cell that contains the road. The planner cache list is used to allow for operation in dynamic environments. Each time that a state is used as part of a state chain (where a state chain is a possible plan segment), that chain's identification is added to the cell's list. If a property of the state changes, perhaps due to new sensor information, all of the state chains that contain the cell are marked as “dirty”.

In order to contain the KD in a limited amount of online-memory, each instantiated feature is stored in a compressed format, and the KD layer is implemented as a scrolling, fixed size *n*-dimensional grid. For the scrolling grid, the system's current state is always maintained at the center of the grid. As the system's state changes, the grid is scrolled to maintain its centered position. This scrolling has the possibly undesirable effect of causing data to be lost off the edges as the layer scrolls. In order to prevent this loss of data, KD layers may implement a function that summarizes this data and passes it up to the world model at the next level of the hierarchy (which has a greater spatial extent at lower resolution). This allows dynamic information to propagate up the hierarchy and is the first of four ways in which the KD layers are populated with information.

The second population technique is that *a priori* information may be built-in to a specific layer. If it is small enough, the entire state-space for the feature class may be maintained in system memory. If not, the system may treat its file system as a data input sensor. As the system traverses the state-space, new areas are loaded in from files.

A priori information and information from other agents may also be populated into the KD layer by pulling information from higher levels of the hierarchy (population technique 3). This information will typically be of a lower resolution than the KD layer and may therefore not be useful for detailed plans. However, rough plans that are refined as sensor data become available are possible.

Real-time sensor reports are the final way to populate specific KD layers with both direct and derived knowledge. The sensor and associated processing are typically tied to the level of the hierarchy that best represents the particular sensor's resolution, spatial extents, and update frequency. This information may then be summarized and sent to higher levels of the hierarchy, or pulled down into lower levels.

The main user of the information stored in the KD is the VJ module. VJ participates in an information loop with the KD and the PS to extract information of interest and make decisions.

4.3. Maintaining and Updating the World Model at the Autonomous Mobility Level

The world model map must be maintained and updated in a timely manner. World model functions have been developed to scroll the map as the vehicle moves, to update map data, to fuse data from redundant sensors, and to extract and group information from the object lists. These functions are described in the following sections.

An efficient, scrolling, local map is used that updates as new sensor data arrive, while keeping the vehicle centered on the map. This approach has the advantage of minimizing grid relocation. No copying of data is done, only updating. When the vehicle moves out of the center grid cell of the map, the scrolling function is enabled. The scrolling function includes recentering the map and reinitializing the map borders. Because the map is vehicle-centered, only the borders of the map contain new regions that must be initialized. While the initialization information could be obtained from *a priori* maps, in the current implementation, the information in available *a priori* maps is too coarse (30 m resolution) for this level of the hierarchy, so the borders are initialized to zero, interpreted as “unknown”.

The map updating algorithm is based on the concept of confidence-based mapping described in Oskard[11]. In this algorithm, confidence measures increase or decrease linearly as the model receives updated information from the sensors. When a map cell receives a vote for a class such as an obstacle, an elevation measurement, a terrain classification, etc., the cell's confidence in that class is incremented by a predefined constant.

Three different types of data must be updated and fused: non-traversable ("*no go*") regions, elevation values, and terrain classifications. The confidence measure of a "*no go*" region is updated based on the obstacles detected from Ladar data[12], stereo range data, and the force bumper, and the results of classification from the color image classifier. The obstacle's confidence increases by predefined constants (i.e. a bumper obstacle constant, a stereo obstacle constant, and a Ladar obstacle constant). When the confidence measure exceeds a pre-defined threshold, the map grid is labeled "*no go*."

The elevation confidence of each grid is updated each computational cycle. The new elevation is updated by the weighted average of the current sensed elevation value and the accumulated elevation value:

$$Elevation_t = \frac{(w \times Elevation_i) + (Conf_{t-1} \times Elevation_{t-1})}{Conf_t}$$

$$\textbf{if}\ (Conf_t \le Maxvalue)\ \textbf{then}\ (Conf_t = Conf_{t-1} + W_i)$$

$$\textbf{else}\ (Conf_t = MaxValue)$$

$Conf_t$ is the confidence measurement of the elevation value at time *t*. W_i is the confidence measurement for $sensor_i$. $Elevation_i$ is the current elevation value read from $sensor_i$ at time *t*, and $Elevation_t$ is the estimated elevation value at time *t*.

Conversely, when a map grid is labeled "ground", the "*no go*" confidence decreases. Decreasing a confidence measure reduces the effects of noise in the sensor data, and results in a fairly static environment.

A map grid is classified into a terrain type and an object type. Examples of terrain types are ground, tall grass, or water. Examples of object types are rocks, bushes, trees, and ruts. The confidence of these classes is updated based on the results of both Ladar classification algorithms[13] and stereo classification algorithms[9]. Confidence values increase by a factor determined by sensor characteristics, which are learned off-line by analyzing and testing data collected for this purpose. Confidence values are used as cost factors in determining the traversablility of a grid.

The map cell contains a pointer to the object list, and each list node contains a pointer to the previous entry, a pointer to the next entry, and a pointer to the originating (attribute) grid (Figure 3). "*No go*" regions are an example of an object group. The object list also contains the object's position in the grid, the object type, the time stamp, and the confidence measure. The attribute pointer currently points back to the map grid location. This object list is used to compute the object's properties, i.e., velocity, size, and moments, which can be used for object recognition. This data structure is very similar to the one described by Shneier[14] in which objects are indexed spatially by a pointer associated with each node in an octree.

4.4. Value judgment

A single Value Judgment (VJ) module exists for each node within each level of the RCS hierarchy. The VJ module is responsible for combining the information from the multiple knowledge database layers and plan simulators into a single response that is sent back to the BG module. These responses must be formulated in such a way that they cause the system behavior to be consistent with the supervisor's commanded goals and objectives. Collecting all of the individual results and applying rules to collapse these results into a form that is usable by BG accomplishes this. BG then uses these values to determine which system states will become planning graph nodes, which nodes will be connected by edges, and the cost of individual edges.

All of the tasks mentioned above are carried out in the same manner. First, each layer of the world model is queried for partial results. The layers apply layer-specific constraints and rules and return candidate results for consideration. The rules applied by the layers may be fixed or changed on a planning cycle or region basis. The VJ then combines these results by applying further global constraints (again, fixed or dynamic), and passes the resulting answers onto BG.

For example, in node selection for the Demo III system, the *a priori* data layer passes a 2-D grid of quantized locations to VJ. These locations may cover the entire ground surface except for areas of dense trees for off-road driving, or may only cover road surfaces for on-road driving. The global rules in the VJ then eliminate any state received from another layer that was not also received from the *a priori* layer.

The result of applying the VJ rules directly affects the manner in which system goals are met. Therefore, several different rule sets may exist that will cause the system to display different behaviors, or proceed to a goal in different ways. The rule set that will elicit behaviors with the closest match to the system objectives is activated for each BG run.

4.5. Plan Simulator

Each layer of the KD is equipped with its own plan simulator. The plan simulator is a mini-expert system that works with the VJ module to determine the cost of graph edges. It is capable of fully simulating the state transitions (at the resolution of its level in the RCS hierarchy) that make up a graph edge with respect to the layer's features. The graph edge that is sent to the plan simulator for evaluation consists of a chain of two or more possibly non-adjacent states (the nodes of the graph). If the states are non-adjacent, the plan simulator will evaluate the shortest, in terms of state transitions, (not necessarily optimal) path between states.

Each simulator performs two evaluations. The first evaluates the conformance of the proposed trajectory with rules that govern its feature set. The second examines the resources that are consumed or produced by the proposed trajectory. To illustrate this, we once again return to our Demo III system and the *a priori* map layer for a mobile robot. In this case, the graph nodes represent (x,y) locations in the world and the plan simulator simulates the robot traversing the graph edge (it simply moves the robot from state to state in a straight line). The KD layer may return such information as the percentage of the edge that lies on a road (conformance to road). In addition, the layer may return the more detailed information of the percentage of the edge that lies on secondary roads, or the percentage that lies on a particular secondary road.

By simulating each state transition in the chain, we gain the benefit of directionality in our cost computation process. The importance of directionality may be seen in the case of a mobile robot traversing one-way streets. Travel in one direction would conform to the layer's feature set while travel in the opposite direction would not.

The second simulation that is performed simulates resource production and consumption. These resources may also be viewed as preconditions (a resource must be available for consumption) and postconditions (a resource is produced or made available as a result of the operation) of the graph edge. A simple example of this would be our mobile robot transitioning from a non-road state to a road state. This transition may require the road-tracking module to be active prior to its execution. These resource requirements are sent to the value judgment module for integration with the other layer's resource requirements.

5. Future Work and Discussion

This paper has described the implementation of a sophisticated world modeling system, but much research remains to be done to add to its capabilities and performance. One area of future research involves tracking moving obstacles because confidence-based mapping may not be adequate for this task. Currently, a hypothesis/prediction model for predicting the motion of moving obstacles is not implemented. Such an approach would be useful for maintaining knowledge of local obstacles at the sensory processing level. This knowledge could be used to improve detection accuracy and could be used to detect moving obstacles.

The use of *a priori* maps would enhance the scope of the current world model. Currently survey maps, GPS maps, and aerial maps contain fine resolution and significant information about existing topology and structures. In order to take advantage of this knowledge, research is needed to register *a priori* maps with our sensor-generated map. Also, higher resolution *a priori* maps must be generated; current geological survey maps are too coarse for autonomous driving applications.

The current implementation of fusion is a straightforward combination of information from different sensors, with each sensor having weights representing its ability to measure different characteristics of the world. We also include a temporal decay that weights newer information more than older information. The weights are currently fixed but should be made dynamically adjustable as the sensors become more self-aware and perhaps as a function of how well the predicted state of the world matches the sensed state. When we incorporate *a priori* knowledge into the world model, we will also need some way of weighting it. This will depend on how well the *a priori* data and the sensed information are registered, and how old the *a priori* data are.

Prediction can make the sensory system appear much more intelligent by providing windows of attention for each sensor and indications of what features are expected to appear in the windows. A prediction capability based initially on *a priori* data but evolving as sensed data are overlaid would clearly take advantage of anything known about the local world, and should show a major performance improvement over the same system running without *a priori* information. Another improvement would be the ability to predict where and what a sensor should expect to observe some feature *in the format and coordinate system of that sensor*. We have already implemented the capability to predict where features can be expected to appear in a color image given that the features were detected by a ladar sensor and placed in the world model. The computations are not too time consuming, consisting largely of coordinate transformations to determine where windows should be placed, and informing the sensor what to search for in each window.

Mode switching is another capability that the world model could provide if it had either relevant *a priori* information or the vehicle had previously traversed the same region of terrain and has stored the features it saw there. The sensors can be much more effective if they have contextual information about the region they are observing. This can be general information, such as time of day or type of weather. It can also be more specific, such as informing the sensors that a road is expected in some area of its field of view, thus allowing the sensor to perform a localized search and perhaps switch into a road-following mode. Together with feature prediction, mode switching can enhance the capabilities of the sensors without incurring too high a processing penalty.

At this point, behaviors generated by the vehicle primarily address path generation and execution based on user preferences and mobility constraints (i.e., avoiding obstacles, staying on road). The soldier applies military tactics techniques and procedures to make high level decisions about the vehicle's mission and how to accomplish it. It is highly desirable to incorporate military expertise into the *a priori* knowledge base of the vehicle so that it may have the capacity to make its own plans or, in the event of an unexpected situation arising while it executes its mission, be able to replan consistent with military doctrine. This higher-level of planning will involve greater use of symbolic representation of knowledge (e.g., rules or formal logic). The world model will also have to take into account multiple vehicles belonging to the same section, so the dimensionality of the planning space (described for the vehicle level above) will grow.

ACKNOWLEDGMENT

This research is a jointly sponsored effort which leverages technology from the NIST Intelligent Machines Initiative program and receives financial support from the Army Research Lab DemoIII UGV program and the AUTONAV research project agreement between the German Ministry of Defense and the US DoD.

REFERENCES

1. Albus, J. S. and Meystel, A., *Engineering of Mind: an introduction to the science of intelligent systems* New York: John Wiley and Sons, 2001.

2. Lacaze, A., Albus, J. S., and Meystel, A. Planning in the Hierarchy of NIST-RCS for Manufacturing. Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective. Gaithersburg, MD, 1996.

3. Murphy, K., Abrams, M., Balakirsky, S., Coombs, D., Hong, T., Legowik, S., Chang, T., and Lacaze, A. Intelligent Control for Unmanned Vehicles. Proceedings of the World Automation Congress Conference (WAC 2000).

4. Kalman, R. E., "A new approach to linear filtering and prediction problems," *Transactions ASME Series D, J, Basic Engineering*, vol. 82 pp. 35-45, 1960.

5. Chang, T., Hong, T., Legowik, S., and Abrams, M. Concealment and Obstacle Detection for Autonomous Driving. Proceedings of the Robotics & Applications 1999 Conference, Santa Barbara, CA.

6. Hong, T., Abrams, M., Chang, T., and Shneier, M. O., "An Intelligent World Model for Autonomous Off-Road Driving," *Computer Vision and Image Understanding*, 2000.

7. Ojala T., Pietikainen, M., and Harwood, D., "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, vol. 29 pp. 51-59, 1996.

8. Matthies, L., Kelly, A., and Litwin, T. Obstacle Detection for Unmanned Ground Vehicles: A Progress Report. 1995. Jet Propulsion Lab.

9. Belluta, P., Manduchi, R., Matthies, L., Owens, K., and Rankin, A. Terrain Perception for Demo III. Proceedings of the Intelligent Vehicles Symposium 2000. Dearborn, Michigan.

10. Hebert, M., Thorpe, C., and Stentz, A., *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon* Kluwer Academic Publishers, 1997.

11. Oskard, D. N., Hong, T., and Shaffer, C. A. Real-Time Algorithms and Data Structures for Underwater Mapping. Proceedings of the SPIE Advances in Intelligent Robotics Systems Conference. 1988. Boston, MA.

12. Hong, T., Legowik, S., and Nashman, M. Obstacle Detection and Mapping System. NISTIR 6213. 1998.

13. Hoffman, R. and Jain, A. K., "Segmentation and Classification of Range Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 608-620, Sept.1987.

14. Shneier, M. O., Kent, E. W., and Mansbach, P. Representing Workspace and Model Knowledge for a Robot with Mobile Sensors. Proceedings of the 7th International Conference on Pattern Recognition. 1984. Montreal, Canada.