Intelligent Control of Mobility Systems

James Albus, Roger Bostelman*, Raj Madhavan, Harry Scott, Tony Barbera, Sandor Szabo, Tsai Hong, Tommy Chang, Will Shackleford, Michael Shneier, Stephen Balakirsky, Craig Schlenoff, Hui-Min Huang, and Fred Proctor

Intelligent Systems Division, National Institute of Standards and Technology (NIST), 100 Bureau Drive, Mail Stop 8230, Gaithersburg, MD 20899-8230, USA

1 Introduction

The National Institute of Standards and Technology (NIST) Intelligent Control of Mobility Systems (ICMS) Program provides architectures and interface standards, performance test methods and data, and infrastructure technology needed by the U.S. manufacturing industry and government agencies in developing and applying intelligent control technology to mobility systems to reduce cost, improve safety, and save lives. The ICMS Program is made up of several areas including: defense, transportation, and industry projects, among others. Each of these projects provides unique capabilities that foster technology transfer across mobility projects and to outside government, industry and academia for use on a variety of applications. A common theme among these projects is autonomy and the Four Dimensional (3D + time)/Real-time Control System (4D/RCS) standard control architecture for intelligent systems that has been applied to these projects.

NIST's Intelligent Systems Division (ISD) has been developing the 4D/RCS [1, 2] reference model architecture for over 30 years. 4D/RCS is the standard reference model architecture that ISD has applied to many intelligent systems [3–5]. 4D/RCS is the most recent version of RCS developed for the Army Research Lab (ARL) Experimental Unmanned Ground Vehicle program. ISD has been applying 4D/RCS to the ICMS Program for defense, industry and transportation applications.

The 4D/RCS architecture is characterized by a generic control node at all the hierarchical control levels. Each node within the hierarchy functions as a goal-driven, model-based, closed-loop controller. Each node is capable of accepting and decomposing task commands with goals into actions that accomplish task goals despite unexpected conditions and dynamic perturbations in the world. At the heart of the control loop through each node is the world model, which provides the node with an internal model of the external world. The fundamental 4D/RCS control loop structure is shown in Fig. 1.

The world model provides a site for data fusion, acts as a buffer between perception and behavior, and supports both sensory processing and behavior generation. In support of behavior generation, the world model provides knowledge of the environment with a range and resolution in space and time that is appropriate to task decomposition and control decisions that are the responsibility of that node.

The nature of the world model distinguishes 4D/RCS from conventional artificial intelligence (AI) architectures. Most AI world models are purely symbolic. In 4D/RCS, the world model is a combination of instantaneous signal values from sensors, state variables, images, and maps that are linked to symbolic representations of entities, events, objects, classes, situations, and relationships in a composite of immediate experience, short-term memory, and long-term memory. Real-time performance is achieved by restricting the range and resolution of maps and data structures to what is required by the behavior generation module at each level. Short range, high resolution maps are implemented in the lower levels, with longer range, lower resolution maps at the higher levels.

A world modeling process maintains the knowledge database and uses information stored in it to generate predictions for sensory processing and simulations for behavior generation. Predictions are compared with

*Corresponding author, roger.bostelman@nist.gov

J. Albus et al.: *Intelligent Control of Mobility Systems*, Studies in Computational Intelligence (SCI) **132**, 237–274 (2008)
www.springerlink.com

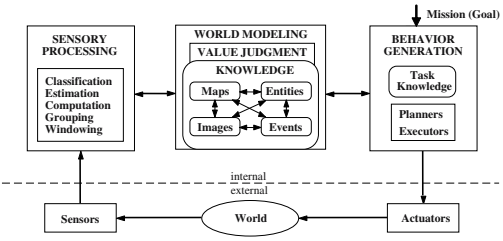

Fig. 1. The fundamental structure of a 4D/RCS control loop

observations and errors are used to generate updates for the knowledge database. Simulations of tentative plans are evaluated by value judgment to select the "best" plan for execution. Predictions can be matched with observations for recursive estimation and Kalman filtering. The world model also provides hypotheses for gestalt grouping and segmentation. Thus, each node in the 4D/RCS hierarchy is an intelligent system that accepts goals from above and generates commands for subordinates so as to achieve those goals.

The centrality of the world model to each control loop is a principal distinguishing feature between 4D/RCS and behaviorist architectures. Behaviorist architectures rely solely on sensory feedback from the world. All behavior is a reaction to immediate sensory feedback. In contrast, the 4D/RCS world model integrates all available knowledge into an internal representation that is far richer and more complete than is available from immediate sensory feedback alone. This enables more sophisticated behavior than can be achieved from purely reactive systems.

A high level diagram of the internal structure of the world model and value judgment system is also shown in Fig. 1. Within the knowledge database, iconic information (images and maps) is linked to each other and to symbolic information (entities and events). Situations and relationships between entities, events, images, and maps are represented by pointers. Pointers that link symbolic data structures to each other form syntactic, semantic, causal, and situational networks. Pointers that link symbolic data structures to regions in images and maps provide symbol grounding and enable the world model to project its understanding of reality onto the physical world.

Figure 2 shows a 4D/RCS high level diagram duplicated many times, both horizontally and vertically into a hierarchical structure as applied to a single military vehicle (lowest level) through an entire battalion formation (highest level). This structure, now adopted as a reference model architecture for the US Army Future Combat System, among other organizations, could also be applied to civilian on-road single or multiple vehicles as information could be passed from one vehicle to the next or to highway communication and control infrastructure.

This chapter will briefly describe recent project advances within the ICMS Program including: goals, background accomplishments, current capabilities, and technology transfer that has or is planned to occur. Several projects within the ICMS Program have developed the 4D/RCS into a modular architecture for intelligent mobility systems, including: an Army Research Laboratory (ARL) Project currently studying on-road autonomous vehicle control, a Defense Advanced Research Project Agency (DARPA) Learning Applied to Ground Robots (LAGR) Project studying learning within the 4D/RCS architecture with road following application, and an Intelligent Systems Ontology project that develops the description of intelligent vehicle behaviors. Within the standards and performance measurements area of the ICMS program, a Transportation Project is studying components of intelligent mobility systems that are finding their way into commercial crash warning systems (CWS). In addition, the ALFUS (Autonomy Levels For Unmanned Systems) project determines the needs for metrics and standard definitions for autonomy levels of unmanned systems. And a JAUS (Joint Architecture for Unmanned Systems) project is working to set a standard for interoperability between components of unmanned robotic vehicle systems. Testbeds and frameworks underway at NIST include the PRIDE (Prediction in Dynamic Environments) framework to provide probabilistic predictions of a moving object's future position to an autonomous vehicle's planning system, as well as the

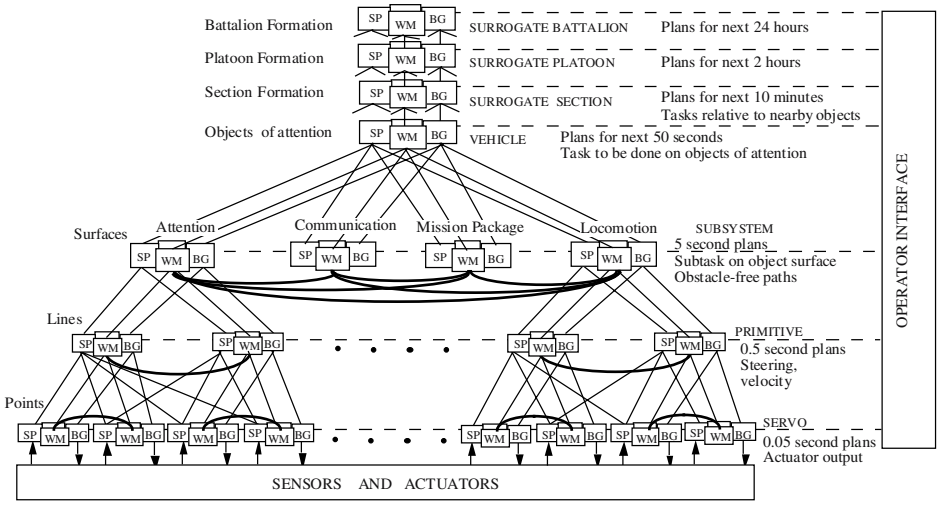

Fig. 2. The 4D/RCS Reference Model Architecture showing multiple levels of hierarchy

USARSim/MOAST (Urban Search and Rescue Simulation/Mobility Open Architecture Simulation and Tools) framework that is being developed to provide a comprehensive set of open source tools for the development and evaluation of autonomous agent systems. A NIST Industrial Autonomous Vehicles (IAV) Project provides technology transfer from the defense and transportation projects directly to industry through collaborations with automated guided vehicles manufacturers by researching 4D/RCS control applications to automated guided vehicles inside facilities. These projects are each briefly described in this Chapter followed by Conclusions and continuing work.

2 Autonomous On-Road Driving

2.1 NIST HMMWV Testbed

NIST is implementing the resulting overall 4D/RCS agent architecture on an ARL High-Mobility Multipurpose Wheeled Vehicle (HMMWV) testbed (see Fig. 3). Early work has focused on the lower agent control modules responsible for controlling the speed, steering, and real-time trajectory paths based on sensed road features such as curbs. This effort has resulted in sensor-based, on-road driving along dynamically-smooth paths on roadways and through intersection turns [6]. Future work includes the implementation of selected driving and tactical behaviors to further validate the knowledge set.

NIST has put in place several infrastructural elements at its campus to support the intelligent vehicle systems development described above. An aerial survey was completed for the entire site, providing high resolution ground truth data. A GPS base station was installed that transmits differential GPS correction data across the site. Testbed vehicles, equipped with appropriate GPS hardware, can make use of these corrections to determine their location in real-time with an uncertainty a few centimeters. This data can be collected and compared to the ground truth data from the survey to make possible extensive vehicle systems performance measurements of, for example, mobility control and perception components. In addition, lane markings consistent with the DOT Manual on Uniform Traffic Control Devices, have been added to parts of the campus to support development of perception capabilities required for urban driving.

Fig. 3. ARL/NIST HMMWV Testbed showing several sensors mounted above the cab

Fig. 4. Sensor suite mounted on the HMMWV cab

Perception

The perception in an autonomous mobile robot provides information about the world that enables a mobile robot to autonomously navigate its environment and react to events or changes that influence its task. The goal of Perception is to build and maintain the internal representation of the world (World Model) that the behavior generation components of the mobile robot can use to plan and execute actions that modify the world or the robot's position in the world. Since the world in general is not static, the perception algorithms must update the internal model rapidly enough to allow changes, such as the positions of moving objects, to be represented accurately. This places constraints on the sensors that can be used and on the processing that can be applied.

This section will address the perception aspects of an autonomous mobile robot under the ARL project. Under this project, we developed and demonstrated solutions to a number of perception problems in autonomous navigation. In order to achieve goals in supporting navigation through real-time algorithms, color cameras, FLIR (forward looking infrared), LADARs (laser detection and ranging) (see Fig. 4) are used and the following topics are investigated and implemented:

Sensors, Sensor Registration, and Sensor Fusion: Sensors in use include color cameras, FLIR and scanning and imaging LADARs. These sensors have to be registered so that data from the sensors can be combined. Registration involves accurately locating the sensors relative to each other and time- and position-stamping the outputs of each sensor and combined with vehicle motion. Sensor fusion allows a richer description of

the world and is particularly useful in associating range information with data from a two-dimensional, monocular color camera.

Obstacle Detection: Obstacles may be features that lie above the ground or holes in the ground. They need to be detected well before the vehicle reaches them and to know the true size of an obstacle to avoid it. Hence a three-dimensional analysis is required. This can be obtained from LADAR or stereo sensors, while a color camera may provide the best data to identify the obstacle.

Feature Detection: To build up a rich description of the world, a variety of features need to be recognized. These include roads, road signs, water, other vehicles, pedestrians, etc. Special purpose processing is needed for each of these features, which must not only be detected, but tracked while they are within the immediate vicinity of the robotic vehicle.

Performance Evaluation: Sensory processing plays a critical part in keeping the vehicle operating safely. Evaluating the performance of the sensory processing algorithms, which involves algorithm testing in realistic scenarios, provides a way to ensure that they work correctly and robustly enough in the real world.

The world model is used in this project as a basis for temporal fusing the extracted feature from the perception algorithms on different sensors and producing an internal world model. The world model contains a representation of the current state of the world surrounding the vehicle and is updated continually by the sensors. It acts as a bridge between perception and behavior generation by providing 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 therefore constructs and maintains all the information necessary for intelligent path planning.

2.2 4D/RCS Task Decomposition Controller for On-Road Driving

The 4D/RCS design methodology has evolved over a number of years as a technique to capture task knowledge and organize it in a framework conducive to implementation in a computer control system. A fundamental premise of this methodology is that the present state of the task sets the context that identifies the requirements for all of the support processing. In particular, the task activity at any time determines what is to be sensed in the world, what world model states need to be evaluated, which situations need to be analyzed, what plans should be invoked, and what behavior generation knowledge needs to be accessed. This view has resulted in the development of a design methodology that concentrates first and foremost on a clear understanding of the task and all of the possible subtask activities.

The application of this methodology to the on-road driving task has led to the design of a 4D/RCS control system that is formed from a number of agent control modules organized in a hierarchical relationship where each module performs a finer task decomposition of the goal it receives from its supervising module. Each of these control modules receives a goal task/command, breaks it down into a set of simpler subtasks, determines what has to be known from the internal world model to decide on the next course of action, alerts the sensory processing as to what internal world objects have to have their states updated by new sensory readings/measurements, and evaluates the updated state of the world model to determine the next output action.

An on-road driving 4D/RCS control hierarchy is shown in Fig. 5, which is an application of multiple levels of the 4D/RCS reference model architecture diagram shown in Fig. 2. Here, over 170 intermediate level subtask commands have been identified. These are shown as input commands to the corresponding control module where they execute. A further expansion of these commands is shown at the Vehicle Manager and the Mobility agent control modules. Their commands are underlined in Fig. 5 and the corresponding set of possible plans that can be selected are listed under each underlined command. For example, at the Mobility module, the **FollowRoad** command can select a plan to **PassVehInFront**, or **DriveOnTwoLaneRd**, or **DriveOnMultiLaneRd**, or **PullOntoRoad**, etc. depending on the present world state. Each of these plans is a separate state table describing this task behavior with an appropriate set of rules. Each of these rules would have a branching condition/situation from which are derived detail world states, objects, and sensing resolutions. An example of the command for each module at a particular instant in time is shown in Fig. 6: a **GoTo *PostOffice*** command to the Destination Manager might result in a **GoOn *SouthDr* TurnLeftOnto *ServiceDr*** command to Vehicle Manager resulting in a **TurnRightAtInterTo_*ServiceDr*** command to

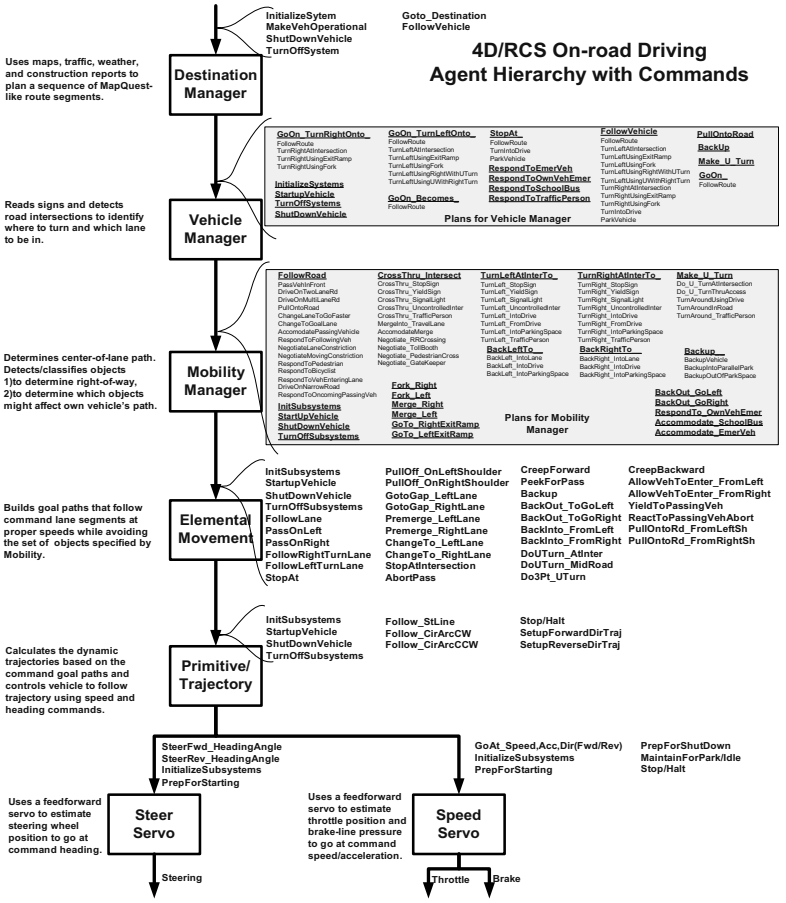

Fig. 5. 4D/RCS on-road driving control hierarchy with over 170 identified intermediate level subtask commands/plans that decompose the **GoTo_*Destination*** command all the way down to the instant-by-instant steer and speed/acceleration commands to control the vehicle

Mobility Manager resulting in a **FollowRightTurnLane** command to Elemental Movement resulting in a **Follow_CirArcCW** goal path command to Primitive/Trajectory resulting in a **SteerFwd_*HeadingAngle*** command to the Steer Servo module and a **GoAt_*Speed,Acc,Fwd*** command to Speed Servo module. With each of these control modules in Fig. 6, a summary view of the road network data structure is also pictured. The commands (goal/action-verbs) in this example are emphasized by a red font with parameters shown in green. There are, of course, many more parameters (not shown here) associated with each of these commands that identify specific road and lane parameters, and objects of interest that can affect right-of-way and own vehicle motion paths. At each control cycle, each of the agent control modules executes as a state machine where its input state is defined by the input command from its supervisor module, status from its subordinate module and the present state of its view of the world model. If this input state has changed from the last cycle,

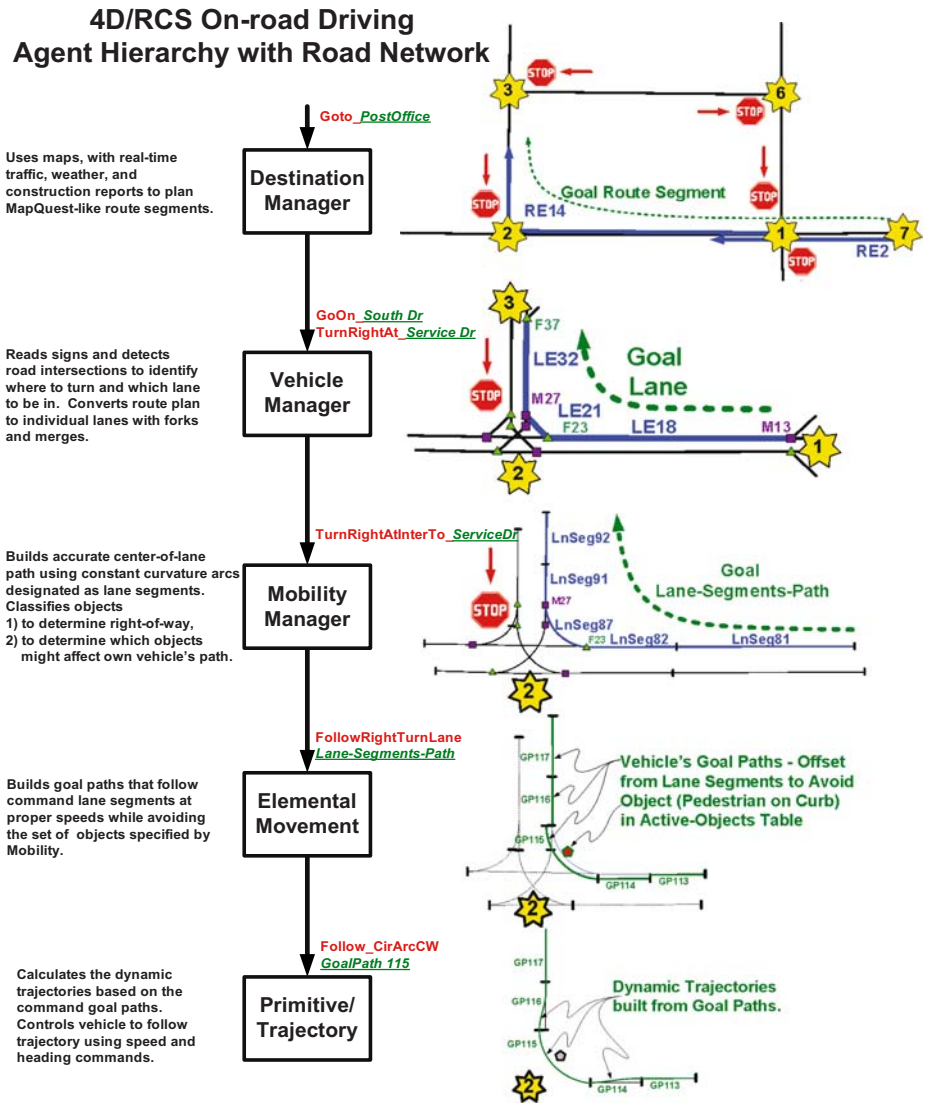

Fig. 6. Upper 4D/RCS agent control modules of an on-road driving control hierarchy summarizing the road network interactions for each control module. Each lower module processes a more detailed representation of the vehicle's goal path

then the control module transitions to a new state and may change the output command to a subordinate module, and/or the status to the supervisor module, along with changes to the world model.

The following outline will detail activities at each of these agent control modules summarizing the level and type of knowledge and data each process. This outline uses a standardized format where each module is described by a five part section consisting of summary of its responsibilities and actions including the world model data generated, its input command from the supervisor control module, its input from the world model, its output command to its subordinate control module, and its output status. Again, each command name will be highlighted in red (with parameters in green). Road network data structure elements will be highlighted in blue.

Destination Manager Control Module

Responsibilities and Actions: This module's primary responsibility is to plan the route to get to a single commanded destination, which may include passing through specified intermediate control points. This is route planning at the level that determines which road to follow and which intersections to turn at. At any time, real-time inputs concerning weather, traffic, construction, civil situations, etc. can cause a re-planning/re-sequencing of **route elements** used to define the route.

*Retrieves and builds a sequence of **route elements** to specify a **route segment**.*

This module is commanded to go to a ***destination***, which is described by not only the destination point itself but also by a list of possible intermediate control points so that the supervisor module can control the shape of the solution path. The input command further constrains the route to be planned to arrive at the destination by a specified time and to be prioritized by the shortest distance, or time, or the least number of turns or controlled intersections. As a result, this module has the responsibility to plan the sequence of the **route elements** (see Fig. 7) that meet the commanded constraints. It creates multiple sets of possible sequences of **route elements** and evaluates them against the commanded constraints to select a plan set to execute. It is continually updating the attributes of the **route elements** based on the real-time reports of weather, traffic, construction, civil situations, and accidents. Anytime new real-time inputs are available, this module re-plans the sequence of **route elements**. It processes present experiences to create a history of average effects of these parameters versus time of day, day of week, time of year to build more accurate estimates of the **route elements** availability and average speeds. From this, over time, it can improve the accuracy of its estimations and its planned routes. It commands the Vehicle control module to execute a ***route segment***, which is a sequence of **route elements**, all of which pass through each intervening intersection except the last route element, which specifies a turn onto a different road. When the subordinate control module indicates it is nearing completion of its commanded route segment, this module re-plans its set of **route elements** and groups the next sequence of **route elements** that represents the next ***route segment*** and issues this command to the subordinate module.

Input-Command: A command to **Goto *Post Office***, which is the form of a ***Goal Destination*** specification. The associated command data includes the parameters of the final goal point along with a sequential list of any intermediate points (control points) to pass through (e.g., **Goto *Post Office*** (from the present position which is the school) by way of the intersection at Service Dr and Research Dr (a control point)) along with a specific time to be at this goal destination (e.g., arrive at Post Office by 10:00 a.m.). Additionally, the command specifies that a route is to be planned that prioritizes on the shortest time, or the shortest distance, or the least number of turns, or least number of controlled intersections.

Input-World Model: Present estimate of this module's relevant map of the **road network** – this is a map at the level of **route elements** (an interconnecting single direction of travel between two intersections with a direction pointing out of the second intersection (e.g., turn left, or pass through, or turn right). Each **route element** is a unique possible path for a vehicle to travel. Each is assigned a number of attributes that identify its availability (i.e., is it available for a vehicle to travel on it), time constraints on its availability (not available from 7:30 to 9:00 a.m., such as might apply to a left turn), average travel speeds with time constraints (e.g., 35 kph from 7:00 p.m. to 7:30 a.m., 15 kph from 7:30 a.m. to 7:00 p.m.), average travel speeds with weather constraints (35 kph clear weather, 20 kph in rain, 15 kph in snow), average travel speeds

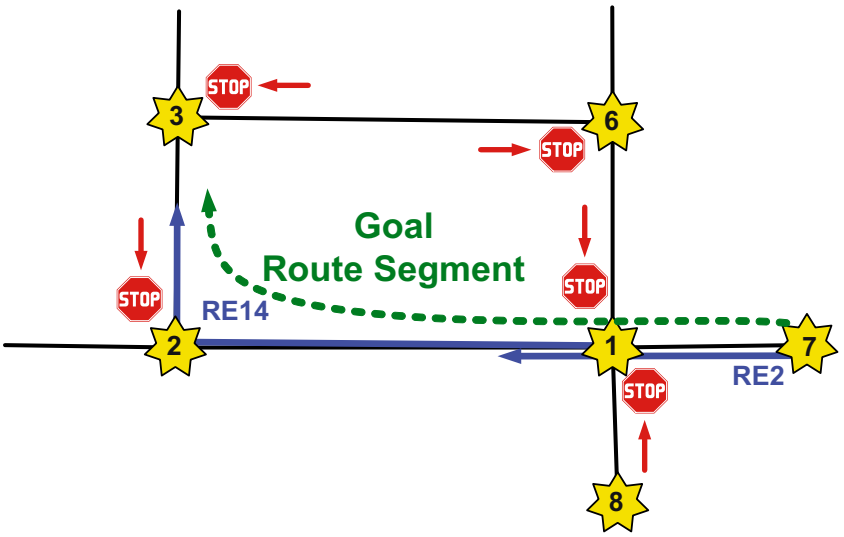

Fig. 7. Route Elements (RE) are route representations that connect two Map Points (which are either intersections or boundary points of the map and are labeled by *yellow stars*) and indicate a direction away from the end Map Point. Here Route Element 2 (RE2) starts at Map Point 7 and ends at Map Point 1 heading in a direction straight through the intersection at Map Point 1 towards Map Point 2. RE14 starts at Map Point 1 and ends at Map Point 2 heading in a direction of a right turn at the intersection towards Map Point 3. The Destination Manager control module plans a route as the sequence of these route elements that best meet the constraints of minimum distance, minimum time, or minimum number of turns

with traffic constraints (35 kph normal traffic, 25 kph in moderate traffic, 15 kph in heavy traffic) conditions that make it unavailable (e.g., unavailable in heavy rain – has a creek that floods, unavailable in snow or ice because has too steep of a hill, unavailable because of accident report or construction report or civil situation report such as a parade or demonstration). Most of the real-time information is usually received from remote sensing since most of the route is beyond own vehicle's sensing capabilities, but own vehicle's sensing can contribute in situations like observing an accident in the road ahead which has not yet been reported on the radio.

World model road network data includes a set of history attributes that store average times to traverse each route element under the various conditions, which should improve planning estimates.

Output-Command: Goal ***Route Segment*** which is a Map Quest-like command **GoOn** ***SouthDr*** **TurnRightAt** ***ServiceDr***, which is basically a command to follow a specified road until you have to turn onto a different road.

Associated command data: Sequential list of the **route elements** that combined make up the specified ***route segment*** command.

Output-Status: Present state of goal accomplishment (i.e., the commanded ***goal destination***) in terms of executing, done, or error state, and identification of which intermediate control points have been executed along with estimated times to complete the remaining control points.

Vehicle Manager Control Module

Responsibilities and Authority: This module's primary responsibility is to use the commanded planned **route elements** to build out the actual available lanes along these **route elements**, identifying wherever a physical

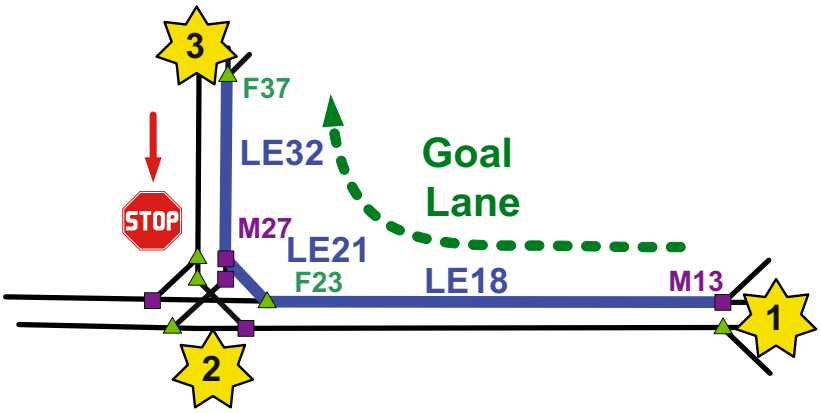

Fig. 8. A Lane Element (LE) is defined single lane section of road that connects two Lane Junctions. Lane Junctions are either Forks (e.g., F23 which is a right turn fork) or Merges (e.g., M27 which is a merge of a left turn and a right turn). This figure illustrates the lane elements for a single Route Element that turns right at the intersection (defined as Map Point 2). Here, Lane Element 18 (LE18) is one lane of a two lane road that goes from Merge 13 (M13) to Fork 23 (F23). LE21 goes from F23 to M27 and LE32 goes from M27 to F37. The Vehicle Manager Control Module carries out this further refinement of the commanded Route Element 14 from the Destination Manager to build the sequence of these three Lane Elements (LE18, LE21, and LE32) that define the planned Goal Lane for the vehicle

branching (fork) exists and selecting the correct branch (such as a right turn as opposed to passing through an intersection). This module assembles the resultant **lane element** sequences in groups that define a **goal lane** on a roadway between intersections or a **goal lane** that defines the maneuver through an intersection. Real-time navigational data input from road name signs, exit signs and lane-use restriction signs are used to assist plan execution.

*Retrieves and builds a sequence of **lane elements** to specify a **goal lane**.*

This module is commanded to go along a ***route segment***, which is described by the corresponding sequence list of **route elements**. This control module converts this higher level of route abstraction into the actual lanes that are available for own vehicle to travel on along these **route elements**. It does this by building out the sequence of **lane elements** that will represent the more detailed route plan for the vehicle and then groups these **lane elements** into commanded ***goal lanes*** as shown in Fig. 8. It further decides, based on navigational concerns at this level, what priority to set on the commanded ***goal lane***. If there is an upcoming right hand turn, for instance, on a multi-lane road, it may specify that the ***goal lane*** (which is the furthest lane to the right) for a **FollowRoad** command is at the priority of only-required-to-be-in-goal-lane-when-reach-end-of-goal-lane so that the subordinate control modules are free to chose adjacent lanes to respond to drive behavior issues but the vehicle must be in that lane at the end of the lane in time for the upcoming right turn.

This module assembles **lane elements** in order to specify either the ***goal lanes*** between intersections or the ***goal lanes*** that navigate through intersections (including turns). In this way, these ***goal lane*** specifications accompany the commands to the subordinate Mobility control module to carry out **Follow Road**, **Turn Right At Intersection**, **Pass Through Intersection**, etc. actions.

This module uses those real-time world model data that can affect the choice of **lane elements**, which affects which fork and/or merge **lane junction** to use. Its world data includes the road names of its own road and all crossing roads as well as navigational signs that indicate upcoming intersections, turn lanes and exit ramps, as well as any lane restrictions such as “No Left Turn – 4:30 to 7:00 p.m.”. It also monitors for

any situations that might cause changes in the set of **lane elements** selected to execute a commanded **route element**. As an example, a right turn ramp might be closed for construction requiring the vehicle to proceed into the intersection to make the turn, or the left turn lane may be blocked by an accident requiring the vehicle to pass through the intersection and make a U-turn then a left turn. But in all cases, the commanded **route element** is still being followed. However, if the right hand turn road were completely blocked by an accident, this would create a situation requiring a response beyond the level of authority of this Vehicle control module since the commanded **route element** can no longer be followed. This data would feedback through the world model to the Destination manager module to trigger a re-planning of a new sequence of **route elements** to deal with this situation. The Destination manager would then command the Vehicle module with a new ***route segment*** composed of a new sequence of **route elements**. The Vehicle control module would, in turn, build out its corresponding set of **lane elements** to specify the new ***goal lane*** and command these to the Mobility control module. All of these responses would happen in a small fraction of a second. This illustrates how each module can only make decisions at its level of responsibility and authority.

Input-Command: A command to **GoOn** ***SouthDr*** **TurnRightAt** ***ServiceDr*** which is the form of a ***Goal Route Segment*** (similar to a MapQuest-like command), which is basically a command to follow a specified road until you have to turn onto a different road.

Associated command data: Sequential list of the **route elements** that when combined, make up the specified route segment command. This is a list of route elements, each of which specifies the route through the next succeeding intersection (these will all be pass-through-the-intersection route elements) until the last **route element** of the list that will specify the turn required to go onto a different road completing the specification of the commanded ***route segment***.

Input-World Model: Present estimate of this module's relevant map of the **road network** – this is a map at the level of **lane junctions** and **lane elements**. **Lane junctions** are all of the points along a single travelway where either a physical fork from one lane to two lanes occurs (a Fork Junction) or two lanes come together into one lane (a Merge Junction). The interconnecting stretches of single lane travelways are the **lane elements**. This is analogous to a train track where all of the switches represent the **lane junctions** and the connecting track sections are the **lane elements**. This is a very appropriate analogy since a vehicle traveling on a road is very much constrained to its present lane as a train is to its track and can only change to a different lane at junction/switch points. Every physical fork junction is a decision point where the controller has to decide which of the two lanes to choose. The decision to change lanes along a route with multiple lanes in the same direction is not a navigation issue but a drive behavior issue that will be discussed below. This module's map representation is concerned with navigational decisions at the level of physical road forks and merges. These structures of **lane junctions** and **lane elements** carry many attributes to aid in route planning. Each **lane junction** is classified as a Fork or a Merge. If it is a Fork, it is classified as forming a right or left turn, or adding a lane. For a Merge, it is classified as to which merging lane, right or left, has the right-of-way. If there is a right-of-way control point at the **lane junction**, then whether the control is a stop sign, yield sign, signal light, or own direction of travel has right-of-way is specified.

Each **lane element** is classified as to its type, i.e., right or left turn, entering, exiting, or passing through an intersection, or a roadway. If there is a right-of-way control point within the **lane element**, then whether the control is a stop sign, yield sign, signal light, or own direction of travel has right-of-way. The data structures for the **lane elements** also maintain a large number of relationships, such as, the next following and previous **lane elements**, the adjacent **lane elements**, the type of road (two-lane undivided, four-lane undivided, four-lane divided, etc.) and the **lane element**'s position relative to other **lane elements** in the roadway, whether it starts or ends at an intersection, passes through an intersection or turns, etc. They are also classified by the action taken to enter them (fork right or left, come from merge) and the action at their end (merge-to-left, merge-to-right, etc.). Each **lane element** also references its start and end **lane junction**. All of the **lane elements** carry navigational information in the form of start and end UTM coordinates, overall length, speed limits, heading direction, average times to traverse, name of the road, and which **route element** (used by the Destination Manager control module) that they are a part of. In a corresponding cross-referencing, each **route element** maintains a list of **lane elements** that make it up.

World model road network data includes a set of history attributes that store average times to traverse each lane element under various conditions. These should improve the planning estimates.

Output-Command: A command to **Follow Road**, or **Turn Right At Intersection**, or **Cross Through Intersection** along with the data specification of the corresponding ***Goal Lane*** in the form of a sequential list of **lane elements** along with a specific time to arrive at end of the goal lane. Additionally, in the case of a multi-lane travelway with lanes traveling parallel with the goal lane, the priority of own vehicle being in the goal lane is specified by parameters such as desired-to-be-in-goal-lane, or required-to-be-in-goal-lane, or only-required-to-be-in-goal-lane-when-reach-end-of-goal-lane.

Associated command data: Sequential list of the **lane elements** that identify the goal lane for own vehicle to travel. Additionally, the time of arrival at the end of the specified ***GoalLane*** is also commanded.

Output-Status: Present state of goal accomplishment (i.e., the commanded route segment) in terms of executing, done, or error state, and identification of which intermediate **route elements** have been executed along with estimated time to complete each remaining **route element**.

Mobility Control Module

Responsibilities and Authority: This module's primary responsibility is to determine own vehicle's present right-of-way based on the road/intersection topography, control devices, rules of the road, and the state of other vehicles and pedestrians. It determines when own vehicle can go and determines the **lane segments** that define its nominal path (see Fig. 9). It also determines which objects might affect its path motion and places these in an **Objects-of-Interests** table to send to the Elemental Movement control module to plan a path around them.

Retrieves and builds a sequence of **lane segments** to specify a **goal lane-segment path**. Builds a table of active objects (with evaluation costs) that might influence own vehicle's path.

This module is commanded to go along a goal lane, which is described by a list of **lane elements**. It derives a set of **lane segments**, which are constant curvature arcs that specify the expected center-of-lane path for the **lane elements**. This module registers/aligns real-time updates of these **lane segments** as derived from sensed position of the road and lanes.

Additionally, this module applies road and intersection topography templates to fill in all of the relevant lane data, specifying such information as to which lanes cross, or merge, or do not intersect own lane

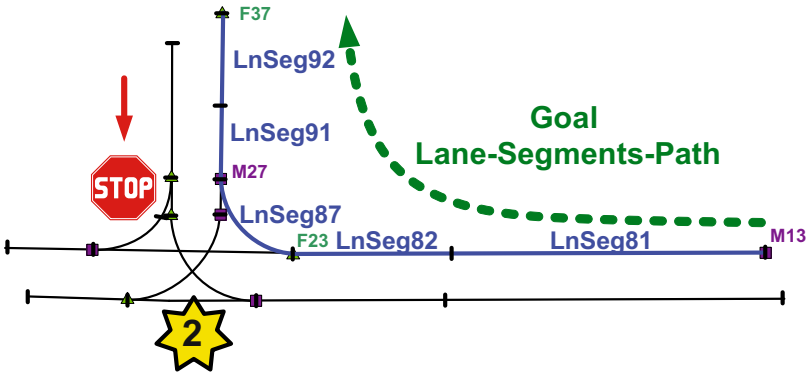

Fig. 9. Lane Segments (LnSeg) are defined as the nominal center-of-lane path representations in the form of *constant curvature arcs and straight lines*. They pass through the forks and merges that bound the Lane Elements. In this example, the Lane Element 18 references a linked list of lane segments containing LnSeg81 and LnSeg82, LE21 references LnSeg87, and LE32 references LnSeg91 and LnSeg92. Thus, the Lane Element is a data structure to abstract and reference the set of actual center-of-lane paths (Lane Segments). The Mobility control module manages this relationship by constantly updating the list of Lane Segments for a particular Lane Element based on real-time sensing of the lane position and curvature

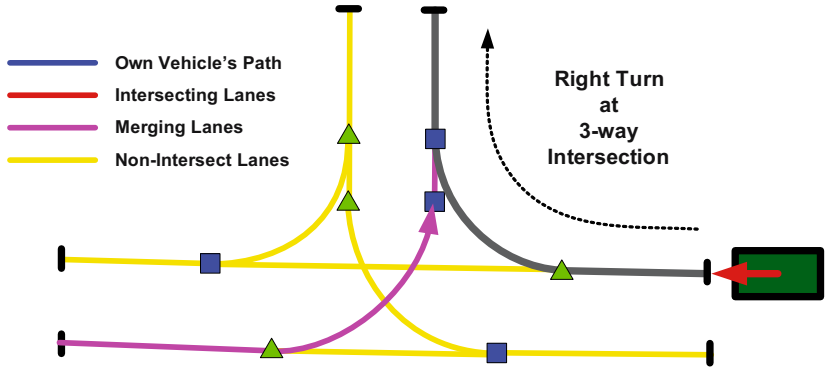

Fig. 10. A Lane Intersection Template for a 3-way intersection describes the type of intersecting interactions between own vehicle goal lane and all of the other lanes in the intersection

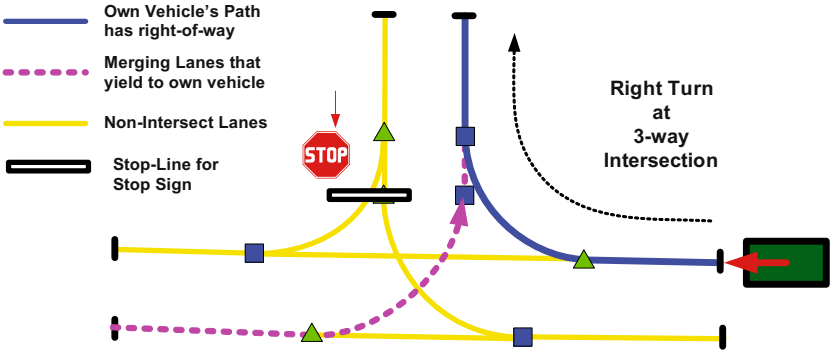

Fig. 11. The specific control devices (here – a Stop Sign) are populated into the template to arrive at the right-of-way behaviors of vehicles in all of the lanes of the intersection

(see Fig. 10) These structures are then further overlaid with present detected control devices such as stop signs to specify the relative right-of-way considerations (see Fig. 11) for all of the lanes nearby own vehicle's goal lane. This world model structure serves as the basis for real-time assessment of own vehicle's right of way when overlaid with other vehicles and their behavior states.

This module uses these observed vehicles' and pedestrians' world states including position, velocity and acceleration vectors, classification of expected behavior type (aggressive, normal, conservative), and intent (stopping at intersection, turning right, asserting right-of-way, etc.) to decide what action own vehicle should do and when to do it.

This module also provides classification of objects relevant to the present commanded driving task (see Fig. 12) by determining which objects are close enough to the identified relevant lanes to have the potential to affect own vehicle's planned path. These objects (see Fig. 13) are placed into an ***Objects-of-Interest*** table along with a number of computed parameters such as required offset distance, passing speed, cost to violate offset or passing speed, cost to collide, as well as dynamic state parameters such as velocity and acceleration vectors and other parameters. This table will serve as part of the command data set to the Elemental Movement control module.

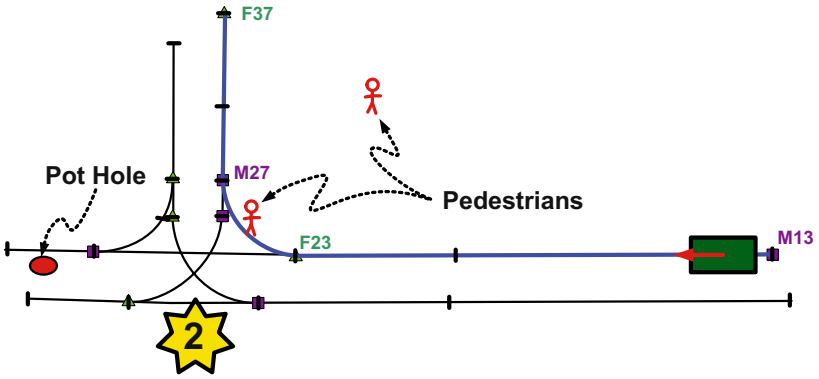

Fig. 12. Objects relevant to on-road driving (here pedestrians and pot holes) are sensed, classified, and placed into the world model

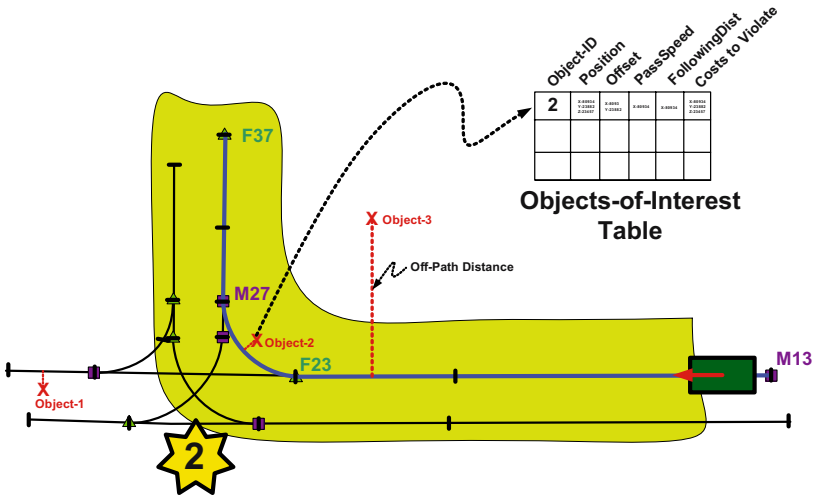

Fig. 13. The Mobility control module tests the objects in the world model to see which ones are within a specified distance to own vehicle's goal lane (here shown as a *shaded green area*). In this figure, only Object-2 (a pedestrian) is in this region. Mobility manager places this object into an Objects-of-Interest table along with parameters of minimum offset distance, passing speed, following distance and cost values to exceed these. This is part of the command to the subordinate Elemental Movement control module

Input-Command: A command to **FollowRoad**, or **TurnRightAtIntersection**, or **Cross ThroughIntersection**, etc. along with the data specification of the corresponding ***Goal Lane*** in the form of a sequential list of **lane elements**, with a specific time to be at the end of the goal lane. In the case of adjacent lanes in the same direction as the goal lane, the priority of own vehicle being in the goal lane is specified with parameters such as desired, or required, or required-when-reach-end-of-lane.

Input-World Model: Present estimate of this module's relevant map of the **road network** – this is a map at the level of **lane segments** which are the nominal center-of-lane specifications in the form a constant curvature arcs. This module builds out the nominal **lane segments** for each **lane element** and cross-references them with the corresponding **lane elements**. The world model contains estimates of the actual **lane segments** as provided by real-time sensing of roads and lanes and other indicators. This module will register these real-time **lane segments** with its initial nominal set.

This module's world model contains all of the surrounding recognized objects and classifies them according to their relevance to the present commanded driving task. All objects determined to have the potential to affect own vehicle's planned path are placed into an ***Objects-of-Interest*** table along with a number of parameters such as offset distance, passing speed, cost to violate offset or passing speed, cost to collide as well as dynamic state parameters such as velocity and acceleration and other parameters.

Other objects include detected control devices such as stop signs, yield signs, signal lights and the present state of signal lights. Regulatory signs such as speed limit, slow for school, sharp turn ahead, etc. are included.

The observed other vehicles' and pedestrians' and other animate objects' world states are contained here and include position, velocity and acceleration vectors, classification of expected behavior type (aggressive, normal, conservative), and intent (stopping at intersection, turning right, asserting right-of-way, following-motion-vector, moving-randomly, etc.).

Additionally, this module's world model contains road and intersection topography, intersection, and right-of-way templates for a number of roadway and intersection situations. These are used to aid in the determination of which lanes cross, or merge, or do not intersect own lane and to aid in the determination of right-of-way.

Output-Command: A command to **FollowLane**, **FollowRightTurnLane**, **FollowLeftTurnLane**, **StopAtIntersection**, **ChangeToLeftLane**, etc. along with the data specification of a ***Goal Lane-Segment Path*** in the form of a sequential list of **lane segments** that define the nominal center-of-lane path the vehicle is to follow. Additionally, the command includes an ***Objects-of-Interest*** table that specifies a list of objects, their position and dynamic path vectors, the offset clearance distances, passing speeds, and following distances relative to own vehicle, the cost to violate these values, these object dimensions, and whether or not they can be straddled.

Output-Status: Present state of goal accomplishment (i.e., the commanded ***GoalLane***) in terms of executing, done, or error state, and identification of which **lane elements** have been executed along with estimated time to complete each of the remaining **lane elements**.

Elemental Movement Control Module

Responsibilities and Authority: This module's primary responsibility is to define the **GoalPaths** that will follow the commanded lane, slowing for turns and stops, while maneuvering in-lane around the objects in the **Objects-of-Interests** table.

*Constructs a sequence of **GoalPaths**.*

This module is commanded to follow a sequence of **lane segments** that define a ***goal lane-segment path*** for the vehicle. It first generates a corresponding set of goal paths for these lane segments by determining decelerations for turns and stops as well as maximum speeds for arcs both along the curving parts of the roadway and through the intersection turns. This calculation results in a specified enter and exit speed for each goal path. This will cause the vehicle to slow down properly before stops and turns and to have the proper speeds around turns so as not to have too large a lateral acceleration. It also deals with the vehicle's ability to decelerate much faster than it can accelerate.

This module also receives a table of ***Objects-of-Interests*** that provides cost values to allow this module to calculate how to offset these calculate ***GoalPaths*** and how to vary the vehicle's speed to meet these cost requirements while being constrained to stay within some tolerance of the commanded ***GoalPath***. This tolerance is set to keep the vehicle within its lane while avoiding the objects in the table. If it cannot meet the cost requirements associated with the ***Objects-of-Interest*** by maneuvering in its lane, it slows the vehicle to a stop before reaching the object(s) unless it is given a command from the Mobility control

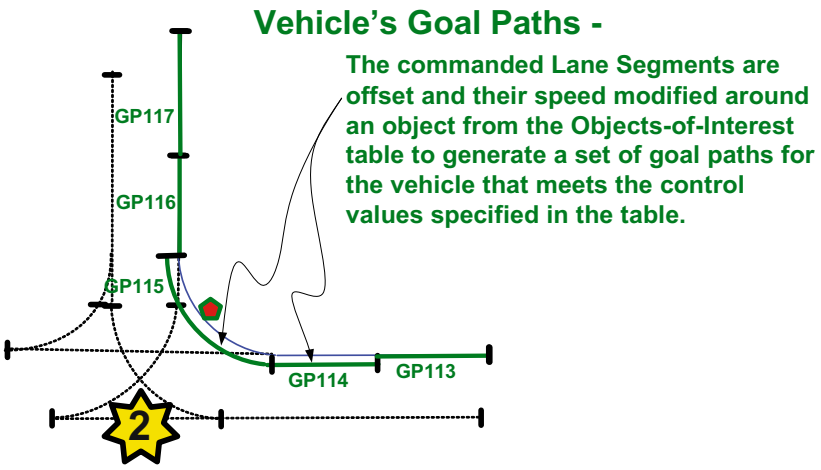

Fig. 14. Elemental Movement control module generates a set of goal paths with proper speeds and accelerations to meet turning, slowing, and stopping requirements to follow the goal lane as specified by the commanded lane segments (center-of-lane paths). However, it will modify these lane segments by offsetting certain ones and altering their speeds to deal with the object avoidance constraints and parameters specified in the Objects-of-Interest table from the Mobility control module. Here, Goal Path 114 (GP114) and GP115 are offset from the original lane segment specifications (LnSeg82 and LnSeg87) to move the vehicle's goal path far enough out to clear the object (shown in *red*) from the Objects-of-Interest table at the specified offset distance. The speed along these goal paths is also modified according to the values specified in the table

module allowing it to go outside of its lane. The Elemental Movement module is continually reporting status to the Mobility control module concerning how well it is meeting its goals. If it cannot maneuver around an object while staying in-lane, the Mobility module is notified and immediately begins to evaluate when a change lane command can be issued to Elemental Movement module.

This module will construct one or more ***GoalPaths*** (see Fig. 14) with some offset (which can be zero) for each commanded lane segment based on its calculations of the values in the ***Objects-of-Interest*** table. It commands one goal path at a time to the Primitive control module but also passes it the complete set of planned ***GoalPaths*** so the Primitive control module has sufficient look-ahead information to calculate dynamic trajectory values. When the Primitive control module indicates it is nearing completion of its commanded ***GoalPath***, the Elemental Movement module re-plans its set of ***GoalPaths*** and sends the next ***GoalPath***. If, at anytime during execution of a ***GoalPath***, this module receives an update of either the present commanded lane segments or the present state of any of the ***Objects-of-Interest***, it performs a re-plan of the ***GoalPaths*** and issues a new commanded ***GoalPath*** to the Primitive control module.

Input-Command: A command to **FollowLane**, **FollowRightTurnLane**, **FollowLeftTurnLane**, **StopAtIntersection**, **ChangeToLeftLane**, etc. along with the data specification of a ***Goal Lane-Segment Path*** in the form of a sequential list of **lane segments** that define the nominal center-of-lane path the vehicle is to follow. Additionally, the command includes an ***Objects-of-Interest*** table that specifies a list of objects, their position and dynamic path vectors, the offset clearance distances, passing speeds, and following distances relative to own vehicle, the cost to violate these values, these object dimensions, and whether or not they can be straddled.

Input-World Model: Present estimate of this module's relevant map of the **road network** – this is a map at the level of present estimated **lane segments**. This includes the **lane segments** that are in the commanded ***goal lane segment path*** as well as the real-time estimates of nearby **lane segments** such as

the adjacent on-coming **lane segments**. This world model also contains the continuously updated states of all of the objects carried in the ***Objects-of-Interest*** table. Each object's state includes the position, velocity, and acceleration vectors, and history and classification of previous movement and reference model for the type of movement to be expected such.

Output-Command: A command to **Follow_StraightLine**, **Follow_CirArcCW**, **Follow_CirArcCCW**, etc. along with the data specification of a single goal path within a sequential list of ***GoalPaths*** that define the nominal path the vehicle is to follow.

Output-Status: Present state of goal accomplishment (i.e., commanded ***goal lane-segment path***) in terms of executing, done, or error state, and identification of which **lane segments** have been executed along with estimated time to complete each of the remaining **lane segments**.

Primitive (Dynamic Trajectory) Control Module

Responsibilities and Authority: This module's primary responsibility is to pre-compute the set of dynamic trajectory path vectors for the sequence of goal paths, and to control the vehicle along this trajectory. Constructs a sequence of **dynamic path vectors** which yields the speed parameters and heading vector.

This module is commanded to follow a ***GoalPath*** for the vehicle. It has available a number of relevant parameters such as derived maximum allowed tangential and lateral speeds, accelerations, and jerks. These values have rolled up the various parameters of the vehicle, such as engine power, braking, center-of-gravity, wheel base and track, and road conditions such as surface friction, incline, and side slope. This module uses these parameters to pre-compute the set of dynamic trajectory path vectors (see Fig. 15) at a much faster than real-time rate $(100 - 1)$, so it always has considerable look-ahead. Each time a new command comes in from Elemental Movement (because its lane segment data was updated or some object changed state), the Primitive control module immediately begins a new pre-calculation of the dynamic trajectory vectors from its present projected position and immediately has the necessary data to calculate the Speed and Steer outputs from the next vehicle's navigational input relative to these new vectors.

On each update of the vehicle position, velocity, and acceleration from the navigation system (every 10 ms), this module projects these values to estimate the vehicle's position at about 0.4 s into the future, finds the closest stored pre-calculated dynamic trajectory path vector to this estimated position, calculates the off-path difference of this estimated position from the vector and derives the next command speed, acceleration, and heading from these relationships.

Input-Command: A command to **Follow_StraightLine**, **Follow_CirArcCW**, **Follow_CirArcCCW**, etc. with the data of a single goal path in the form of a constant curvature arc specification along with the allowed tangential and lateral maximum speeds, accelerations, and jerks. The complete set of constant curvature paths that define all of the planned output ***goal paths*** from the Elemental Movement control module are also provided.

Input-World Model: Present estimate of this module's relevant map of the **road network** – this is a map at the level of goal paths commanded by the Elemental Movement control module. Other world model information includes the present state of the vehicle in terms of position, velocity, and acceleration vectors. This module's world model also includes a number of parameters about the vehicle such as maximum acceleration, deceleration, weight, allowed maximum lateral acceleration, center-of-mass, present heading, dimensions, wheel base, front and rear overhang, etc.

Output-Command: Commanded maximum speed, present speed, present acceleration, final speed at path end, distance to path end, and end motion state (moving or stopped) are sent to the Speed Servo control module. Commanded vehicle center absolute heading, present arc radius, path type (straight line, arc CW, or arc CCW), the average off-path distance, and the path region type (standard-roadway, beginning-of-intersection-turn, mid-way-intersection-turn, arc-to-straight-line-blend) are sent to the Steer Servo control module.

Output-Status: Present state of goal accomplishment (i.e., the commanded ***goal path***) in terms of executing, done, or error state, and estimated time to complete present ***goal path***. This module estimates time to the endpoint of the present goal path and outputs an advance reach goal point state to give an early warning to the Elemental Movement module so it can prepare to send out the next goal path command.

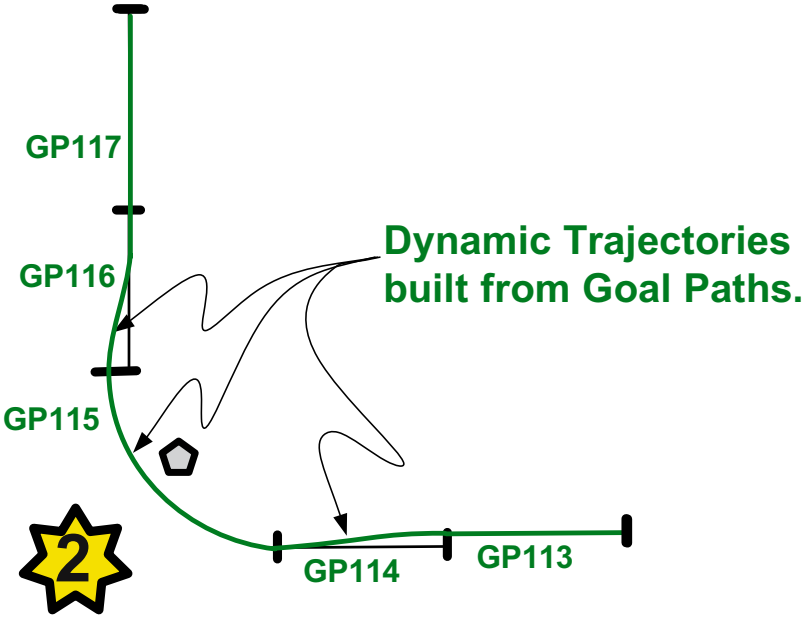

Fig. 15. Primitive/Trajectory control module pre-calculates (at 100× real-time) the set of dynamic trajectory vectors that pass through the specified goal paths while observing the constraints of vehicle-based tangential and lateral maximum speeds, accelerations, and jerks. As seen here, this results in very smooth controlled trajectories that blend across the offset goal paths commanded by the Elemental Movement control module

Speed Servo Control Module

Responsibilities and Authority: This module's primary responsibility is to use the throttle and brake to cause the vehicle to move at the desired speed and acceleration and to stop at the commanded position.

Uses a feedforward model-based servo to estimate throttle and brake-line pressure values.

This module is commanded to cause the vehicle to move at a speed with a specific acceleration constrained by a maximum speed and a final speed at the path end, which is known by a distance value to the endpoint that is continuously updated by the Primitive module.

This module basically uses a feedforward servo module to estimate the desired throttle and brake-line pressure values to cause the vehicle to attain the commanded speed and acceleration. An integrated error term is added to correct for inaccuracies in this feedforward model. The parameters for the feedforward servo are the commanded speed and acceleration, the present speed and acceleration, the road and vehicle pitch, and the engine rpm. Some of these parameters are also processed to derive rate of change values to aid in the calculations.

Input-Command: A command to **GoForwardAt*Speed*** or **GoBackwardAt*Speed*** or **StopAt*Point*** along with the parameters of maximum speed, present speed, present acceleration, final speed at path end, distance to path end, and end motion state (moving or stopped) are sent to the Speed Servo control module.

Input-World Model: Present estimate of relevant vehicle parameters – this includes real-time measurements of the vehicle's present speed and acceleration, present vehicle pitch, engine rpm, present normalized throttle position, and present brake line pressure. Additionally, estimates are made for the projected vehicle

speed, the present road pitch and the road-in-front pitch. The vehicle's present and projected positions are also utilized.

Output-Command: The next calculated value for the normalized throttle position is commanded to the throttle servo module and the desired brake-line pressure value is commanded to the brake servo module.

Output-Status: Present state of goal accomplishment (i.e., the commanded speed, acceleration, and stopping position) in terms of executing, done, or error state, and an estimate of error if this commanded goal cannot be reached.

Steer Servo Control Module

Responsibilities and Authority: This module's primary responsibility is to control steering to keep the vehicle on the desired trajectory path.

Uses a feedforward model-based servo to estimate steering wheel values.

This module is commanded to cause the heading value of the vehicle-center forward pointing vector (which is always parallel to the vehicle's long axis) to be at a specified value at some projected time into the future (about 0.4 s for this vehicle). This module uses the present steer angle, vehicle speed and acceleration to estimate the projected vehicle-center heading at 0.4 s into the future. It compares this value with the commanded vehicle-center heading and uses the error to derive a desired front wheel steer angle command. It evaluates this new front wheel steer angle to see if it will exceed the steering wheel lock limit or if it will cause the vehicle's lateral acceleration to exceed the side-slip limit. If it has to adjust the vehicle center-heading because of these constraints, it reports this scaling back to the Primitive module and includes the value of the vehicle-center heading it has scaled back to.

This module uses the commanded path region type to set the allowed steering wheel velocity and acceleration which acts as a safe-guard filter on steering corrections. This module uses the average off-path distance to continuously correct its alignment of its internal model of the front wheel position to actual position. It does this by noting the need to command a steer wheel value different than its model for straight ahead when following a straight section of road for a period of time. It uses the average off-path value from the Primitive module to calculate a correction to the internal model and updates this every time it follows a sufficiently long section of straight road.

Input-Command: A **GoForwardAt_*HeadingAngle*** or **GoBackwardAt_*HeadingAngle*** is commanded along with the parameters of vehicle-center absolute heading, present arc radius, path type (straight line, arc CW, or arc CCW), the average off-path distance, and the path region type (standard-roadway, beginning-of-intersection-turn, mid-way-intersection-turn, arc-to-straight-line-blend).

Input-World Model: Present estimate of relevant vehicle parameters – this includes real-time measurements of vehicle's lateral acceleration as well as the vehicle present heading, speed, acceleration, and steering wheel angle. Vehicle parameters of wheel lock positions, and estimated vehicle maximum lateral acceleration for side-slip calculations, vehicle wheel base and wheel track, and vehicle steering box ratios.

Output-Command: The next commanded value of steering wheel position along with constraints on maximum steering wheel velocity and acceleration are commanded to the steering wheel motor servo module.

Output-Status: Present state of goal accomplishment (i.e., the commanded vehicle-center heading angle) in terms of executing, done, or error state, along with status on whether this commanded value had to be scaled back and what the actual heading value used is.

This concludes the description of the 4D/RCS control modules for the on-road driving example.

2.3 Learning Applied to Ground Robots (DARPA LAGR)

Recently, ISD has been applying 4D/RCS to the DARPA LAGR program [7]. The DARPA LAGR program aims to develop algorithms that enable a robotic vehicle to travel through complex terrain without having to rely on hand-tuned algorithms that only apply in limited environments. The goal is to enable the control system of the vehicle to learn which areas are traversable and how to avoid areas that are impassable or that limit the mobility of the vehicle. To accomplish this goal, the program provided small robotic vehicles to each

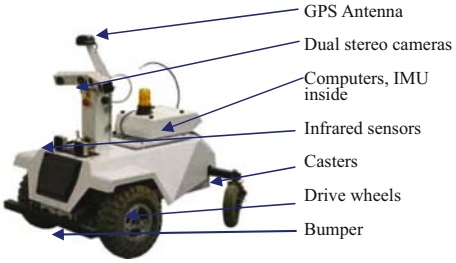

Fig. 16. The DARPA LAGR vehicle

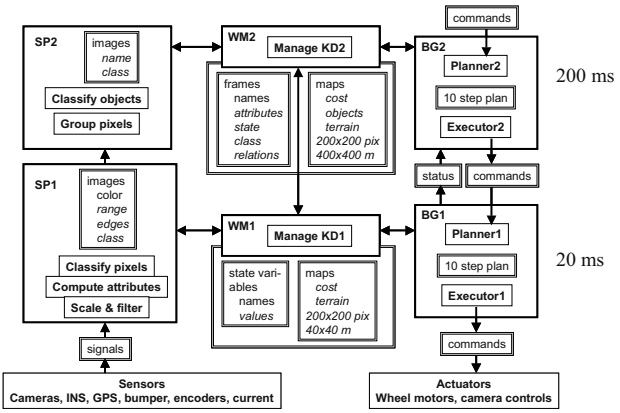

Fig. 17. Two-level instantiation of the 4D/RCS hierarchy for LAGR

of the participants (Fig. 16). The vehicles are used by the teams to develop software and a separate DARPA team, with an identical vehicle, conducts tests of the software each month. Operators load the software onto an identical vehicle and command the vehicle to travel from a start waypoint to a goal waypoint through an obstacle-rich environment. They measure the performance of the system on multiple runs, under the expectation that improvements will be made through learning.

The vehicles are equipped with four computer processors (right and left cameras, control, and the planner), wireless data and emergency stop radios, GPS receiver, inertial navigation unit, dual stereo cameras, infrared sensors, switch-sensed bumper, front wheel encoders, and other sensors listed later in the Chapter.

4D/RCS Applied to LAGR

The 4D/RCS architecture for LAGR (Fig. 17) consists of only two levels. This is because the size of the LAGR test areas is small (typically about 100 m on a side, and the test missions are short in duration (typically less than 4 min)). For controlling an entire battalion of autonomous vehicles, there may be as many as five or more 4D/RCS hierarchical levels.

The following sub-sections describe the type of algorithms implemented in sensor processing, world modeling, and behavior generation, as well as a section that describes the application of this controller to road following [8].

Sensory Processing

The sensor processing column in the 4D/RCS hierarchy for LAGR starts with the sensors on board the LAGR vehicle. Sensors used in the sensory processing module include the two pairs of stereo color cameras, the physical bumper and infra-red bumper sensors, the motor current sensor (for terrain resistance), and the navigation sensors (GPS, wheel encoder, and INS). Sensory processing modules include a stereo obstacle detection module, a bumper obstacle detection module, an infrared obstacle detection module, an image classification module, and a terrain slipperiness detection module.

Stereo vision is primarily used for detecting obstacles [9]. We use the SRI Stereo Vision Engine [10] to process the pairs of images from the two stereo camera pairs. For each newly acquired stereo image pair, the obstacle detection algorithm processes each vertical scan line in the reference image and classifies each pixel as GROUND, OBSTACLE, SHORT_OBSTACLE, COVER or INVALID.

A model-based learning process occurs in the SP2 module of the 4D/RCS architecture, taking input from SP1 in the form of labeled pixels with associated (x, y, z) positions from the obstacle detection module. This process learns color and texture models of traversable and non-traversable regions, which are used in SP1 for terrain classification [11]. Thus, there is two-way communication between the levels, with labeled 3D data passing up, and models passing down. The approach to model building is to make use of the labeled SP1 data including range, color, and position to describe regions in the environment around the vehicle and to associate a cost of traversing each region with its description. Models of the terrain are learned using an unsupervised scheme that makes use of both geometric and appearance information [12].

The system constructs a map of a 40 by 40 m region of terrain surrounding the vehicle, with map cells of size 0.2 m by 0.2 m and the vehicle in the center of the map. The map is always oriented with one axis pointing north and the other east. The map scrolls under the vehicle as the vehicle moves, and cells that scroll off the end of the map are forgotten. Cells that move onto the map are cleared and made ready for new information. The model-building algorithm takes its input from SP1 as well as the location and pose of the vehicle when the data were collected.

The models are built as a kind of learning by example. The obstacle detection module identifies regions by height as either obstacles or ground. Models associate color and texture information with these labels, and use these examples to classify newly-seen regions. Another kind of learning is also used to measure traversability. This is especially useful in cases where the obstacle detection reports a region to be of one class when it is actually of another, such as when the system sees tall grass that looks like an obstacle but is traversable, perhaps with a greater cost than clear ground. This second kind of learning is learning by experience: observing what actually happens when the vehicle traverses different kinds of terrain. The vehicle itself occupies a region of space that maps into some neighborhood of cells in the traversability cost map. These cells and their associated models are given an increased traversability weight because the vehicle is traversing them. If the bumper on the vehicle is triggered, the cell that corresponds to the bumper location and its model, if any, are given a decreased traversability weight. We plan to further modify the traversability weights by observing when the wheels on the vehicle slip or the motor has to work harder to traverse a cell.

The models are used in the lower sensory processing module, SP1, to classify image regions and assign traversability costs to them. For this process only color information is available, with the traversability being inferred from that stored in the models. The approach is to pass a window over the image and to compute the same color and texture measures at each window location as are used in model construction. Matching between the windows and the models operates exactly as it does when a cell is matched to a model in the learning stage. Windows do not have to be large, however. They can be as small as a single pixel and the matching will still determine the closest model, although with low confidence (as in the color model method for road detection described below). In the implementation the window size is a parameter, typically set to 16×16. If the best match has an acceptable score, the window is labeled with the matching model. If not, the window is not classified. Windows that match with models inherit the traversability measure associated with the model. In this way large portions of the image are classified.

World Modeling

The world model is the system's internal representation of the external world. It acts as a bridge between sensory processing and behavior generation in the 4D/RCS hierarchy by providing a central repository for storing sensory data in a unified representation. It decouples the real-time sensory updates from the rest of the system. The world model process has two primary functions: To create a knowledge database and keep it current and consistent, and to generate predictions of expected sensory input.

For the LAGR project, two world model levels have been built (WM1 and WM2). Each world model process builds a two dimensional map (200 × 200 cells), but at different resolutions. These are used to temporally fuse information from sensory processing. Currently the lower level (Sensory Processing level one, or SP1) is fused into both WM1 and WM2 as the learning module in SP2 does not yet send its models to WM. Figure 18 shows the WM1 and WM2 maps constructed from the stereo obstacle detection module in SP1. The maps contain traversal costs for each cell in the map. The position of the vehicle is shown as an overlay on the map. The red, yellow, blue, light blue, and green are cost values ranging from high to low cost, and black represents unknown areas. Each map cell represents an area on the ground of a fixed size and is marked with the time it was last updated. The total length and width of the map is 40 m for WM1 and 120 m for WM2. The information stored in each cell includes the average ground and obstacle elevation height, the variance, minimum and maximum height, and a confidence measure reflecting the "goodness" of the elevation data. In addition, a data structure describing the terrain traversability cost and the cost confidence as updated by the stereo obstacle detection module, image classification module, bumper module, infrared sensor module, etc. The map updating algorithm relies on confidence-based mapping as described in [15].

We plan additional research to implement modeling of moving objects (cars, targets, etc.) and to broaden the system's terrain and object classification capabilities. The ability to recognize and label water, rocky roads, buildings, fences, etc. would enhance the vehicle's performance [16–20].

Behavior Generation

Top level input to Behavior Generation (BG) is a file containing the final goal point in UTM (Universal Transverse Mercator) coordinates. At the bottom level in the 4D/RCS hierarchy, BG produces a speed for

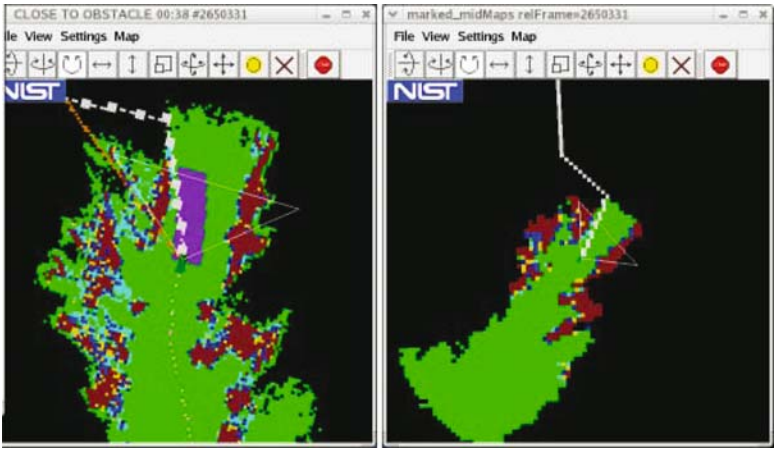

Fig. 18. OCU display of the World Model cost maps built from sensor processing data. WM1 builds a 0.2 m resolution cost map (*left*) and WM2 builds a 0.6 m resolution cost map (*right*)

each of the two drive wheels updated every 20 ms, which is input to the low-level controller included with the government-provided vehicle. The low-level system returns status to BG, including motor currents, position estimate, physical bumper switch state, raw GPS and encoder feedback, etc. These are used directly by BG rather than passing them through sensor processing and world modeling since they are time-critical and relatively simple to process.

Two position estimates are used in the system. Global position is strongly affected by the GPS antenna output and received signal strength and is more accurate over long ranges, but can be noisy. Local position uses only the wheel encoders and inertial measurement unit (IMU). It is less noisy than GPS but drifts significantly as the vehicle moves, and even more if the wheels slip.

The system consists of five separate executables. Each sleeps until the beginning of its cycle, reads its inputs, does some planning, writes its outputs and starts the cycle again. Processes communicate using the Neutral Message Language (NML) in a non-blocking mode, which wraps the shared-memory interface [21]. Each module also posts a status message that can be used by both the supervising process and by developers via a diagnostics tool to monitor the process.

The LAGR Supervisor is the highest level BG module. It is responsible for starting and stopping the system. It reads the final goal and sends it to the waypoint generator. The waypoint generator chooses a series of waypoints for the lowest-cost traversable path to the goal using global position and translates the points into local coordinates. It generates a list of waypoints using either the output of the A* Planner [22] or a previously recorded known route to the goal.

The planner takes a 201×201 terrain grid from WM, classifies the grid, and translates it into a grid of costs of the same size. In most cases the cost is simply looked up in a small table from the corresponding element of the input grid. However, since costs also depend on neighboring costs, they are automatically adjusted to allow the vehicle to continue motion. By lowering costs of unknown obstacles near the vehicle, it does not hesitate to move as it would with for example, detected false or true obstacles nearby. Since the vehicle has an instrumented bumper, the choice is to continue vehicle motion.

The lowest level module, the LAGR Comms Interface, takes a desired heading and direction from the waypoint follower and controls the velocity and acceleration, determines a vehicle-specific set of wheel speeds, and handles all communications between the controller and vehicle hardware.

Road and Path Detection in LAGR

In the LAGR environment, roads, tracks, and paths are often preferred over other terrain. A color-based image classification module learns to detect and classify these regions in the scene by their color and appearance, making the assumption that the region directly in front of the vehicle is traversable. A flat world assumption is used to estimate the 3D location of a ground pixel in the image. Our algorithm segments an image of a region by building multiple color models similar to those proposed by Tan et al. [23], who applied the approach to paved road following. For off-road driving, the algorithm was modified to segment an image into traversable and non-traversable regions. Color models are created for each region based on two-dimensional histograms of the colors in selected regions of the image. Previous approaches to color modeling have often made use of Gaussian mixture models, which assumes Gaussian color distributions. Our experiments showed that this assumption did not hold in our domain. Instead, we used color histograms. Many road detection systems have made use of the RGB color space in their methods. However, previous research [24–26] has shown that other color spaces may offer advantages in terms of robustness against changes in illumination. We found that a 30×30 histogram of red (R) and green (G) gave the best results in the LAGR environment.

The approach makes the assumption that the area in front of the vehicle is safe to traverse. A trapezoidal region at the bottom of the image is assumed to be ground. A color histogram is constructed for the points in this region to create the initial ground model. The trapezoidal region is the projection of a 1 m wide by 2 m long area in front of the vehicle under the assumption that the vehicle is on a plane defined by its current pose. In [27] Ulrich and Nourbakhsh addressed the issue of appearance-based obstacle detection using a color camera without range information. Their approach makes the same assumptions that the ground is flat and that the region directly in front of the robot is ground. This region is characterized by Hue and Saturation histograms and used as a model for ground. Ulrich and Nourbakhsh do not model the background, and have

only a single ground model (although they observe that more complex environments would call for multiple ground models). Their work was applied to more homogeneous environments than ours, and we found that multiple models of ground are essential for good performance in the kinds of terrain used to test the LAGR vehicles. Substantial work has since been done in the DARPA LAGR program on learning traversability models from stereo, color, and texture (e.g., [16–20]). Other work, such as [28], that makes use of LADAR instead of stereo has also been of growing interest, but is not covered here.

In addition to the ground models, a background model is also built. Construction starts by randomly sampling pixels in the area above the horizon, assuming that they represent non-traversable regions. Because this area might only contain sky pixels, we extend the sampling area to 50 pixels below the horizon. The horizon is the line determined by the points where the line of sight of the cameras stops intersecting the ground plane. Once the algorithm is running, the algorithm randomly samples pixels in the current frame that the previous result identified as background. This enables the background regions to expand below the horizon. These samples are used to update the background color model using temporal fusion. Only one background model is constructed since there is no need to distinguish one type of background from another.

To enable the vehicle to remember traversable terrain with different color characteristics, multiple ground color models are learned. As new data are processed, each color distribution model is updated with new histograms, changing with time to fit changing conditions. Potential new histograms for representing ground are compared to existing models. If the difference is less than a threshold, the histogram is used to upgrade the best matching ground model. Otherwise, if a maximum number of ground models has not yet been reached, the algorithm enters a period known as learning mode. During learning mode, the algorithm monitors new histograms in an attempt to pick out the histogram that is most different from the existing ground models. This is done to avoid picking color models that contain significant amounts of overlap. In the learning mode, if a histogram is found to be more different than a previous histogram, learning mode is extended. Eventually learning mode will end, and the most different histogram is used to create a new color model.

Learning mode is turned off when the region in front of the vehicle assumed to be ground contains obstacles. This is determined by projecting obstacles from the world model map into the image. It is also disabled if the LAGR vehicle is turning faster than 10 degrees per second or if the LAGR vehicle is not moving. The algorithm can also read in a priori models of certain features that commonly appear in the LAGR tests. These include models for a path covered in mulch or in white lime. Figure 19 shows two examples of the output of the color model based classifier. Figure 19a shows a view of an unpaved road, while Fig. 19b shows a path laid down in a field that the vehicle is supposed to follow.

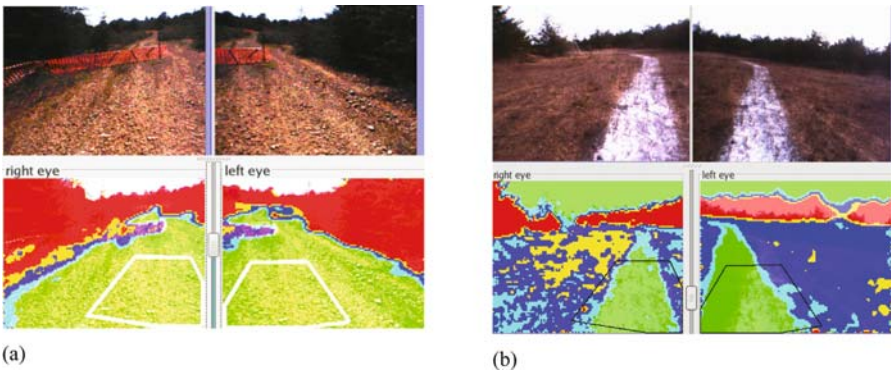

Fig. 19. (**a**) *Top*: the original images, with the classification images based on the histogram color model shown underneath. *Green* means ground, the other colors are background regions with higher traversability costs. (**b**) Another scene showing clearly the algorithm's ability to learn to associate traversability with a distinctively-colored path

Given the models, the algorithm goes through every pixel in an image and calculates a ground probability based upon its color. The end result is a probability map that represents the likelihood that an area is ground. Given the pixel's color, a ground color model and the model for background, ground probability is calculated as:

$$P_{ground} = \frac{N_{ground}}{N_{ground} + N_{background}}$$

where N_{ground} is the count in the ground histogram bin indexed by the pixel, $N_{background}$ is the count in the background histogram bin indexed by the pixel, and P_{ground} is the probability that the pixel is a ground pixel. When there are multiple ground models, all are matched and the largest ground probability is selected. Thus multiple ground probabilities are calculated at each pixel. The largest ground probability is selected as the ground probability for that pixel.

The next step applied to the ground histograms is temporal fusion. This combines ground probabilities across multiple image frames to improve stability and reduce noise. The temporal fusion algorithm can be described as a running average with a parameter to adjust for the influence of new data. The update equation is:

$$P_t = \frac{(w_{t-1} \times P_{t-1}) + (c \times P)}{(w_{t-1} + c)}$$

$$w_t = w_{t-1} + c \text{ if } w_{t-1} < w_{\max}$$

where P is the current probability that the pixel should be labeled ground, P_{t-1} is the probability that the same pixel was labeled ground in the previous frame, P_t is the temporally-fused ground probability of the pixel, and w and c are weighting constants. $w_{\max}$ is the maximum number of images used for temporal fusion. The final probability map is used to determine the traversability cost at each pixel as

$$Cost = (1 - P_t) * 250$$

Costs run from 0 being most traversable to 250 being least traversable.

In order to reduce processing requirements, probabilities are calculated on a reduced version of the original image. The original image is resized to 128×96 pixels using an averaging filter. This step has the additional benefit of noise reduction. Experiments show that this step does not significantly impact the final segmentation. A noteworthy aspect of this algorithm is that the color models are constructed from the original image for better accuracy, whereas probabilities are calculated on a reduced version of the image for greater speed. The cost of each pixel in the image is sent to the world model with a 3D location determined using the assumption of a flat ground plane.

Summary

The NIST 4D/RCS reference model architecture was implemented on the DARPA LAGR vehicle, which was used to prove that 4D/RCS can learn. Sensor processing, world modeling, and behavior generation processes have been described. Outputs from sensor processing of vehicle sensors are fused with models in WM to update them with external vehicle information. World modeling acts as a bridge between multiple sensory inputs and a behavior generation (path planning) subsystem. Behavior generation plans vehicle paths through the world based on cost maps provided from world modeling. Road following is the example used here to describe how 4D/RCS can be applied in a two-level architecture.

Future research will include completion of the sensory processing upper level (SP2) and developing even more robust control algorithms than those described above.

In the second 18-month phase of the LAGR program, NIST was tasked to develop a standard operator control unit color scheme [29] for all performers to use.

The Operator Control Unit (OCU) for a mobile robot needs to display a lot of complex information about the state and planned actions of the vehicle. This includes displays from the robot's sensors, maps of what it knows about the world around it, traces of the path it has already traveled and predictions of the path

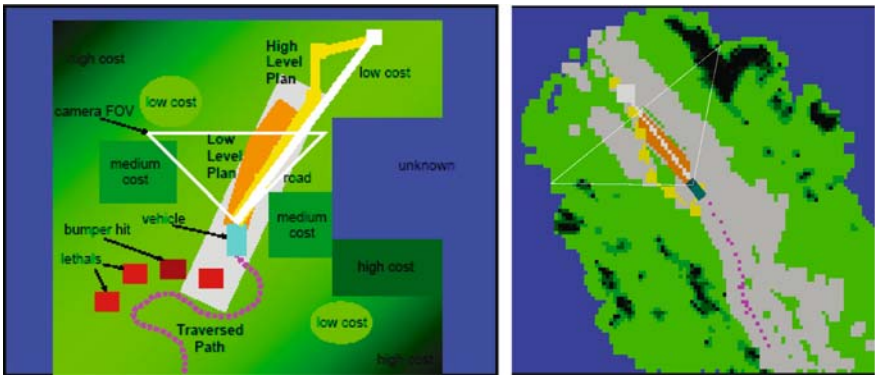

Fig. 20. (*left*) A schematic showing the meaning of the colors on the map, (*right*) A sample NIST high resolution, short-range map

it is planning to take, and information about obstacles, clear ground, and unseen regions. The information needs to be easy to understand even by people who have no understanding of the way the control system of the robot works, and should enable them to halt the vehicle only if it is about to take an action that will cause damage.

In order to display all the information in an understandable way, it is necessary to use color to represent the different types of region. Figure 20 shows the common color scheme NIST developed that is being used for the LAGR program. The color scheme was distributed to the teams in December 2006 for use by the January 2007 test. Use of the color scheme has had the desired effect. The Government evaluation team can more easily understand the OCUs of different teams. It was a useful, if somewhat tedious process to develop the common color scheme. Work needs to be done more broadly to develop common color schemes for different application areas, such as medical images, geographic information systems, and other complex visual displays that require substantial effort to understand.

Also in the second phase, NIST has defined interfaces in a "best-of" LAGR controller for performers to send their best algorithms to NIST for plug-and-play testing against other performers' algorithms. This work is currently ongoing and expected to be completed in 2008.

3 Standards and Performance Measurements

3.1 Autonomy Levels for Unmanned Systems (ALFUS)

The ICMS Program emphasizes to the mobile robotics research community a standard set of related terminology and representation of knowledge. The Autonomy Levels for Unmanned Systems (ALFUS) Ad Hoc Work Group [28], led by NIST, aims at developing a framework to facilitate the characterization of the autonomy capabilities of unmanned systems (UMS). The Group was formed in 2003 to respond to the user community needs, including many Federal Agencies and private industry. The group has been holding quarterly workshops ever since.

An ALFUS premise is that the autonomous capabilities are characterized with three significant aspects: mission complexity, environmental complexity, and human independence, as shown in Fig. 21. Detailed metrics, in turn, further characterize each of the three aspects.

ALFUS is collaborating with the U.S. Army Future Combat System (FCS) (http://www.army.mil/fcs/) and a number of other UMS programs on the issues of autonomy requirements, testing, and evaluation.

Fig. 21. ALFUS framework contextual autonomous capability model

Interim results have been published [21–23]. They have been significantly referenced in various public documents, including the ASTM Standards E2521-07, F 2395-05, and F 2541-06 for various UMSs as well as the U.S. Army UNMANNED and AUTONOMOUS SYSTEMS TESTING Broad Agency Announcement.

3.2 Joint Architecture for Unmanned Systems (JAUS)

The Joint Architecture for Unmanned Systems (JAUS) is a standard for interoperability between components of unmanned robotic vehicle systems such as unmanned ground, air and underwater vehicles (UGV, UAV and UUV, respectively). JAUS is sponsored by the Office of the Under Secretary of Defense for Acquisition, Technology and Logistics through the Joint Ground Robotics Enterprise (JGRE). It is mandated for use by all JGRE programs. The goals of JAUS are to reduce life cycle costs, reduce development and integration time, provide a framework for technology insertion, and accommodate expansion of existing systems with new capabilities. JAUS is a standard published by the Society of Automotive Engineers (SAE) via their Aerospace Avionics Systems Division AS-4 committee on Unmanned Systems.

JAUS is a component based, message-passing architecture that specifies data formats and methods of communication among computing nodes. It defines messages and component behaviors that are independent of technology, computer hardware, operator use, and vehicle platforms and isolated from mission. It uses the SAE Generic Open Architecture (GOA) framework to classify the interfaces.

JAUS benefits from an active user and vendor community, including Government programs such as FCS, academia such as University of Florida and Virginia Polytechnic Institute and State University (Virginia Tech), and industry such as Applied Perception, Autonomous Solutions, Kairos Autonomi, OpenJAUS, RE2 and TORC Technologies, which can easily be identified with a search on the internet. Users define requirements and sponsor pilot projects. Together with vendors, they participate in testing that helps JAUS evolve toward better performance and to support new technology.

JAUS products include commercially-available ground robots as well as software development kits (SDKs) that help robot vendors put a JAUS-compliant interface on their new or legacy products. Open-source implementations of JAUS exist that serve as reference implementations and help speed deployment.

NIST was one of two teams that participated in an early Red Team/Blue Team interoperability test. The test objective was to determine how well the JAUS specification enabled independent implementors to build JAUS-compliant systems that worked seamlessly together. For this particular test, one team built a JAUS component that continually produced vehicle position data. The other team built a component that displayed the current vehicle position in real time. Neither team knew the identity of the other, and interacted only with the JGRE test sponsor. The sponsor provided information necessary for interoperability but not part of the JAUS specification, such as the particular communication mechanism (in this case, TCP/IP Ethernet). The sponsor also resolved ambiguities in the specification that resulted in different interpretations of how JAUS works. These resolutions were provided to the working groups responsible for the specification for incorporation into the next versions of the relevant documents.

Since those early tests, the JAUS working group has expanded their test activities. These are now conducted by the Experimental Test Group (ETG). The ETG is chartered with implementing and testing

proposed changes or extensions of the JAUS specification, and reporting back with recommendations for how these proposals should be formalized into the specification. The ETG is comprised of a core group of JAUS experts from the vendor community who are well equipped to quickly build and test new JAUS capabilities. The JAUS ETG has invested in infrastructure development, such as setting up a distributed virtual private network that allows participants to connect to a JAUS system from any location.

3.3 The Intelligent Systems (IS) Ontology

The level of automation in ground combat vehicles being developed for the Army's objective force is greatly increasing over the Army's legacy force. This automation is taking many forms in emerging ground vehicles, varying from operator decision aides to fully autonomous unmanned systems. The development of these intelligent systems requires a thorough understanding of all of the intelligent behavior that needs to be exhibited by the system so that designers can allocate functionality to humans and/or machines. Traditional system specification techniques focus heavily on the functional description of the major systems of a vehicle and implicitly assume that a well-trained crew would operate these systems in a manner to accomplish the tactical mission assigned to the vehicle. In order to allocate some or all of these intelligent behaviors to machines in future ground vehicles, it is necessary to be able to identify and describe these intelligent behaviors.

The U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC) has funded NIST to explore approaches to model the ground vehicle domain with explicit representation of intelligent behavior. This exploration has included the analysis of modeling languages (i.e., UML, DAML, OWL) as well as reference architectures. A major component of this effort has been the development of an Intelligent Systems (IS) Ontology.

NIST has taken the view that an IS can be viewed as a multi-agent system, where agents can represent components within the vehicle (e.g., a propulsion system, a lethality system, etc.). In addition, an Intelligent Ground Vehicle (IGV), as a whole, can serve as a single agent within a troop, platoon, or section, where multiple IGVs are present. In order for a group of agents to work together to accomplish a common goal, they must be able to clearly and unambiguously communicate with each other without the fear of loss of information or misinterpretation. The IS Ontology has been used to specify a common lexicon and semantics to address this challenge.

The IS Ontology uses that OWL-S upper ontology [24] as the underlying representation to document 4D/RCS in a more open XML (eXtensible Markup Language) format. OWL-S is a service ontology, which supplies a core set of markup language constructs for describing the properties and capabilities of services in an unambiguous, computer-interpretable format. This ontology has been built within the Protégé framework [25], which is an ontology editor, a knowledge-base editor, as well as an open-source, Java tool that provides an extensible architecture for the creation of customized knowledge-based applications.

The IS Ontology is based on the concept of agents, service that the agents can perform, and procedures that the agents follow to perform the services. Figure 22 shows an agent hierarchy for a light cavalry troop. A detailed description of this hierarchy is outside the scope of this chapter. Also specified in the constraints for each class is whom each agent can send external service requests to and who they can receive them from. Any activity that can be called by another agent is considered a service in OWL-S. Any activity that the agent performs internally that cannot be externally requested is called a process. As such, "Conduct A Tactical Road March to an Assembly Area" is modeled as a service that is provided by a Troop agent (and can be called by a Squadron agent). The Troop agent can call services provided by other agents. In this example, a service called "Conduct Route Reconnaissance" is defined and associated with the Scout Platoon agent.

Process models in OWL-S are used to capture the steps that must be accomplished to carry out the service, and the ordering constraints on those steps. Each step can be performed internally by the agent or could involve making an external service request (a service call) to another agent. OWL-S provides a number of control constructs that allow one to model just about any type of process flow imaginable. Control constructs provided in OWL-S have been sufficient to model all the behaviors explored to date.

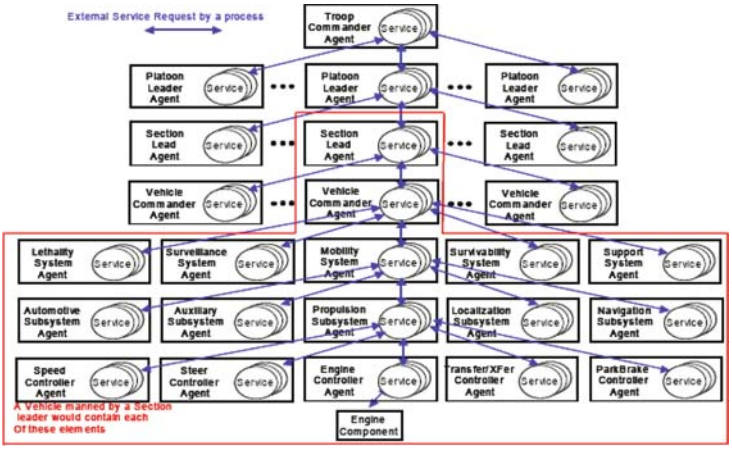

Fig. 22. Agent hierarchy

Environmental entities and their attributes are a primary output of the 4D/RCS methodology. These include other vehicles, bridges, vegetation, roads, water bodies; anything that is important to perceive in the environment relative to task that is being performed. An environment ontology in OWL-S has been built from the bottom up (i.e., including only entities that prove to be important). Environmental ontologies have started to be explored to see what could be leveraged.

3.4 DOT Integrated Vehicle Based Safety System (IVBSS)

The Transportation Project within the ICMS Program, although not 4D/RCS based, is an important part of the program. While autonomous vehicles on the nations highway are still many years in the future, components of intelligent mobility systems are finding their way into commercial crash warning systems (CWS). The US Department of Transportation, in attempt to accelerate the deployment of CSW, recently initiated the Integrated Vehicle-Based Safety System (IVBSS) program designed to incorporate forward collision, side collision and road-departure warning functions into a single integrated CSW for both light vehicles and heavy trucks [26]. Current analyses estimate that IVBSS has the potential to address 3.6 M crashes annually, of which 27,500 result in one or more fatalities.

NIST role in the IVBSS program is to assist in the development of objective tests and to evaluate system performance independently during the tests. NIST approach for measurement-based performance evaluation starts by developing a set of scenarios describing the problem, which in this case evolved from a DOT analysis of the crash database statistics. The scenarios lead to a set of track- and road-based test procedures to determine the system's effectiveness in dealing with the problems. Current plans call for carrying out over 34 track-based tests. Figure 23 provides an example of a multiple threat scenario that forces the system to recognize and respond to both a forward- and side-collision situation.

The pass/fail criteria for the tests include metrics that quantify acceptable performance. Various Crash Prevention Boundary (CPB) equations [27] determine the acceptable time for a warning; a driver cannot respond to late warnings and early warning may annoy the driver. To promote objectivity further, the tests rely on an independent measurement system (IMS) to provide performance data as opposed to using measurements taken by the system under test. NIST is currently developing a third generation IMS that incorporates calibrated cameras (for lane geometry measurements) and two laser-scanners (for obstacle range measurements). Figure 24 shows the sensor package mounted on the front of the NIST/DOT test bed vehicle.

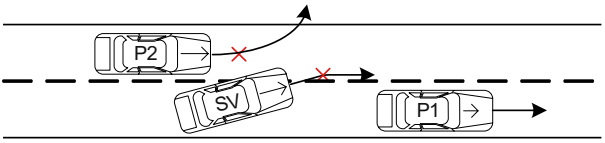

Fig. 23. Multiple threat test scenario. SV (equipped with IVBSS) encounters stopped vehicle (P1) and attempts to avoid collision by changing lanes into second vehicle (P2)

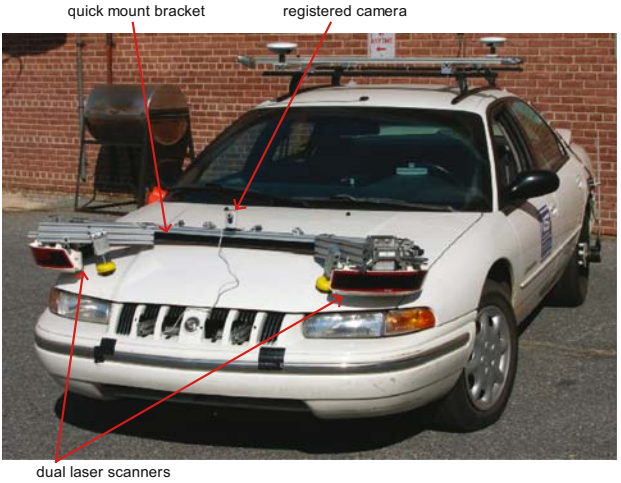

Fig. 24. IMS consisting of calibrated cameras and laser-scanners mounted on the NIST/DOT testbed vehicle. Program goals for this year include track testing and on-road testing of an IVBSS developed for light vehicles and a second IVBSS developed for heavy truck

The IMS design allows for quick installation on cars and trucks. Data is collected in real time and a suite of software exists for post-process analysis.

NIST is developing static and dynamic accuracy tests to verify the IMS meets requirements that range error be less than 5% of the actual range. Static tests evaluate sensor performance from a stationary position with targets at different ranges, reflectivity and orientations. The mean errors (mean of differences between measured range and actual range) serve as a calibration factor and the standard deviations of the errors define the uncertainty. Dynamic tests yield insight into sensor performance when operated from a fast moving vehicle (e.g., highway speeds). No standard procedure exists for dynamic testing and NIST is working on a novel approach involving precise time measurement, using a GPS receiver, of when a range reading takes place. For a dynamic test, an optical switch mounted on the vehicle senses when the vehicle crosses over reflectors placed on a track at known distances from a target. The switch causes a GPS receiver to latch the GPS time. The approach requires that all measurements, for example video streams, laser-scanner range measurements, etc., be stamped using GPS time (which is the case with the IVBSS warning system and the IMS). The range error comes from the difference (at the GPS time the vehicle crosses the reflector) between the laser-scanner's measured range to the target and the known range. We compute a mean error and uncertainty from measurements taken over several laps around a track. Current results indicate the laser scanner uncertainty is approximately 1% of the actual range.

4 Testbeds and Frameworks

4.1 USARSim/MOAST Framework

Many of the systems described in this chapter may be viewed as intelligent embodied agents. These agents require an environment to operate in, an embodiment that allows them to affect and move in the environment, and intelligence that allows them to execute useful behaviors that have the desired outcome in the environment. There are many aspects of the development of these agents in which simulations can play a useful role. If correctly implemented, simulation can be an effective first step in the development and deployment of new algorithms. Simulation environments enable researchers to focus on the algorithm development without having to worry about hardware aspects of the robots such as maintenance, availability, and operating space. Simulation provides extensive testing opportunities without the risk of harm to personnel or equipment. Major components of the robotic architecture (for example, advanced sensors) that may be too expensive for an institution to purchase can be simulated and enable the developers to focus on algorithm development. The remainder of this section will present an overview of a control framework that provides all three aspects of the embodied agent. Urban Search and Rescue Simulation (USARSim) provides the environment and embodiment, while Mobility Open Architecture Simulation and Tools (MOAST) provides the intelligence.

Urban Search and Rescue Simulation (USARSim)

The current version of Urban Search and Rescue Simulation (USARSim) [2] is based on the UnrealEngine2[1] game engine that was released by Epic Games as part of UnrealTournament 2004. This engine may be inexpensively obtained by purchasing the Unreal Tournament 2004 game. The USARSim extensions may then be freely downloaded from *sourceforge.net/projects/usarsim*. The engine handles most of the basic mechanics of simulation and includes modules for handling input, output (3D rendering, 2D drawing, and sound), networking, physics and dynamics. USARSim uses these features to provide controllable camera views and the ability to control multiple robots. In addition to the simulation, a sophisticated graphical development environment and a variety of specialized tools are provided with the purchase of Unreal Tournament.

The USARSim framework builds on this game engine and consists of:

- Standards that dictate how agent/game engine interaction is to occur
- Modifications to the game engine that permit this interaction
- An Application Programmer's Interface (API) that defines how to utilize these modifications to control an embodied agent in the environment
- 3-D immersive test environments
- Models of several commercial and laboratory robots and effectors
- Models of commonly used robotic sensors

The USARSim interaction standards consist of items such as robot coordinate frame definitions and unit declarations while the API specifies the command vocabulary for robot/sensor control and feedback. Both of these items have become the *de facto* standard interfaces for use in the RoboCup Rescue Virtual Competition which utilizes USARSim to provide an annual Urban Search and Rescue competition. In 2007 this competition had participation from teams representing five countries.

Highly realistic environments are also provided with the USARSim release. Example indoor and outdoor environments may be seen in Fig. 25a, b. In addition, a provided editor and the ability to import models simplifies the creation of additional worlds. In addition to environments, USARSim provides numerous robot and sensor models. Figure 26 shows the virtual version of the Talon robot. This robot features a simulated

[1] Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

(a) Indoor office scene (b) Outdoor accident scene

Fig. 25. Sample USARSim environments

Fig. 26. Simulated Talon Robot

track and arm with gripper. In addition to laboratory robots, aerial, road vehicles, commercial robots (both for manufacturing and bomb disposal), and robotic arms are modeled.

USARSim does not provide a robot controller. However, several open source controllers may be freely downloaded. These include the community developed Mobility Open Architecture Simulation and Tools (MOAST) controller (sourceforge.net/projects/moast), the player middle-ware (sourceforge.net/projects/playerstage), and any of the winning controllers from previous year's competitions (2006 and 2007 winning controllers may be found on the robocup rescue wiki (www.robocuprescue.org/wiki). A description of the winning algorithms may be found in [1].

Mobility Open Architecture Simulation and Tools (MOAST)

MOAST is a framework that provides a baseline infrastructure for the development, testing, and analysis of autonomous systems that is guided by three principles:

- Create a multi-agent simulation environment and tool set that enables developers to focus their efforts on their area of expertise without having to have the knowledge or resources to develop an entire control system.
- Create a baseline control system which can be used for the performance evaluation of the new algorithms and subsystems.
- Create a mechanism that provides a smooth gradient to migrate a system from a purely virtual world to an entirely real implementation.

- MOAST has the 4D/RCS architecture at its core (described elsewhere in this chapter) and consists of the additional components of control modules, interface specs, tools, and data sets. MOAST is fully integrated with the USARSim simulation system and may communicate with real hardware through the Player interface.

MOAST provides an implementation of primitive echelon through the section echelon of the 4D/RCS reference model architecture. This implementation is not designed to be complete, but rather is designed to provide examples of control strategies and a starting point for further research. The framework provides methods of mobility control for vehicles including Ackerman steered, skid steered, omni-drive, helicopter-type flying machines, and water craft. Control modalities increase in complexity as one moves up the hierarchy and include velocity/steering angle control, waypoint following, and an exploration behavior.

All of the control modules are designed to be self-contained and fully accessible through well-defined interfaces. This allows a developer to create a module that conforms to the specifications and replace any MOAST provided system. The idea is to allow a researcher to utilize all of the architecture except for the modules that are in the area of their expertise. These modules would be replaced with their own research code.

In order to simplify this replacement, several debug and diagnostic tools are also provided. These allow for unit testing of any module by providing a mechanism to send any command, status, and data into a module that is under test. In this way, a module may be fully and repeatedly tested. Once the system is debugged in simulation, the standardized interfaces allow the user to slowly move systems from simulation to actual hardware. For example, planning algorithms may be allowed to control the actual vehicle while sensing may be left in simulation.

4.2 PRediction in Dynamic Environments (PRIDE) Framework

There have been experiments performed with autonomous vehicles during on-road navigation. Perhaps the most successful was that of Prof. Dr. Ernst Dickmanns [33] as part of the European Prometheus project in which the autonomous vehicle performed a trip from Munich to Odense (>1,600 km) at a maximum velocity of 180 km h^{-1}. Although the vehicle was able to identify and track other moving vehicles in the environment, it could only make basic predictions of where those vehicles were expected to be at points in the future, considering the vehicle's current velocity and acceleration.

What is missing from all of these experiments is a level of situation awareness of how other vehicles in the environment are expected to behave considering the situation in which they find themselves. To date, the authors are not aware of any autonomous vehicle efforts that account for this information when performing path planning. To address this need, a framework, called PRIDE (PRediction in Dynamic Environments) was developed that provides an autonomous vehicle's planning system with information that it needs to perform path planning in the presence of moving objects [34]. The underlying concept is based upon a multi-resolutional, hierarchical approach that incorporates multiple prediction algorithms into a single, unifying framework. This framework supports the prediction of the future location of moving objects at various levels of resolution, thus providing prediction information at the frequency and level of abstraction necessary for planners at different levels within the hierarchy. To date, two prediction approaches have been applied to this framework.

At the lower levels, estimation theoretic short-term predictions is used via an extended Kalman filter-based algorithm using sensor data to predict the future location of moving objects with an associated confidence measure [35]. At the higher levels of the framework, moving object prediction needs to occur at a much lower frequency and a greater level of inaccuracy is tolerable. At these levels, moving objects are identified as far as the sensors can detect, and a determination is made as to which objects should be classified as "objects of interest". In this context, an object of interest is an object that has a possibility of affecting the path in the planning time horizon. Once objects of interest are identified, a moving object prediction approach based on situation recognition and probabilistic prediction algorithms is used to predict where object will be at various time steps into the future. Situation recognition is performed using spatio-temporal reasoning and pattern matching with an a priori database of situations that are expected to be seen in the environment.

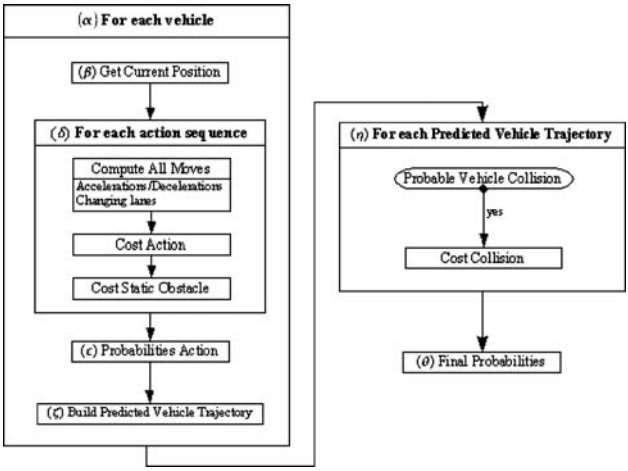

Fig. 27. Moving object prediction process

The algorithms are used to predict the future location of moving objects in the environment at longer time planning horizons on the order of tens of seconds into the future with plan steps at about one second intervals.

The steps within the algorithm shown in Fig. 27 are:

- For each vehicle on the road (α), the algorithm gets the current position and velocity of the vehicle by querying external programs/sensors (β).
- For each set of possible future actions (δ), the algorithm creates a set of next possible positions and assigns an overall cost to each action based upon the cost incurred by performing the action and the cost incurred based upon the vehicle's proximity to static objects. An underlying cost model is developed to represent these costs.
- Based upon the costs determined in Step 2, the algorithm computes the probability for each action the vehicle may perform (ε).
- Predicted Vehicle Trajectories (PVT) (ξ) are built for each vehicle which will be used to evaluate the possibility of collision with other vehicles in the environment. PVTs are a vector that indicates the possible paths that a vehicle will take within a predetermined number of time steps into the future.
- For each pair of PVTs (η), the algorithm checks if a possible collision will occur (where PVTs intersect) and assigns a cost if collision is expected. In this step, the probabilities of the individual actions (θ) are recalculated, incorporating the risk of collision with other moving objects.

At the end of the main loop, the future positions with the highest probabilities for each vehicle represent the most likely location of where the vehicles will be in the future. More information about the cost-based probabilistic prediction algorithms can be found in [35].

4.3 Industrial Automated Guided Vehicles

Study of Next Generation Manufacturing Vehicles

This effort, called the Industrial Autonomous Vehicles (IAV) Project, aims to provide industries with standards, performance measurements, and infrastructure technology needs for the material handling industry.

The NIST ISD have been working with the material handling industry, specifically on automated guided vehicles (AGVs), to develop next generation vehicles. A few example accomplishments in this area include: determining the high impact areas according to the AGV industry, partnering with an AGV vendor to demonstrate pallets visualization using LADAR towards autonomous truck unloading, and demonstrating autonomous vehicle navigation through unstructured facilities. Here, we briefly explain each of these points.

Generation After Next AGV

NIST recently sponsored a survey of AGV manufacturers in the US, conducted by Richard Bishop Consulting, to help determine their "generation-after-next" technology needs. Recognizing that basic engineering issues to enhance current AGV systems and reduce costs are being addressed by AGV vendors, the study looks beyond today's issues to identify needed technology breakthroughs that could open new markets and improve US manufacturing productivity. Results of this study are described in [36].

Within the survey and high on the list, AGV vendors look to the future for: reduced vehicle costs, navigation in unstructured environments, onboard vehicle processing, 3D imaging sensors, and transfer of advanced technology developed for Department of Defense. Current AGVs are "guided" by wire, laser or other means, operate in structured environments tailored to the vehicle, have virtually no 3D sensing and operate from a host computer with limited onboard-vehicle control.

Visualizing Pallets

Targeting the high impact area of using 3D imaging sensors on AGV, NIST ISD teamed with Transbotics, an AGV vendor, to visualize pallets using panned line-scan LADAR towards autonomous truck unloading [37]. A cooperative agreement between NIST and Transbotics allowed NIST to: (1) set up mock pallets, conveyer and truck loading on a loading dock, (2) to develop software to visualize pallets, the conveyer and the truck in 3D space, and (3) verify if the pallet, conveyor and truck are in their expected location with respect to the AGV. The project was successful on mock components used at NIST and software was transferred to Transbotics for implementation on their AGV towards use in a production facility.

Navigation Through Unstructured Facilities

Also targeting a high impact AGV industry requested area, the ICMS Program has been transferring technology from defense mobility projects through its IAV Project to the AGV industry. By focusing on AGV industry related challenges, for example autonomous vehicle navigation through unstructured facilities [38], the IAV project attempts to provide improved AGV capabilities to do more than point–to–point, part pick-up/delivery operations. For example, AGV could avoid obstacles and people in the vehicle path, adapt to facilities instead of vice versa, navigate both indoors and outdoors using the same adaptable absolute vehicle position software modules – all towards doing more with end users' vehicle capital investments and developing niche markets.

A number of changes were made to the LAGR control system software in order to transfer the military outdoor vehicle application to an indoor industrial setting. Two RFID sensors, batteries, laptop, and network hub were added. Active RFID sensors were integrated into the vehicle position estimate. Also, a passive RFID system was used including tags that provide a more accurate vehicle position to within a few centimeters. RFID systems updates replaced the outdoor GPS positioning system updates in the controller.

The control system also needed to be less aggressive for safety of people and equipment, use stereo vision indoors, negotiate tighter corners than are typically encountered outdoors, display facility maps and expected paths (see Fig. 28), and many other modifications detailed in [38]. The demonstration was successful and allowed the AGV industry to view how vehicles could adapt to a more cluttered facility than AGVs typically navigate.

Future research will include integration of a 2D safety sensor to eliminate false positives on obstacles near ground level caused by low stereo disparity. Demonstration of controlling more than one intelligent vehicle at a time in the same unstructured environment along with other moving obstacles is also planned.

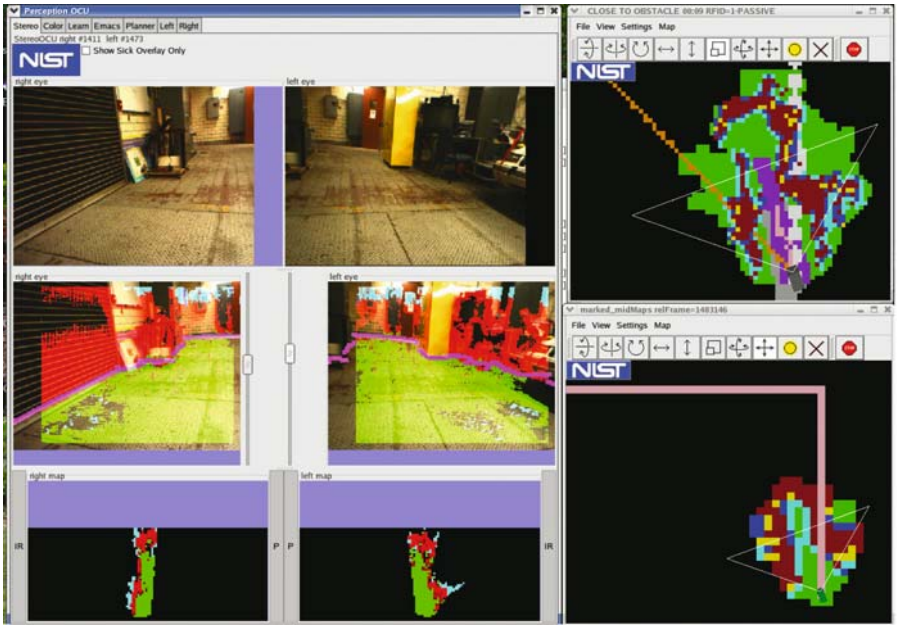

Fig. 28. LAGR AGV Graphical Displays – *right* and *left* stereo images (*upper left*); images overlaid with stereo obstacle (*red*) and floor (*green*) detection and 2D scanner obstacle detection (*purple*) (*middle left*); *right* and *left* cost maps (*lower left*); low level map (*upper right*); and high level map (*lower right*)

5 Conclusions and Continuing Work

The field of autonomous vehicles has grown tremendously over the past few years. This is perhaps most evident by the performance of these vehicles in the DARPA-sponsored Grand Challenge events which occurred in 2004, 2005 and most recently in 2007 [39]. The purpose of the DARPA Grand Challenge was to develop autonomous vehicle technologies that can be applied to military tasks, notably robotic "mules" or troop supply vehicles. The Grand Challenge courses gradually got harder, with the most recent event incorporating moving on-road objects in the urban environment. The 2007 Challenge turned out to have more civilian focus than military's, with the DARPA officials and many teams emphasizing safe robotic driving as a very important objective. The performance of the vehicles improved tremendously from 2004 to 2007, even as the environment got more difficult. This is in part due to the advancement of technologies that are being explored as part of the ICMS program. The ICMS Program and its development of 4D/RCS has been ongoing for nearly 30 years with the goal to provide architectures and interface standards, performance test methods and data, and infrastructure technology needed by US manufacturing industry and government agencies in developing and applying intelligent control technology to mobility systems to reduce cost, improve safety, and save lives.

The 4D/RCS has been the standard intelligent control architecture on many of the Defense, Learning, and Industry Projects providing application to respective real world issues. The Transportation Project provides performance analysis of the latest mobile system sensor advancements. And the Research and Engineering Projects allow autonomy capabilities to be defined along with simulation and prediction efforts for mobile robots.

Future ICMS efforts will focus deeper into these projects with even more autonomous capabilities. Broader applications to robots supporting humans in manufacturing, construction, and farming are expected once major key intelligent mobility elements in perception and control are solved.

References

1. Albus, J.S., Huang, H.-M., Messina, E., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S., Shneier, M.O., Hong, T., Scott, H., Horst, J., Proctor, F., Shackleford, W., Szabo, S., and Finkelstein, R., 4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems, NIST, Gaithersburg, MD, NISTIR 6912, 2002
2. Balakirsky, S., Messina, E., Albus, J.S., Architecting a Simulation and Development Environment for Multi-Robot Teams, Proceedings of the International Workshop on Multi Robot Systems, 2002
3. Balakirsky, S.B., Chang, T., Hong, T.H., Messina, E., Shneier, M.O., A Hierarchical World Model for an Autonomous Scout Vehicle, Proceedings of the SPIE 16th Annual International Symp. on Aerospace/Defense Sensing, Simulation, and Controls, Orlando, FL, April 1–5, 2002
4. Albus, J.S., Juberts, M., Szabo, S., RCS: A Reference Model Architecture for Intelligent Vehicle and Highway Systems, Proceedings of the 25th Silver Jubilee International Symposium on Automotive Technology and Automation, Florence, Italy, June 1–5, 1992
5. Bostelman, R.V., Jacoff, A., Dagalakis, N.G., Albus, J.S., RCS-Based RoboCrane Integration, Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, October 20–23, 1996
6. Madhavan, R., Messina, E., and Albus, J. (Editors), Low-Level Autonomous Mobility Implementation part of Chapter 3: Behavior Generation in the book, Intelligent Vehicle Systems: A 4D/RCS Approach, 2007
7. Jackel, Larry, LAGR Mission, http://www.darpa.mil/ipto/programs/lagr/index.htm, DARPA Information Processing Technology Office
8. Albus, J., Bostelman, R., Chang, T., Hong, T., Shackleford, W., and Shneier, M., 2006. Learning in a Hierarchical Control System: [4D/RCS in the DARPA LAGR Program. Journal of Field Robotics, Special Issue on Learning in Unstructured Environments, 23(11/12): 975–1003.]
9. Konolige, K., SRI Stereo Engine, http://www.ai.sri.com/∼konolige/svs/
10. Tan, C., Hong, T., Shneier, M., and Chang, T., Color Model-Based Real-Time Learning for Road Following, in Proceedings of the IEEE Intelligent Transportation Systems Conference (Submitted) Toronto, Canada, 2006
11. Shneier, M., Chang, T., Hong, T., Shackleford, W., Bostelman, R., and Albus, J. S. Learning traversability models for autonomous mobile vehicles. Autonomous Robots 24, 1, January 2008, 69–86.
12. Oskard, D., Hong, T., Shaffer, C., Real-time Algorithms and Data Structures for Underwater Mapping, Proceedings of the SPIE Advances in Intelligent Robotics Systems Conference, Boston, MA, November, 1988
13. Shackleford, W., The NML Programmer's Guide (C++ Version) *http://www.isd.mel.nist.gov/projects/rcslib/NMLcpp.html*
14. Heyes-Jones, J., A* algorithm tutorial, http://us.geocities.com/jheyesjones/astar.html
15. Tan, C., Hong, T., Shneier, M., Chang, T., Color Model-Based Real-Time Learning for Road Following, Proceedings of the IEEE Intelligent Transportation Systems Conference, 2006
16. He, Y., Wang, H., Zhang, B., Color-based road detection in urban traffic scenes, IEEE Transactions on Intelligent Transportation Systems, 5(4), 309–318, 2004
17. Kristensen, D., Autonomous Road Following. PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2004
18. Lin, X., Chen, S., Color image segmentation using modified HSI system for road following, IEEE International Conference on Robotics and Automation, 1991, pp. 1998–2003
19. Ulrich, I., Nourbakhsh, I., Appearance-Based Obstacle Detection with Monocular Color Vision, Proceedings of the AAAI National Conference on Artificial Intelligence, 2000
20. Shneier, M., Bostelman, R., Albus, J.S., Shackleford, W., Chang, T., Hong, T., A Common Operator Control Unit Color Scheme for Mobile Robots, National Institute of Standards and Technology, Gaithersburg, MD, August, 2007
21. Huang, H.-M., The Autonomy Levels for Unmanned Systems (ALFUS) Framework–Interim Results, in Performance Metrics for Intelligent Systems (PerMIS) Workshop, Gaithersburg, Maryland, 2006
22. Huang, H.-M. et al., Characterizing Unmanned System Autonomy: Contextual Autonomous Capability and Level of Autonomy Analyses, in Proceedings of the SPIE Defense and Security Symposium, April, 2007
23. Huang, H.-M. ed., Autonomy Levels for Unmanned Systems (ALFUS) Framework, Volume I: Terminology, NIST Special Publication 1011, Gaithersburg: National Institute of Standards and Technology, 2004

24. The OWL Services Coalition, "OWL-S 1.0 Release," http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2003
25. Schlenoff, C., Washington, R., and Barbera, T., Experiences in Developing an Intelligent Ground Vehicle (IGV) Ontology in Protege, Proceedings of the 7th International Protege Conference, Bethesda, MD, 2004
26. http://*www.its.dot.gov/ivbss*
27. Szabo, S., Wilson, B., Application of a Crash Prevention Boundary Metric to a Road Departure Warning System. Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) Workshop, NIST, Gaithersburg, MD, August, 24–26, 2004. http://www.isd.mel.nist.gov/documents/szabo/PerMIS04.pdf
28. *http://www.isd.mel.nist.gov/projects/autonomy_levels/*
29. Balakirsky, S., Scrapper, C., Carpin, S., and Lewis, M., USARSim: Providing a Framework for Multi-robot Performance Evaluation, Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) Workshop, 2006
30. Scrapper, C., Balakirsky, S., and Messina, E., MOAST and USARSim – A Combined Framework for the Development and Testing of Autonomous Systems, SPIE 2006 Defense and Security Symposium, 2006
31. USARSim Homepage. *http://usarsim.sourceforge.net/*, 2007
32. MOAST Homepage. *http://sourceforge.net/projects/moast/*, 2007
33. Dickmanns, E.D., A General Dynamic Vision Architecture for UGV and UAV, Journal of Applied Intelligence, 2, 251, 1992
34. Schlenoff, C., Ajot, J., and Madhavan, R., PRIDE: A Framework for Performance Evaluation of Intelligent Vehicles in Dynamic, On-Road Environments, Proceedings of the Performance Metrics for Intelligent Systems (PerMIS) 2004 Workshop, 2004
35. Madhavan, R. and Schlenoff, C., The Effect of Process Models on Short-term Prediction of Moving Objects for Autonomous Driving, International Journal of Control, Automation and Systems, 3, 509–523, 2005
36. Bishop, R., Industrial Autonomous Vehicles: Results of a Vendor Survey of Technology Needs, Bishop Consulting, February 16, 2006
37. Bostelman, R., Hong, T., Chang, T., Visualization of Pallets, Proceedings of SPIE Optics East 2006 Conference, Boston, MA, USA, October 1–4, 2006
38. Bostelman, R., Hong, T., Chang, T., Shackleford, W., Shneier, M., Unstructured Facility Navigation by Applying the NIST 4D/RCS Architecture, CITSA 06 Conference Proceedings, July 20–23, 2006
39. Iagnemma, K., Buehler, M., Special Issues on the DARPA Grand Challenge, Journal of Field Robotics, Volume 23, Issues 8 & 9, Pages 461–835, Aug/Sept 2006