Task Analysis of Autonomous On-road Driving

Tony Barbera[a], John Horst[a], Craig Schlenoff[a], David Aha[b]

[a]National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD, USA 20899
[b] Intelligent Decision Aids Group, Naval Research Laboratory, Washington, DC 20375

ABSTRACT

The Real-time Control System (RCS) Methodology has evolved over a number of years as a technique to capture task knowledge and organize it into a framework conducive to implementation in computer control systems. The fundamental premise of this methodology is that the present state of the task activities sets the context that identifies the requirements for all of the support processing. In particular, the task context at any time determines what is to be sensed in the world, what world model states are to be evaluated, which situations are to be analyzed, what plans should be invoked, and which behavior generation knowledge is to be accessed.

This methodology concentrates on the task behaviors explored through scenario examples to define a task decomposition tree that clearly represents the branching of tasks into layers of simpler and simpler subtask activities. There is a named branching condition/situation identified for every fork of this task tree. These become the input conditions of the if-then rules of the knowledge set that define how the task is to respond to input state changes. Detailed analysis of each branching condition/situation is used to identify antecedent world states and these, in turn, are further analyzed to identify all of the entities, objects, and attributes that have to be sensed to determine if any of these world states exist.

This paper explores the use of this 4D/RCS methodology in some detail for the particular task of autonomous on-road driving, which work was funded under the Defense Advanced Research Project Agency (**DARPA**) Mobile Autonomous Robot Software (**MARS**) effort (Doug Gage, Program Manager).

Keywords: sensory processing, task analysis, autonomous, driving, finite state machines, task knowledge, world model

INTRODUCTION

The National Institute of Standards and Technology (NIST) has researched and developed a design methodology for complex systems [1], one that emphasizes the identification of task domain knowledge and its subsequent system requirements definition. This methodology has been labeled the Real-time Control System (RCS) methodology – more recently referred to as the 4D/RCS [2] methodology with the 4D referring to the three spatial dimensions plus the time dimension. The RCS methodology attempts to deal with complex control from the point of view of the goal directed task behaviors. This changes the approach from a more traditional component-based functional design to a system-based task design methodology. Rather than identify the component functions of the task, this approach attempts to define the task knowledge as a description of the detailed coordination and sequencing of activities and output actions required to successfully accomplish all aspects of the goal tasks at all levels of task detail. The functional elements are instead interleaved at various appropriate levels of the task decomposition and related to the task description as a natural part of this analysis. This task description uses the concept of a hierarchical organization of agent control modules as the execution device to carry out the goal tasks in much the same way one can think of the hierarchical organization of people (these are the human agent control modules) in a business or the military carrying out those organizations' goal tasks.

The use of the generic agent control module as the structuring element to bound the knowledge at each layer of the task decomposition provides a formalized mechanism that creates well delineated (in authority and responsibility) containers in which to place newly discovered task knowledge. This paper will explore how this 4D/RCS task analysis is used to discover and structure the knowledge of the autonomous on-road driving task.

Mobile Robots XVII, edited by Douglas W. Gage, Proceedings of SPIE Vol. 5609
(SPIE, Bellingham, WA, 2004) · 0277-786X/04/$15 · doi: 10.1117/12.580233

1. 4D/RCS TASK ANALYSIS METHODOLOGY

The 4D/RCS methodology models real-time, goal oriented task control as containing three major processing components (figure 1) at each layer in the agent hierarchy [2]:

1) sensory-processing to measure, recognize, and classify entities and events of task interest in the environment;
2) internal world model processing that represents and derives world states, situations, and evaluations in a task context manner; and
3) behavior-generation processing that reasons from this world model, selects plans, and makes value judgments to decide on the next appropriate output action to accomplish the goal tasks.

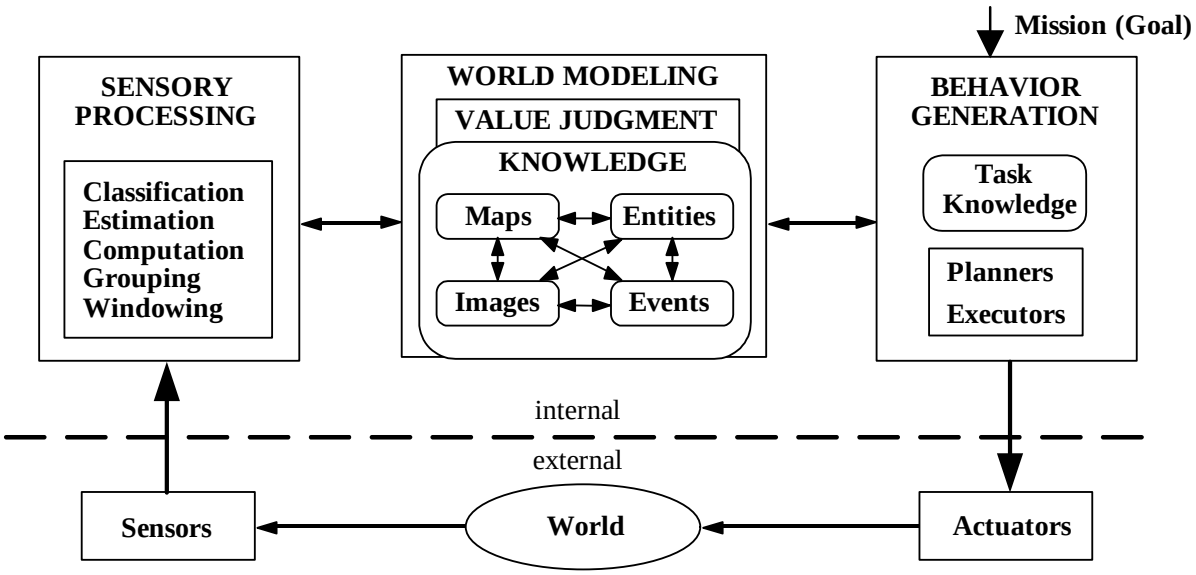

Figure 1. The basic internal structure of a 4D/RCS agent control module.

These three components work together, receiving a goal task, breaking it down into a set of simpler subtasks, determining what has to be known in the internal world model to decide on the next course of action, and alerting the sensory processing as to what internal world objects have to have their states updated by new sensory readings/measurements. This is the model of the internal processing for each agent control module that will formalize the knowledge representation at each level of the task decomposition. A system is structured as a hierarchical organization of these agent control modules, each of which does a partial task decomposition of its input goal task, and outputs simpler subtask goals to the next lower subordinate agent control module.

2.1 RCS Methodology Summary

The RCS methodology concentrates on the task decomposition as the primary means of understanding the knowledge required for intelligent control. This approach begins with the knowledge "mining" activities to retrieve knowledge from subject matter experts (SMEs). The gathering and formatting of this knowledge can be summarized in six steps, each of which follows from the knowledge uncovered by the previous steps (figure 2 presents a high level summary view of the overall approach which will be detailed step-by-step in the following sections):

1) The first step involves an intensive analysis of domain knowledge from manuals and SMEs, especially through the use of scenarios of particular subtask operations. The output of this effort is a structuring of this knowledge into a task tree form of simpler and simpler commands (actions/verbs) at simpler and simpler levels of task description. The importance of scenarios to drive the requirements' specifications has also been discussed by Mettala et al. [3].

2) Next, the hierarchical organization of agent control modules that will execute these layers of commands is defined. This is the same as coming up with a business or military organizational structure of agent control modules (people, soldiers) to accomplish the desired tasks. This step forces a more formal structuring of all of the subtask activities in the sense of defining which knowledge will reside with which agent control module.
3) This step clarifies how each agent's input command will decompose through the use of rules that identify all of the task branching conditions with their corresponding output commands (Input Condition - Output Action rules). Each of these command decompositions at each agent control module will be represented in the form of a state-table of ordered production rules (which is an implementation of an extended finite state machine (FSM)). The ordered list of simpler output commands required to accomplish the input command and the named input condition-branching situations that transition the state-table to the next output command are the primary knowledge represented in this step.
4) The above named condition-branching situations are refined in great detail in terms of their dependencies on world and task states. This step attempts to define the detailed antecedent states of the world that cause a particular situation to be true.
5) Here, we identify and name all of the objects and entities together with their particular features and attributes that are relevant to defining the above world states and situations.
6) The last step is to use the context of the particular task activities to establish the required minimum sensing distances and, therefore, the resolutions at which the above objects and entities must be measured and recognized by the sensory processing component. This step establishes a set of requirements and/or specifications for the sensor system at the level of each separate subtask activity.

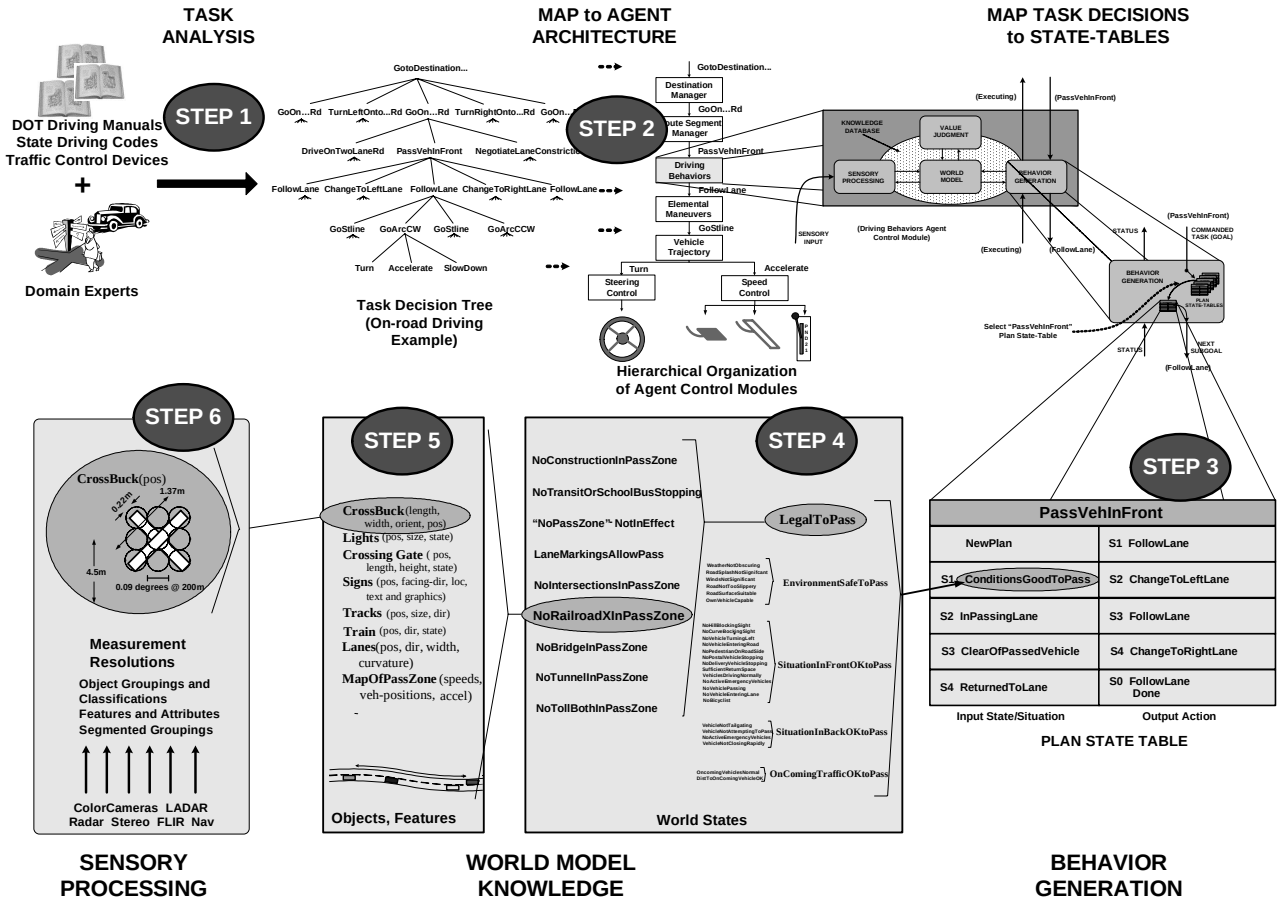

Figure 2. The six steps of the RCS methodology approach to knowledge acquisition and representation.

We will now cover these six steps in detail using the on-road driving example.

2.1.1 Step 1 – Task Decomposition Design

The first step is to gather as much task related knowledge as possible with the goal of defining a set of commands that incorporate all of the activities at all levels of detail. For on-road driving, this knowledge source would include driving manuals[4], state and federal driving codes, manuals on traffic control devices and detailed scenario narratives by Subject Matter Experts (SMEs) of large numbers of different driving experiences.

Many scenarios are explored in great detail in an attempt to come up with the names of commands that describe the activities at finer and finer resolutions of detail. Figure 3 provides an example. The high level goal of "Goto destination" (such as "go to post office") is broken down into a set of simpler commands – "GoOnRoad-*name*", "TurnLeft Onto-*name*" (MapQuest-like commands). At the next level down, these commands are broken down to simpler commands such as "Drive On Two Lane Road", "Pass Vehicle In Front" and these are then decomposed to yet simpler commands such as "FollowLane", ChangeToLeftLane",etc

Four very important things are being done with the knowledge in this step.

1) The first is the discovery and naming of simpler component subtasks that go into making up the more complex tasks.
2) The second is that for each of these component subtasks, we are defining a subtask command name.
3) The third is the understanding of the coordination of subtask activities that the task involves.
4) The fourth is the careful grouping of these commands by layer and decomposition to ensure that all of the examples of on-road driving tasks can be completely described, from the start to finish of a scenario, by the proper sequencing of these commands at the appropriate levels.

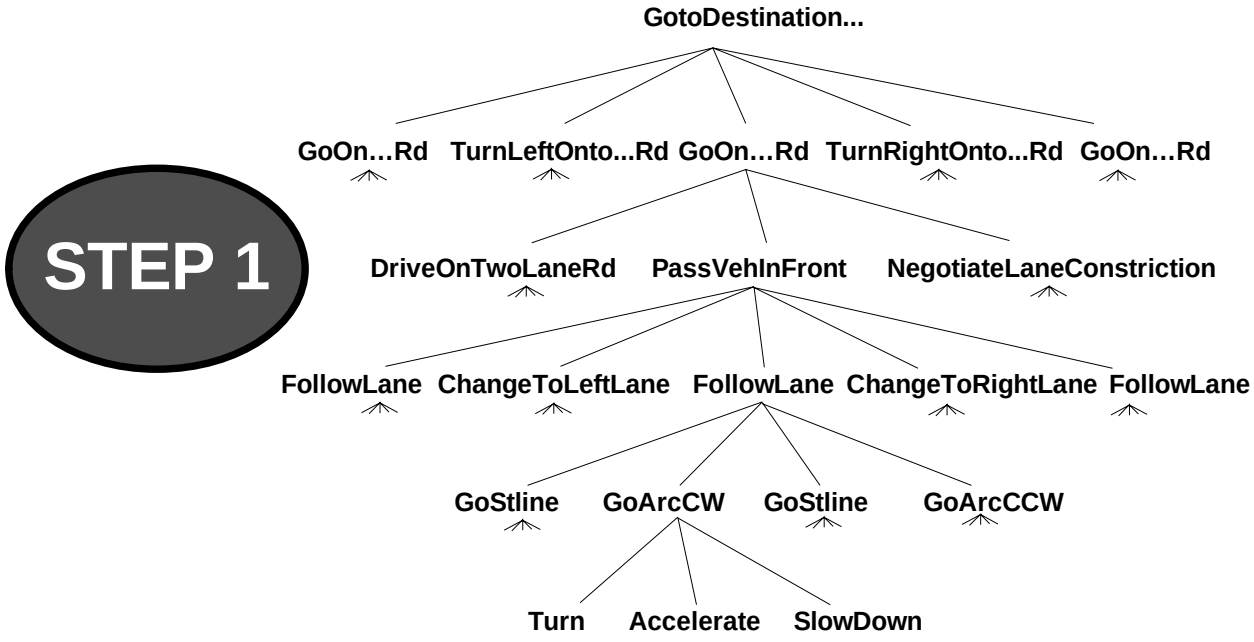

Figure 3. Task decomposition decision tree for on-road driving example. Shows the simpler commands that are used at each lower layer to represent the finer and finer resolutions of detail activities.

2.1.2 Step 2 – Agent Control Module Organization

Once a set of commands is defined, we need an organization to execute them. This step is identical to laying out an organizational structure of people in a business or the military. You know what you want to do at various levels of detail – now you need an organization of intelligent agents to do it. This structure is built from the bottom up. The above detailed task decomposition will tell us how many layers of agents to have in our organization but not how many agents at a level or how they are grouped and coordinated. This step starts at the bottom with an agent control module assigned to each actuator in the system and then uses the knowledge of the task activities to understand which subordinate agents are grouped under which supervisor to best coordinate the task commands from step 1.

Figure 4 illustrates how a grouping of agent control modules is assembled to accomplish the commands defined in step 1. In this example, the lowest level, servo agent control modules are represented by icons of the actuators being controlled. The steering servo control module is represented by a steering wheel icon, the brake servo by a brake pedal icon, etc. For this simple example, only four actuator agent control module icons are shown. The brake, throttle, and transmission servo agent control modules are grouped under a single supervisor agent control module, which we will call the Speed Control Agent. This supervisor agent control module will receive commands such as "Accelerate" at some value and has to coordinate its output commands to the brake, the throttle, and the transmission to accomplish this command. By a similar analysis, the Steering Servo Agent is placed under a supervisor agent we call the Steering Control Agent Module.

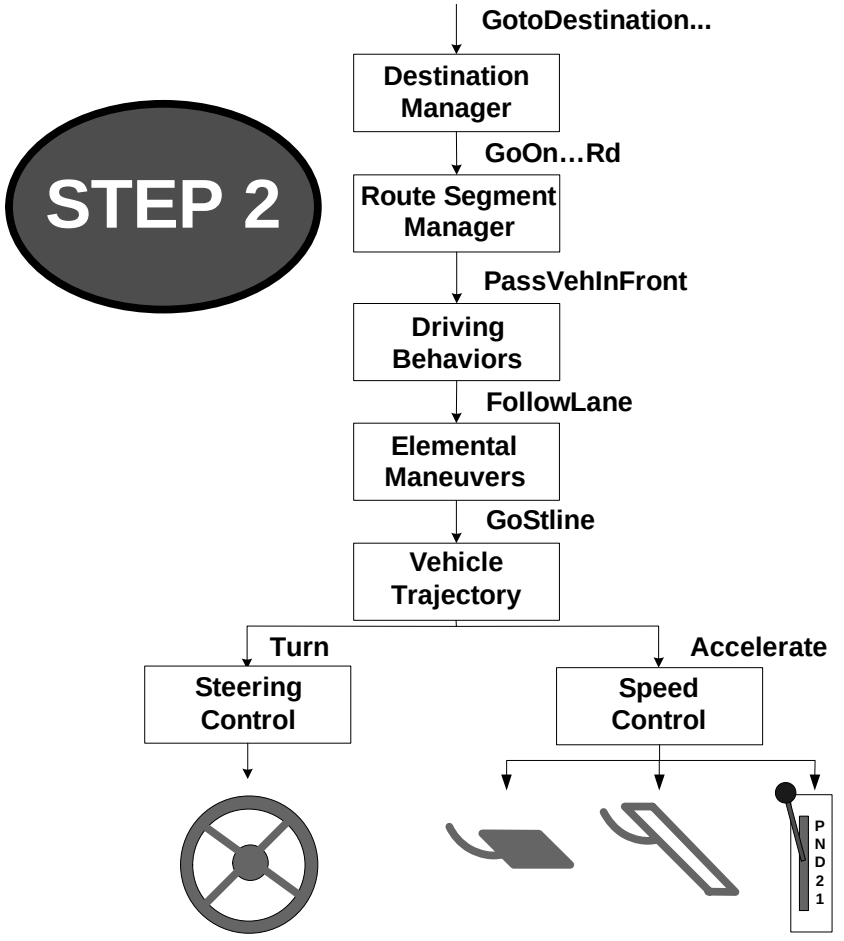

Figure 4. The hierarchical organization of agent control modules that are to execute the task command decomposition.

The Vehicle Trajectory Control Agent coordinates steering commands to the Steering Control Agent and speed commands to the Speed Control Agent to accomplish dynamically feasible moves. The command decomposition of the commands at levels above the Vehicle Trajectory Control Agent are shown being executed by a single agent at each layer since there are no more subordinate agent control modules to be coordinated at these levels in this simple example.

In a more detailed example implementation, there would be additional agent control modules for ignition and engine starting, lights, turn signals, windshield wiper/washer, pan/tilt turrets that carry the sensor sets, etc. This step 2 would be the point at which that organizational structure would be defined and laid out to properly coordinate those modules' activities in accordance with the task decomposition descriptions from step 1.

2.1.3 Step 3 – State-Table Definitions

At this stage of the knowledge definition process we know the vocabulary and syntax of commands. We also know what set of subcommands each command decomposes into, and where in the agent control hierarchy these decompositions take place. Step 3 is to establish the rules that govern each command's decomposition into its appropriate sequence of simpler output commands. These rules are discovered by first listing the approximate sequence set of output commands for a particular input command and then by identifying the conditions that choose when to execute these output commands.

Figure 5 illustrates this step with a state-table of the "Pass Veh(icle) In Front" command at the Driving Behaviors Agent Control Module. This PassVehInFront command is decomposed into five simpler output commands – "Follow Lane", "Change to Left Lane", "Follow Lane", "Change to Right Lane", and "Follow Lane". These output commands came

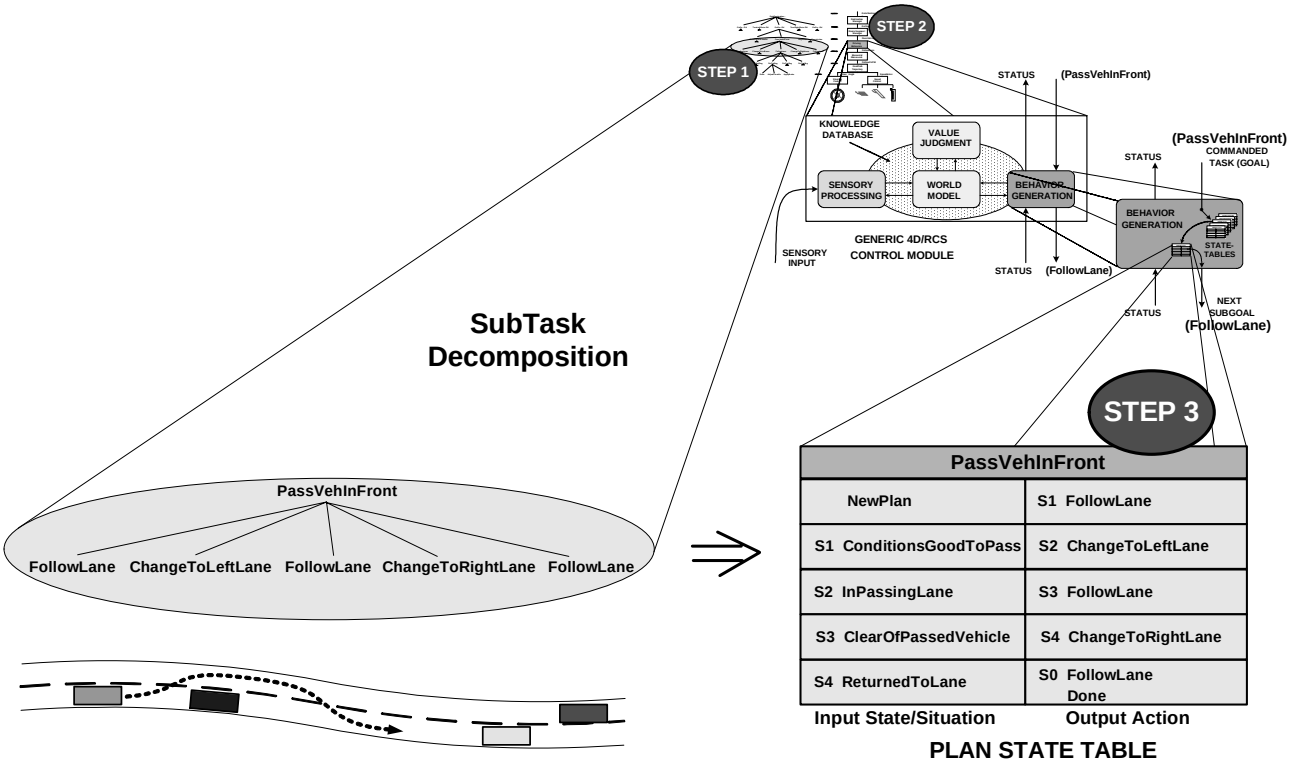

Figure 5. State-table for the Pass Vehicle In Front command. Each line is a rule relevant to this task. The left column contains the branching conditions of the task decision tree. The right column contains the output commands.

from the task decomposition tree for PassVehInFront identified in step1 and assigned to the particular agent control module where this state table resides in step 2. These output commands are placed in the right hand side or Output Action column of the state table. The knowledge that is being added by this step 3 is to identify and name the situations (Input State/Situation - the left hand column of the state-table) that will transition to each of these output commands. These named situations are the conditions that cause the branching in the task tree to execute this particular subtask.

Each of these newly named state transition situations with their corresponding output command actions represent a single production rule that is shown as a single line in the state-table. The sequence that these lines (rules) are executed is ordered by the addition of a state variable (S1, S2, etc). In the example in figure 5, the first rule shown in the state-table says that if this is a "New Plan" (input condition), then the output action side of the rule (the right hand side of the state-table) sets the state to "S1" and outputs the command to "Follow Lane". As a result of executing this rule, this agent module is now in state "S1" and can only execute rules that include the state value of "S1" in their input condition.

The only rules that will be searched by this module are those in the particular state-table that clusters the rules relating to this particular input command ("PassVehInFront"). In this state table as shown, there is only one line (rule) that contains the state value "S1" as one of its input conditions. Thus, only that line can match and it will not match until the situation "ConditionsGoodToPass" is also true. When this situation occurs, this line will match (this rule will fire) and the module will go to state "S2" and output the command to "ChangeToLeftLane". This output command is sent to the subordinate agent control module (Elemental Maneuvers Agent Control Module) where it becomes that module's input command invoking a corresponding state-table evaluation of that command.

Thus, the large set of rules governing the task decision tree execution is clustered both by the layer of resolution in the hierarchy and by the task context of the particular command at each layer so that only a very small number of rules that are relevant to the particular active command at each agent control module have to be searched at any given time. Within an individual agent, within its active state table, the execution order of these selected rules is controlled by the addition of state values. Again, it is important to note that this knowledge discovery, representation and organization have been completely driven by looking at the problem from the detailed task decomposition point of view.

We will now make an important summary about the structuring of the knowledge base in these first three steps. These three steps were concerned with task knowledge expressed as the finer and finer branching of the decision process whereby the output action is sequenced in order to accomplish the assigned task. This third step identifies named condition branching situations identified to encompass everything that the task depends upon at this point and at this level of resolution in its execution. In this example, it was named "ConditionsGoodToPass".

These first three steps provide a complete listing of the task decomposition rules (i.e. these rules that determine when the system has to do something different in order maintain progress towards the goal.) These rules have been grouped into layers of resolution, and within each layer, clustered into tables of rules relevant to a single input command. Within each table they are ordered in their execution sequence by additional state values.

We can think of these first three steps as identifying the knowledge involved in the task decomposition process, i.e. naming all of the task branching conditions and resultant corrective output actions. The next three steps will identify all of the knowledge that is used to evaluate whether or not the branching conditions are true.

2.1.4 Step 4 – Situation Dependencies on World States

In this step, the world knowledge we want to identify are those antecedent world states on which the task branching situations (in the input condition side of the state tables) depend in order to be evaluated. This is best illustrated with an example. Figure 6 shows the "PassVehInFront" state-table. The output command to "Change To Left Lane" is issued when the "ConditionsGoodToPass" situation occurs as described above.

Here, we ask of our expert knowledge sources "what do we have to know about the state of the world at this time to say that the conditions are good to pass". Again, we use a number of detailed scenarios and our knowledge sources to drill down to the parameters that go into this situation assessment. We find that there is a surprisingly large set of states of the world that affect this decision. We list these as they come up in scenarios and from manuals –e.g. "there cannot be

an on-coming car within the passing distance", "there must be a broken yellow lane marker on our side of center in the lane", "there cannot be a railroad crossing within the passing distance", "our own vehicle is not being passed", etc. We call these items world states since they seem to describe certain attributes about the present state of the world that are relevant to our present task.
Once we have listed the world states relevant to "ConditionsGoodToPass", we go over the list and see if groupings can be made. In this case, we group some world states into a category we call "LegalToPass", and others into other categories we call "EnvironmentSafeToPass", "SituationInFrontOkToPass", "SituationInBackOkToPass", "OncomingTrafficOKToPass", etc. These groupings are aggregate world states leading to the evaluation of the situation "ConditionsGoodToPass". For this example for this situation to be true, all of the aggregate world states have to be true. For each of the aggregate world states to be true, all of their component world states have to be true.

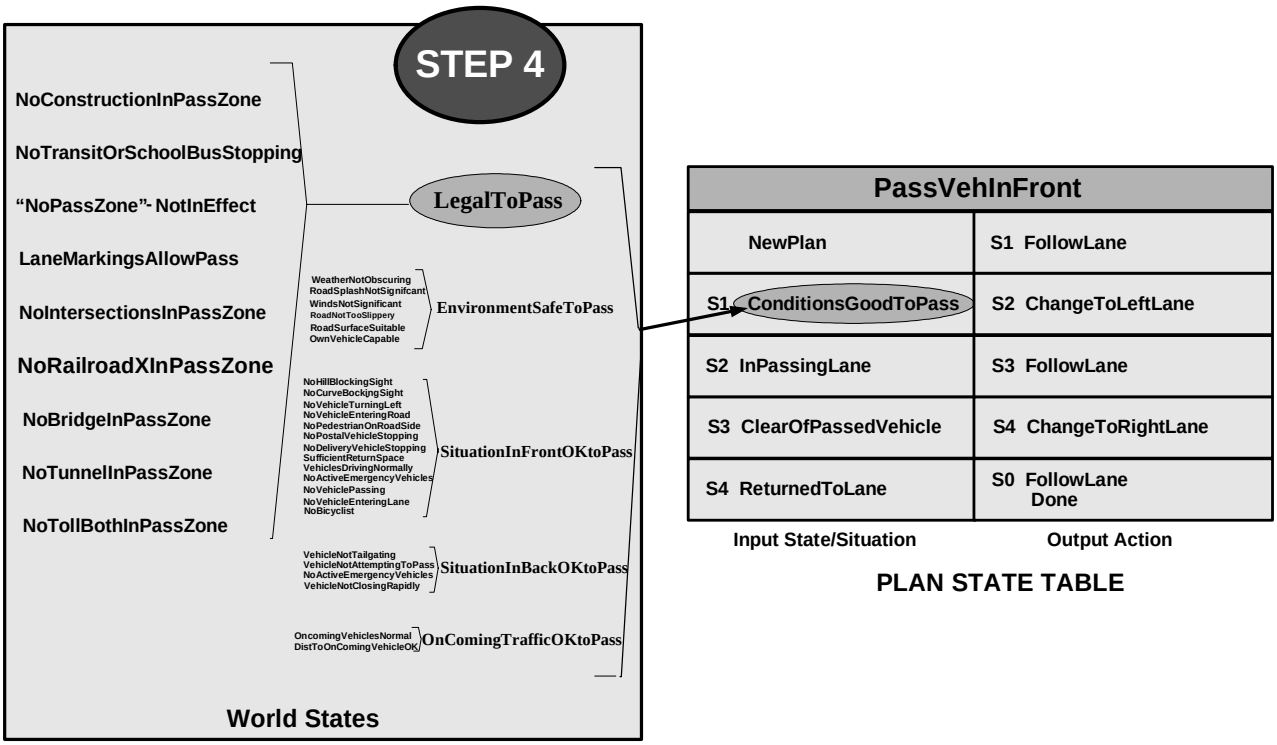

Figure 6. The identification of all of the antecedent world states used to evaluate whether the situation "ConditionsGoodToPass" is true or not.

The purpose of this step is to provide a listing of all of the dependent world states that affect whether the task branching condition/situation is true or not. We have not identified the sensitivity of this condition/situation to these world states or what functions are used to weight and evaluate their individual or combined truth. The identification of these world states is independent of whatever technique is used to evaluate their contribution to the condition/situation (such as "ConditionsGoodToPass"). Different implementation paradigms will determine this sensitivity, weighting, costing, and other evaluation functions.

2.1.5 Step 5 – World State Dependencies on Objects

This step identifies all of the objects, their features and attributes that need to be measured by the sensing system to create the world states described above. Figure 7 continues with the passing example. As described above, one of the aggregate world model states was "LegalToPass" which was a grouping of a number of related world states which all deal with various legal restrictions on the passing operation. One of these component world states that identify a legal restriction is "NoRailroadCrossingInPassZone". In this step, we wish to identify all of the objects, their features, and attributes that are relevant to creating each world state. For the world state named "NoRailroadCrossingInPassZone",

these relevant objects would include the railroad crossbuck emblem, crossing lights, crossing gate, crossing signs either alongside the road or painted on the road surface, the railroad tracks, and the train itself. We further define each object in terms of its characteristic features or attributes that will be used for recognition of the object (e.g. the width and length and height above ground of the crossbuck planks) and/or its state (e.g. flashing lights or a lowered gate as indicator of active warning state).

2.1.6 Step 6 – Measurement Resolutions

In this last step, we want to define the resolution requirements for the measurement of objects by the sensors. We do this by determining the expected distances to these objects during particular task activities. In the case of the task activity of passing a vehicle in front, we have to be able to see objects such as the railroad crossing crossbuck at the far limit of the expected passing zone. For a vehicle passing on a 75kph road, the passing zone could easily be 200 m or more. This means that the crossbuck, which is found at the railroad crossing itself, would have to be sensed and recognized by the sensory processing system at this distance of 200 m, figure 7. From this distance together with the knowledge of the size of the crossbuck plank elements, we can make an estimate of the absolute minimum sensory processing capability required to recognize it at this distance. If we assume a minimum of 3 pixel square to detect the crossbuck which works out to the image sensors having a minimum resolution capability of .09 degrees

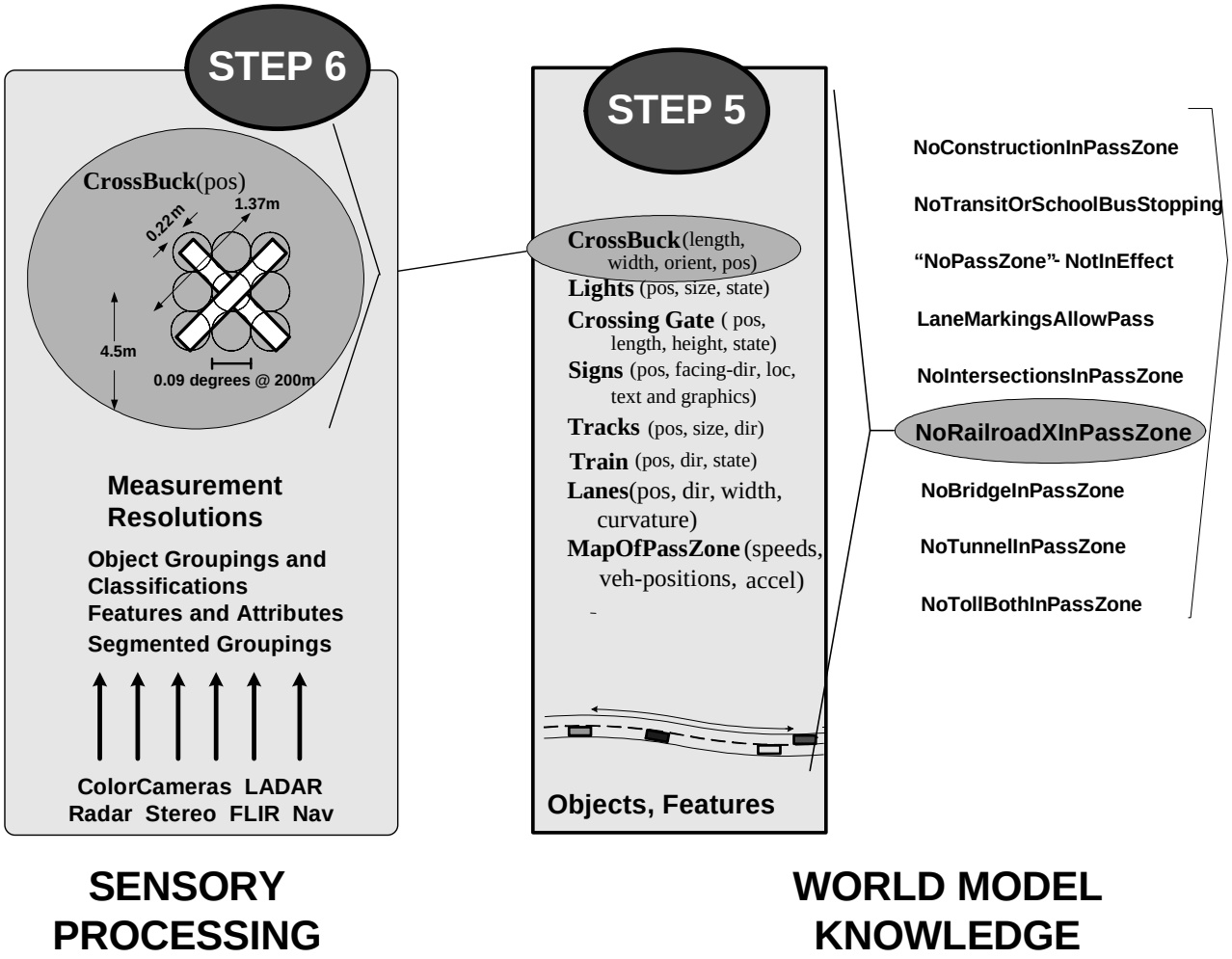

Figure 7. Example of the objects that are used to establish the "NoRailroadXinPassZone" world state and how the sensor measurement resolutions are determined.

3. SUMMARY AND CONCLUSIONS

This paper has presented a description of the use of the task-decomposition-oriented RCS methodology as an approach to acquiring and structuring the knowledge required for the implementation of the on-road driving tasks. This effort has produced a control architecture of six layers of agent control modules (figure 8) with a total of over 170 command/plans, each of which is described by a state table of Input Condition-Output Action rules. For the RouteSegment and DriveBehavior agent control modules, the commands can be further broken down by a plan selection based on the present state of the system. In figure 8, the commands are underlined and the plans that can be selected are listed under the appropriate underlined command. For example, at the DriveBehavior agent, 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 measurement resolutions as described in this paper. A large number of these plans have been detailed out in their respective state tables, identifying the named branching condition/situations required. A number of these situations have been examined to specify their antecedent world states and objects. As a result, an initial estimate has been made of the number of knowledge items involved for autonomous on-road driving and these are:

a) 1000 named condition/branching situations in the input condition side of the state tables.

b) 10,000 world states that are antecedents to the situation evaluations.

c) 1000 to 2000 objects to be detected in the world to arrive at the world states.

As can be seen, on-road autonomous driving is characterized by an extremely large knowledge set that would be impossible to deal with if not for some way to organize it. The task decomposition approach to this structuring has given us a single consistent process of well-defined sequential steps to both discover the relevant knowledge and to organize it. The knowledge has been partitioned into two large elements:

1) task procedural knowledge (i.e. how to do things) concerned with the description and representation of the task decomposition through layers of agent control modules performing control decision branching decisions, encoded as rules in state-tables. This is basically the set of procedures to follow for every situation that occurs in on-road driving; this is an explicit set of knowledge represented in rules.

2) world knowledge (i.e. what is the present situation) concerned with the description and representation of all of the states of the world and all of the objects to be sensed that are used to generate each of the condition branching situations in each state-table. This is basically the expert pattern recognition and reasoning knowledge that looks at the world and identifies/assesses the present situation so that a procedure from 1) can be followed. This is a more intuitive set of expert knowledge and harder to get at. It is represented in the situation dependencies on world states and objects in the environment.

The representational format for all of the knowledge has been driven by a requirement to identify everything according to its relevance to task activities. This has a very important impact on the implementation. This organization of the task knowledge is in a form that can be directly implemented. It has partitioned the knowledge by layers of resolution or abstraction so high level commands are clustered higher in the hierarchy and low level, equipment control knowledge is clustered at lower levels in the hierarchy. It has threaded access to all of the knowledge from the task through world model states to objects and their attributes to be measured. This is exactly the form the control system needs so it can access all of the information relevant to the present task activity as rapidly as possible.

This representational format leaves the knowledge in a form that continues to be easily accessible by non-programmer SMEs because all of the knowledge is indexed through the task behavior, which is in exactly the same format that the SMEs remembered it for example scenario discussions. This means that continual evolution and updating of the knowledge structure is possible by both the SMEs and the system designers, together or separately. New task commands, new branching condition situations and output commands, new world states, additional objects, features and attributes are all easily added and the appropriate place to add them is indexed by the task decomposition itself.

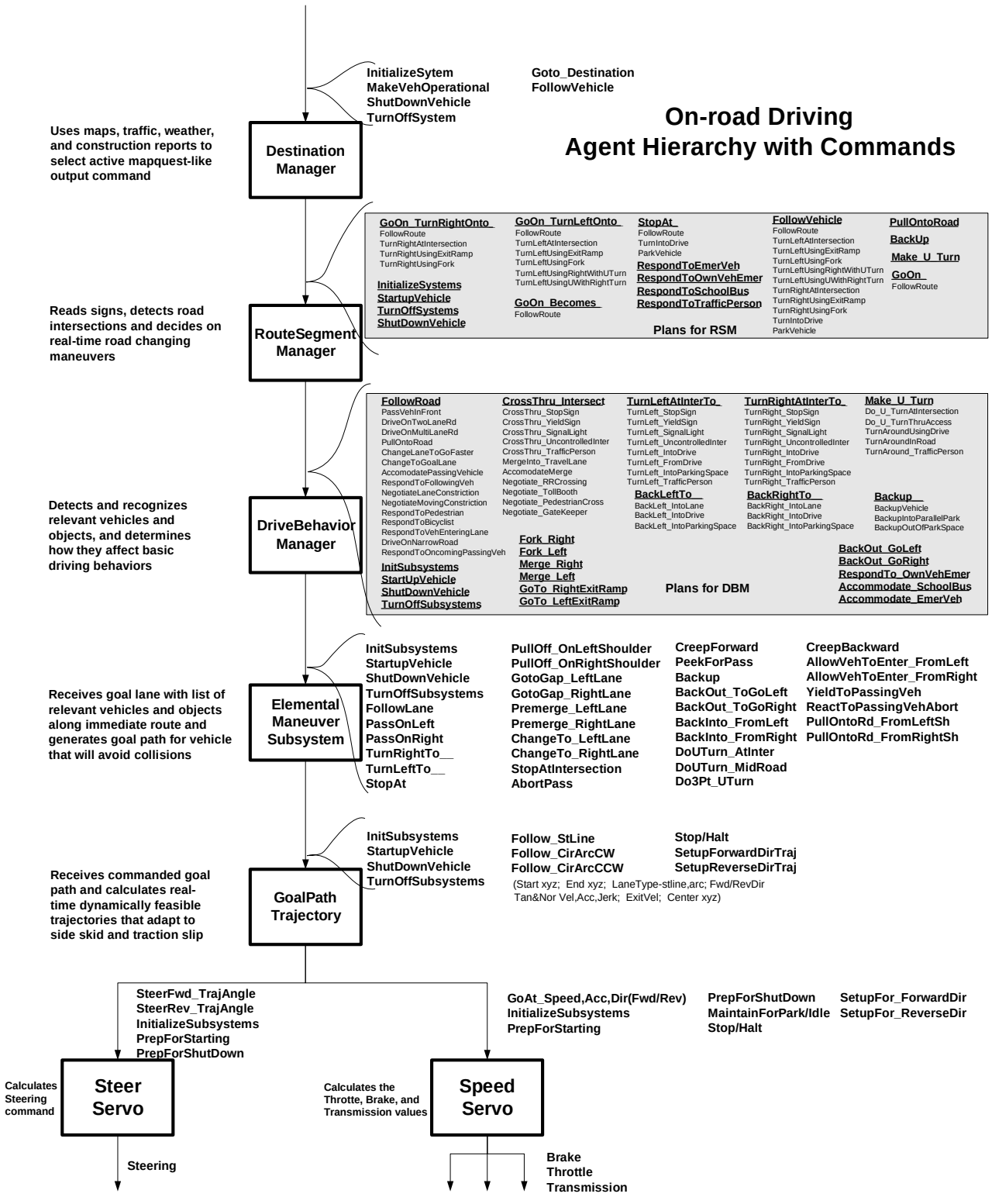

Figure 8. Illustration of the agent control architecture for on-road driving with the command/plan names listed and a brief description of the task responsibility provided at each level.

The definition of the knowledge base in this task-oriented format also supports a number of other processes, in particular, this knowledge base:

1) is the basis for developing procurement specifications and conformance tests for the procuring of complex autonomous systems,
2) provides the essential task detail to support writing operational requirements documents (ORDs) and operational and organizational (O&O) plan documents,
3) provides the consistent detailed requirements definitions necessary for system developers to build robust, reliable control systems,
4) by defining the required objects to be sensed and the sensor resolutions, identifies where the project goals are limited by the present state of sensor and sensory processing capabilities, and clearly identifies those research and development areas where significant impacts can be made in autonomous systems,
5) with its detailed level of specification of what has to be sensed, what world states have to be discerned, what situations recognized, and what response behaviors to perform, greatly aids in the creation of performance metrics [5] that can be used to accurately assess component capabilities at very fine levels rather than treating entire autonomous systems as black boxes.

NIST will continue with the development of this on-road driving knowledge base and will implement portions of it on an autonomous vehicle (an instrumented HMMWV) to provide checks on its usefulness. Additionally, NIST is involved in exploring mechanisms such as ontologies [6] to store this knowledge in a semantic model format in a computer-interpretable form. The hope is that ontology tools and techniques will provide a more consistent single representational solution through agent, process, class, property, and relationship definitions that will eventually be able to capture this knowledge and all of the implied relationships and then provide it back to support all of the above described activities.

Acknowledgements

This work was supported by DARPA's Mobile Autonomous Robotics Software (MARS) program (PM. D. Gage).

References

1. J. Albus, and A. Meystel, 2001, *Engineering of Mind*. New York,NY: John Wiley & Sons, Inc.
2. J. Albus, et.al. 2002, "4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems," NISTIR 6910, National Institute of Standards and Technology, Gaithersburg, MD.
3. E. Mettala, D. J. Cook and K. Harbison, Application of the Scenario-Based Engineering Process to the Unmanned Ground Vehicle Project, in *Reconnaissance, Surveillance, and Target Acquisition for the Unmanned Ground Vehicle* , O. Firschein and T. Strat (eds.), 1997.
4. J. McKnight, and B. Adams, *Driver Education Task Analysis. Volume 1. Task Descriptions*, Human Resource Research Organization, Department of Transportation, National Highway Safety Bureau 1970.
5. A. Barbera, J. Horst, C. Schlenoff, E. Wallace, D. Aha, 2003, Developing World Model Data Specifications as Metrics for Sensory Processing for On-Road Driving Tasks. In *Proceedings of the 2003 PerMIS Workshop.* Gaithersburg, MD.: NIST Special Publication 990.
6. C. Schlenoff, R. Washington, A. Barbera, 2004, Experiences in Developing An Intelligent Ground Vehicle (IGV) Ontology In Protégé, presented at the 2004 Protégé Conference in Washington, D.C.