4-D/RCS
A Reference Model Architecture
for Demo III

Version 0.1

Objective

To provide a reference model architecture for the design, engineering, integration, and test of intelligent supervised-autonomy controllers for unmanned ground vehicle systems.

Df: **Intelligent supervised-autonomy controllers**
are controllers capable of accepting mission level commands from human supervisors and executing those commands in unstructured and often hostile environments such as are found in urban warfare zones or rural battlefields; in mountain, wooded, farmland, or desert terrain, in all kinds of weather, day or night. An intelligent supervised-autonomy controller is capable of executing its assigned mission with, or without, direct communication from a human supervisor.

Background

The Robotics Technology Workshop for Ground Maneuver Warfare was held at the Institute for Defense Analyses (IDA) on May 15-16, 1996, to develop input for a Robotic Technology Research Plan for ground maneuver warfare. User needs and technology readiness for next generation unmanned ground vehicles were assessed, and mission scenarios and system requirements were addressed. Among the conclusions of that workshop were that:

1. A system architecture that defines the functionality of modules and interfaces between modules is required to define requirements for subsystem components, to integrate subsystems into operational systems, and to permit systems to be incrementally upgraded and reliably maintained.

2. Advanced software engineering methods and software development tools will be required to facilitate the design, development, and configuration of the enormous amount of software required for semi-autonomous vehicle systems.

3. Many of the technologies relevant to the unmanned ground vehicle program are being developed by sources outside the program. Therefore, the program should provide a system architecture and integration mechanisms that support incremental upgrades so that technological advances developed elsewhere can migrate easily into the program as soon as they become ready.

General description

An analysis of user requirements suggests that at least two classes of multi-mission vehicles may be needed:
1. a relatively large vehicle, possibly a modified HMMWV, for missions requiring high mobility, heavy payload, and self transportability.

2. a relatively small vehicle, weighing on the order of 1100 kg, capable of being carried in the back of a HMMWV, or being transported by helicopter, for missions requiring low observability and easy transportability.

Each vehicle should be equipped with a suite of sensors and a supervised-autonomy intelligent controller so as to be capable of successfully executing its assigned mission with, or without, direct communication from a human supervisor.

Operational Profile Description

Semi-autonomous vehicles should be capable of projecting force, protecting assets, providing information to battle visualization systems. Extensive use of semi-autonomous vehicle systems should enable commanders to shape the battle space so as to conduct decisive operations with minimum risk to human personnel of casualties or hostage taking.

Semi-autonomous vehicles should be capable of being deployed individually, or in groups, in concert with manned vehicles, or without close human interaction. The types of missions to be performed include:

Scout:
The vehicles should be capable of being sent on extended scout missions of from 2 km to 15 km with the ability to follow a planned route, avoid obstacles, maintain stealth, reach a designated destination, observe and report enemy activity, and return to base, with or without operator assistance. They should also be able to operate in close proximity to manned vehicles, for example to serve as lead vehicles on a patrol.

RSTA:
The vehicles should be capable of carrying RSTA packages and performing a full range of RSTA activities when in position, and limited RSTA capabilities while in motion. Pairs or teams of vehicles should be able to perform wide baseline observations, and serve as communications relays for each other.

Counter Mine:
The vehicles should be capable of carrying mine detection equipment while maneuvering to avoid and mark mines.

Smoke:
The vehicles should be capable of carrying smoke laying equipment and performing retrotraverse through smoke.

Decoy:
The vehicles should be able to carry inflatable or fiber glass shells with acoustic and thermal signature generating systems that simulate more valuable assets such as M1 tanks, and Bradley vehicles.

Military Operations in Urban Terrain:
The vehicles should be capable of operating on urban streets, using maps and GPS to navigate, and obstacle avoidance sensors and algorithms to deal with rubble, barricades, and concertino wire. They should be able to relay communications so as to coordinate with manned units operating nearby.

Security:
The vehicles should be able to patrol streets and base perimeters, and detect intruders and security breaches.

Weapons:
The vehicles should be capable of carrying a variety of weapons, both lethal and non-lethal.

Logistics:
The vehicles should be capable of providing logistics support in transporting ammunition or material between designated locations.

Mobility

Deployment:
By ground or air. The larger vehicles should be capable of being driven by a human driver, or convoyed autonomously at highway speeds of 65 km/h (40 mph) for distances of hundreds of kilometers. Smaller vehicles should be transportable on land by HMMWV and in the air by helicopters or V-22 aircraft. The smaller vehicles should also have autonomous convoy capability, with highway speeds up to 65 km/h, and be towable.

Range:
Both large and small vehicles should have an operating radius of 15 km under supervised autonomous control.

Speed:
Appropriate to terrain and vehicle dynamics to 33 km/h (20 mph) (day, dry), or 17 km/h (10 mph) (night or wet) cross country, and 65 km/h (40 mph) on roads. The vehicles should be able to autonomously sense both positive and negative obstacles and anticipate rough terrain conditions in time to slow down to a safe speed.

Terrain:
The vehicles must be able to keep up with the manned units to which they are attached. The larger vehicles should be able to climb a 55 ˚ slope, ford a stream 1 m (40 inches) deep and avoid man-made and natural obstacles. The smaller vehicles may have reduced mobility capabilities and may require being transported in larger vehicles.

Multi Vehicle:
The vehicles should be capable of operating either independently or in groups of up to six under one supervisor.

Precision:
The vehicles should be capable of following a planned path on a map to an accuracy of 1 m to 5 m over a distance of 1 km using only inertial reference, and to maintain similar accuracy over distances of 200 km using GPS.

Operating conditions:
The vehicles should be capable of day or night operation in limited adverse weather (rain/dust). Night operations are extremely important.

Duty cycle:
The vehicles should be capable of operating on station (not on the move) for at least 72 h without refueling.

Control

Modality:
The control system should have a variety of modes including direct human operation, teleoperation, and supervised autonomy.

Operator Interface:
The operator interface should provide a digital terrain map display with overlays indicating roads, buildings, and the vehicles position. Windows on the operator's screen should be available for target id. The operator should be able to designate way-points and determine formation control for multivehicle operation.

Communications Link:
Fiber optics or satellite communications will be available only in rare and unusual situations. Therefore, communications bandwidth will almost always be limited. Line of sight microwave or laser communication may provide 1 megabyte per second data rates. However, the vehicles will normally operate for most of the time under non-line of sight conditions where communications will be limited to 20 kilobytes per second or less. Even these data rates may be intermittent and unreliable.

Thus, the vehicles must have the capability to carry out their missions with low data rate communications, and operate for extended periods without human intervention.

Technology Drivers

There are a number of technological problems that must be addressed to support the user needs. These consist of the following:

Sensing:
A broad variety of sensors will be required for driving, navigation, and vehicle health and safety. Vision sensors, including CCD color cameras, FLIRs, night vision cameras, LADARs, radar, acoustic arrays may be used for driving. For navigation, GPS and inertial sensors such as gyros, accelerometers, and magnetic compass will be required. For vehicle health and safety, a variety of sensors such as odometers, speedometers, oil pressure, vibration, battery voltage, fuel level, steering angle, brake, throttle, gear shift, and engine rpm will be required. Mission operations may require many of the same type of sensors used for driving, plus sensors for chemical, magnetic, and other signatures.

Obstacle detection and avoidance:
In order to process images in real-time so as to detect and avoid obstacles and stay on the road or chosen path, it is imperative that the most effective image processing algorithms be used, and computational resources be used efficiently. This suggests that the sensor system should have both wide angle and narrow field of view cameras, and the image processing system be capable of attending to the most important regions in the visual field. This implies that camera pointing systems be capable of directing high resolution cameras to stabilize on and track targets of attention.

Image stabilization
Image stabilization will also be required for high-speed driving on rough terrain. In order to drive at high speeds, the vehicle must detect potential obstacles at long distances. This requires that cameras have high resolution. However, high resolution images are sensitive to blurring due to roll, pitch, and yaw motions caused by engine vibrations and high speed driving over rough terrain. Tracking of objects and features over multiple frames is

severely compromised by camera rotation. A combination of mechanical and electronic stabilization should be studied to achieve the most cost effective stabilization of images.

Depth images
Measurement of depth is absolutely critical to successful driving. There are three methods of measuring depth in visual images: stereo, image flow (or motion parallax), and laser range imaging cameras. Each of these methods has great promise, but significant technical difficulties remain.

Stereo is computationally intensive and, therefore, difficult to achieve with high resolution in real-time. Thus, it is difficult to reliably detect small obstacles, or negative obstacles (ditches or pot holes), at distances large enough for high-speed driving. However, stereo techniques are improving, and the use of multiple baseline stereo systems and pyramid processing techniques are promising.

Image flow can be used for computing clearance between the vehicle path and obstacles. It can also compute time to collision and may be useful for steering on roads with well-defined lane markings. However, image flow requires precisely stabilized camera images, and performance is degraded in the most important part of the image (i.e., that part close to the vector of motion).

Scanning laser range imagers (LADAR) produce relatively high resolution images and can reliably detect obstacles at significant distances. However, LADAR scanning processes are slow and typically cannot generate images faster than 1 to 4 images per second. One image per second is probably too slow to support high speed driving, and four per second is marginal. Scanning methods that could concentrate the scanned laser beam on objects of attention are possible and would provide considerable benefits for both resolution and speed.

Strobed laser range imagers can, in principle, produce high resolution images at high frame rates, but they are still in the experimental phase. Also, they emit large amounts of radiation that would compromise stealth on the battlefield, and may present an eye hazard to humans.

In the near term, the most promising approach seems to be to combine stereo or laser range imagers with stabilized CCD cameras or FLIR imagers. Predictive filtering methods may then achieve the necessary speeds.

In the mid- and long-term, the development of higher speed LADAR cameras, higher speed stereo techniques, and multiple baseline camera systems should eventually produce systems that can support driving at human levels of performance.

Navigation and landmark recognition:
In the near term, navigation via GPS will be the most cost effective and reliable. Differential GPS can measure location to better than 1 m over distances of kilometers. However, GPS is subject to jamming and requires that several GPS satellites be visible. It can be blocked by tree foliage and overpasses. Inertial navigation systems are expensive, but can maintain position within about 0.1% of distance traveled. A combination of GPS and inertial navigation is quite effective.

Landmark recognition is very situation dependent. In some terrain (such as flat desert), there may be no landmarks. In other situations, landmarks, such as trees and buildings, may be ambiguous. Road signs are typically used by humans, but reading road signs is difficult for current machine vision systems.

Cooperative target detection
The detection and tracking of targets from more than one observer often improve the accuracy of the results. The ability of two or more semi-autonomous vehicles to triangulate on a single target may have significant benefits.

Image understanding
While the semi-autonomous vehicle program may require some elements of image understanding, it is not clear that extensive efforts should be spent on this problem. At least for the foreseeable future, the image processing capabilities listed above should suffice for the missions envisioned.

Situation awareness
Display and visualization of terrain maps with overlays to indicate positions of friendly and enemy vehicles and positions can be accomplished with near-term technology. Methods to make this information available to the semi-autonomous vehicles and their human operators should be pursued as a high priority.

Information integration
Real-time updates in terrain map data using information from vehicle sensors is possible with near-term technology developed for the DARPA SIMNET (now known as MODSAF) program and managed by the Army Topographic Engineering Center.

Uncertainty assessment
The reliability of incoming data from sensors needs to be evaluated and data must be tagged with an assessment of uncertainty.

Map based planning (local and global)
Use of map information for route planning is relatively well developed. What is needed is technology for handling multiple resolution maps so that both local and global route planning can be accomplished simultaneously.

Dynamic world modeling
The semi-autonomous intelligent vehicle will require: (1) a model of the world that maintains a list of objects of attention, (2) a set of digital terrain maps at various resolutions that are maintained and dynamically updated at a rate that can support the requirements of planning and reactive control at many different levels simultaneously. This is a capability that is available in only a few limited laboratory experimental systems. This area needs significant attention.

Reactive behavior and replanning
Semi-autonomous vehicles and their human operators need to be able to react to changing situations and replan in real-time to adapt to dynamic situations. A few systems can perform this type of activity, but only for a limited range of cases. Work is needed here.

Reasoning and problem solving
Artificial intelligence systems that are able to perform reasoning and problem solving are mostly limited to situations that can be adequately described by a set of situation-action rules. The ability of computer programs to reason and solve problems amid the complexities and uncertainties of a real battlefield does not exist. This capability will require significant research and development before it becomes a practical reality.

Communications
Communications bandwidth on the battlefield is a precious and limited commodity. While the control range and number of vehicles that can be controlled by a single operator will increase, the bandwidth for communications is not likely to grow significantly. Non-line of sight (non-los) radio communications will be the predominant mode of operation.

Operator interface
Operator interfaces that enable semi-autonomous vehicles to be easily operated by soldiers in the field, and seamlessly integrated into the command and control structure of operational military units are required.

Mobility and Navigation

The types of mobility and navigation capabilities that can be expected for the future in the near term, mid term, and far term are as follows:

Near term 0-3 years
Vehicles will be able to map terrain geometry and stationary solid obstacles. Day and night operations will be possible. Sensors will include video (CCD, FLIR, LADAR, and multispectral), inertial, GPS, and vehicle health and safety. Low cost inertial and GPS systems will become available. Semi-autonomous control techniques with smooth transition between human and machine control will be developed. Real-time replanning of routes will be possible along with a modest capacity for fault recovery.

Mid term 3-7 years
Obstacle detection will include moving objects and non-solid obstacles such as concertino wire. All weather operations will become possible. Sensors may include radar and acoustics. Landmark recognition and navigation will be used. Resource use optimization may become important.

Far term > 7 years
Vehicles will possess capabilities for image understanding and situation analysis. Autonomous reasoning about scene motion will be included. Semantic scene description will become important. Full 3-D mapping of complex terrain will be available. Vehicles will be able to maneuver through extreme terrain.

Technology development strategy
It is expected that the technologies to support unmanned ground vehicle operations will mature rapidly. The most fundamental problems are image processing for real-time terrain mapping, off-road driving, and obstacle avoidance. In these areas, the underlying technologies are developing at a fast pace. While the general image understanding problem remains unsolved and will probably continue to pose difficulties well into the next century, the machine vision problem for semi-autonomous navigation and mobility is much more limited and amenable to practical solution. LADAR and stereo vision, combined with terrain mapping and processing technologies, are close to producing the real-time information needed for many unmanned vehicle mission scenarios.

The most difficult problems are detection and characterization of negative obstacles such as ravines and craters, and thin obstacles such as concertino wire. These problems are unique to the unmanned ground vehicle mission.

It should be noted that many of the technologies required for the semi-autonomous multi mission vehicle are currently being developed by and for other programs, in both military

and civilian domains. This suggests that the UGV technology development program will be able to leverage technologies being developed elsewhere, so that the limited resources of this program can be focused on those technologies that are not being developed elsewhere. For example, LADAR technology, FLIRs, night vision systems, camera stabilization, GPS, inertial navigation, terrain mapping, and image processing algorithms for highway driving, and obstacle are all being developed elsewhere.

This suggests that the unmanned ground vehicle program should develop a system architecture that can support the incremental integration of many different subsystems as new technologies developed elsewhere become ready for migration into the unmanned ground vehicle program.

Approach

4-D/RCS is a reference model architecture that integrates the NIST (National Institute of Standards and Technology) RCS Real-time Control System us 96],with the German (Universitat der Bundeswehr Munchen) VaMoRs 4-D approach to dynamic machine vision [Dickmanns 94]. It incorporates many aspects of the David Sarnoff Laboratory image processing library and VFE hardware, the Dornier LADAR camera and range image processing algorithms, and the Army Research Lab computer-aided mission planning system and terrain visualization technology. 4-D/RCS will provide the DOD Demo III project with an open system operational architecture that will facilitate the integration of a wide variety of subsystems. These include: (1) a foveal/peripheral CCD color camera pair, (2) a Laser Range Imager, (3) a MAPS inertial guidance package, (4) a GPS satellite navigation system, (5) the stereo image processing algorithms developed at the NASA Jet Propulsion Lab, and (6) a HMMWV telerobotic driving system with interfaces for a wide variety of mission packages. 4-D/RCS will provide a theoretical basis for designing, engineering, integrating, and testing intelligent controllers for unmanned ground vehicle systems.

4-D/RCS will comply with all relevant standards developed under the following programs: Battlefield Digitization, the Army Technical Architecture, the C4ISR Architecture Framework, the Joint Technical Architecture, the UGV Joint Program Office standard architecture JAUGS, and the TARDEC Vetronics architecture. The 4-D/RCS open system architecture will incorporate a number of commercial and military standards to facilitate the maintenance, debugging, and upgrading of subsystems and software.

4-D/RCS will build on top of these standards an intelligent control system architecture that incorporates concepts and techniques from artificial intelligence, control theory, operations research, game theory, pattern recognition, image understanding, automata theory, and cybernetics. The theory embodied in 4-D/RCS borrows heavily from cognitive psychology, semiotics, neuroscience, and artificial intelligence. 4-D/RCS closes a feedback control loop between sensing and acting through perception, world modeling, and planning.

Scope

The intended scope of the 4D/RCS architecture is the design, development, integration, and testing of intelligent supervised autonomy controllers for experimental military vehicle systems. 4-D/RCS provides mechanisms by which intelligent vehicle controllers can analyze the past, perceive the present, and plan for the future. It enables systems to assess cost, risk, and benefit of past events and future plans, and to make intelligent choices between alternative courses of action.

The 4-D/RCS model addresses the problem of intelligent control at three levels of abstraction: 1) a conceptual framework, 2) a reference model architecture, and 3) an engineering guideline.

1. A Conceptual Framework
At the highest level of abstraction, 4-D/RCS is intended to provide a conceptual framework for addressing the general problem of intelligent vehicle systems operating in man-made and natural environments to accomplish mission goals supervised by human commanders.

The 4-D/RCS conceptual framework spans the entire range of operations that affect intelligent vehicles, from those that take place over time periods of milliseconds and distances of millimeters to those that take place over time periods of months and distances of many kilometers. The 4-D/RCS model is intended to allow for the representation of activities that range from detailed dynamic analysis of a single actuator in a single vehicle subsystem to the combined activity of planning and control for hundreds of vehicles and human beings in full dimensional operations covering an entire theater of battle. The 4-D/RCS architecture is designed to integrate easily into the information intensive structure of Force XXI Operations and advanced concepts for the strategic Army and Marine Corps of the early 21st century.

In order to span the wide range of activities included within the conceptual framework, 4-D/RCS adopts a multilevel hierarchical architecture with different range and resolution in time and space at each level.

2. A Reference Model Architecture
At a lower level of abstraction, 4-D/RCS is intended to provide a reference model architecture to support the design and development of intelligent vehicle systems and to provide a theoretical basis for the development of future standards. To accomplish this, the 4-D/RCS architecture closely follows the existing command and control structure of the military hierarchy in assigning duties and responsibilities and in requiring knowledge, skills, and abilities.

4-D/RCS defines functional modules at each level such that each module embodies a set of responsibilities and priorities that are typical of operational units in a military organization. This enables the 4-D/RCS architecture to map directly onto the military command and control organization to which the intelligent vehicles are assigned. The result is a system architecture that is understandable and intuitive for the human commander and integrates easily into battle space visualization and simulation systems.

3. Engineering Guidelines
At a still lower level of abstraction, 4-D/RCS is intended to provide engineering guidelines for building and testing specific intelligent vehicle systems. In order to build a practical system in the near term, 4-D/RCS engineering guidelines will be developed bottom-up, starting with a single vehicle and its subsystems. The 4-D/RCS engineering guidelines define how intelligent vehicles should be configured to work together in groups with other intelligent vehicles, both manned and unmanned, in units of various sizes.

The type of problems to be addressed by the 4-D/RCS engineering guidelines include:
1) navigation and driving both on and off roads,
2) responding to human supervisor commands and requests,
3) accomplishing mission goals and priorities amid the uncertainties of the battlefield,
4) cooperating with friendly agents,
5) acting appropriately with respect to unfriendly agents, and
6) reacting quickly, effectively, and resourcefully to obstacles and unexpected events.

Intelligent vehicle systems will consist of a variety of sensors, actuators, navigation and driving systems, communications systems, mission package interfaces, and weapons systems controlled by an intelligent controller.

Sensors may include TV cameras, LADARs, FLIRs, radar, acoustic, chemical, radiation, and radio frequency receivers, as well as inertial, force, pressure, and temperature sensors.

Actuators may include motors, pistons, steering, brakes, throttle, lasers, r.f. transmitters, and pointing systems for cameras, antennae, and weapons.

Navigation and driving systems may include mission planning systems using digital terrain maps and computer aided route planning systems, GPS and inertial guidance systems, and machine vision systems for on and off road driving with obstacle avoidance and target tracking capabilities.

Communications may include microwave, packet radio, and lasers, and may include relay via other ground vehicles, air vehicles, or satellite communications. Updates for terrain data or over-the-hill visualization may be supplied by unmanned air vehicles.

Mission packages may include reconnaissance, surveillance, and target acquisition systems, laser designators, counter mine equipment, sniper detection equipment, smoke generators, electronic warfare equipment, logistics support, or communications relay antennae.

Weapons systems might include machine guns, recoilless rifles, cannons, missile launching systems, mortars, or a variety of non-lethal weapons.

The intelligent control system will consist of sensory processing, world modeling, value judgment, behavior generation, and operator interface components. These will be arranged in a hierarchical control architecture that is modeled after the command and control structure used throughout the military.

4-D/RCS as a Conceptual Framework

To facilitate a precise understanding of the 4-D/RCS architecture, from this point in the text, terms will be defined in the text near where they are first used.

Df: **framework**
a description of the functional elements, the representation of knowledge, and the flow of information within the system

The 4-D/RCS conceptual framework defines how intelligent unmanned vehicle systems can be integrated into the current command and control structure of the military. 4-D/RCS establishes a system architecture that assures that intelligent vehicle systems will always obey direct orders from human commanders. It also assures that decisions made autonomously by intelligent vehicles while out of contact with human supervisors will respect the goals, priorities, and rules of engagement set by human commanders.

4D/RCS is a hierarchical architecture. At every level within the control hierarchy, behavior generation modules receive goals and priorities from superiors and decompose those goals into subgoals and priorities for subordinates at levels below. At each level, plans are

formulated, decisions are made, and actions are taken that affect both peers and subordinates.

At every level, sensory processing modules filter and process information derived from observations by subordinates. Events are detected, objects recognized, situations analyzed, and status reported to superiors at the next higher level.

At every level, sensory processing and behavior generation modules have access to a model of the world that is resident in a knowledge database. This world model enables the intelligent system to analyze the past, plan for the future, and perceive sensory information in the context of expectations.

At every level, a set of cost functions enable value judgments and determine priorities that support intelligent decision making, planning, and situation analysis. This provides a robust form of value driven behavior that can function effectively in an environment filled with uncertainties and unwanted signals.

At the top level of any military hierarchy, there is a human commander supported by staff who provide intelligence and decision support functions. This is where high level strategy is defined and strategic goals are established. The top level commander decides what kind of operations will be conducted, what rules of engagement will be followed, and what values will determine priorities and shape tactical decisions.

Throughout the hierarchy, strategic goals are decomposed through a chain of command that consists of operational units made up of intelligent agents (humans or machines), each of which possesses a particular combination of knowledge, skills, and abilities, and each of which has a well-defined set of duties and responsibilities. Each operational unit accepts tasks from a higher level unit and issues sub tasks to subordinate units. Within each operational unit, intelligent agents are given job assignments and allocated resources with which to carry out their assignments. Within each operational unit, intelligent agents schedule their activities so as to achieve the goals of the jobs assigned to them. Each agent is expected to make local executive decisions to achieve goals on schedule by solving local problems and compensating for local unexpected events. Within a unit, each agent acts as a member of a team in planning and coordinating with peers at the same level, while simultaneously acting as the commander of a subordinate unit at the next lower level.

Each agent in each operational unit has knowledge of the world environment in which he, she, or it must function. This knowledge includes state variables, maps, images, and symbolic descriptions of the state of the world, and of objects and groups that exist in the environment, including their attributes and relationships, and processes which develop over time. Knowledge is kept current and accurate through sensors that detect events and sensory processing systems that compute attributes of objects and situations in the world. Knowledge of the world includes laws of nature that describe how the environment behaves under various conditions, as well as values and cost functions that can be used to evaluate the state of the world and the performance of the intelligent control system itself.

At the bottom of the hierarchy, physical actions take place and sensors measure phenomena in the environment.

For any chain of command, there is an organizational chart that describes functional groupings and defines who reports to whom. However, typical organizational charts do not show all the communication pathways by which information flows throughout the organization. Much information flows horizontally between agents and operational units, through both formal and informal channels. Multiple agents within an operational unit share

knowledge about objects and events in the world, and status of other agents. For example, units operating on the battlefield often can see each other and may respond to requests for help from peers without explicit orders from superiors. Also, plans developed in one operational unit may be communicated to other units for implementation.

4-D/RCS explicitly allows for the exchange of information between organizational units and agents at the same level, or even at different levels. Commands and status reports flow only between supervisor and subordinates, but queries, replies, requests, and broadcasting of information -- by posting in common memory or by messaging mechanisms -- may be used to convey information between any of the units or agents in the entire 4-D/RCS architecture.

The 4-D/RCS organizational chain of command is defined by the duties and responsibilities of the various modules and by the flow of commands and status reports between them, not by access to information or the ability to communicate. This means that while the relationships between supervisors and subordinates is that of a tree, the exchange of information between modules is a graph that, in principle, could be fully connected. In practice, however, the communication network is not typically fully connected because many of the modules simply have nothing to say to each other.

4-D/RCS as a reference model architecture

At a level of abstraction below the conceptual framework, 4-D/RCS is intended to provide a reference model architecture for the design and development of intelligent vehicle systems and software, and to provide the theoretical basis for future standards.

Df: **architecture**
the structure of components, their relationships, and principles of design; the assignment of functions to subsystems and the specification of the interfaces between subsystems

Df: **reference model architecture**
an architecture in which the entire collection of entities, relationships, and information units involved in interactions between and within subsystems are defined and modeled

Df: **intelligent system**
a system with the ability to act appropriately in an uncertain environment

Df: **appropriate action**
that which increases the probability of success

Df: **success**
the achievement of mission goals

Df: **mission goal**
a desired state that a mission is designed to achieve or maintain

The 4-D/RCS reference model architecture has the following properties:

1. It defines the functional elements, subsystems, interfaces, entities, relationships, and information units involved in intelligent vehicle systems.

2. It supports the selection of goals, the generation of plans, the decomposition of tasks, the scheduling of subtasks, and provides for feedback to be incorporated into control

processes so that both deliberative and reactive behaviors can be combined into a single integrated system.

3. It supports the processing of signals from sensors into knowledge of situations and relationships and the storage of knowledge in representational forms that can support reasoning, decision-making, and intelligent control.

4. It provides both static (long-term) and dynamic (short-term) means for representing the richness and abundance of knowledge necessary to describe the environment and state of a battlefield and the intelligent vehicle systems operating within it.

5. It supports the transformation of information from sensor signals into symbolic and iconic representations of objects, events, and situations, including semantic, pragmatic, and causal relationships. It also supports transformations from iconic (pictorial) to descriptive (symbolic) forms, and vice versa.

6. It supports the acquisition (or learning) of new information and the integration and consolidation of newly acquired knowledge into long-term memory.

7. It provides for the representation of values, the computation of costs and benefits, the assessment of uncertainty and risk, the evaluation of plans and behavioral results, and the optimization of control laws.

Df: **functional elements**
the functional elements are the fundamental computational units of a system

Axiom 1: *The functional elements of an intelligent system are behavior generation, sensory processing, world modeling, and value judgment.*

Df: **behavior generation**
behavior generation is the planning and control of action designed to achieve behavioral goals.

Df: **goal**
a desired state that a behavior is designed to achieve or maintain

Behavior generation accepts task commands with goals and priorities, formulates and/or selects plans, and controls action. Behavior generation develops or selects plans by using a priori task knowledge and value judgment functions combined with real-time information provided by world modeling to find the best assignment of tools and resources to agents, and to find the best schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). Behavior generation controls action by both feedforward action plans and by feedback error compensation. Behavior generation may also learn system models and optimize control laws.

Df: **control law**
a set of differential equations (for a continuous system) or difference equations (for a discrete system) that predict how the system output will respond to a given input.

A control law is typically of the form

$$\mathbf{y} = \mathbf{C}(\mathbf{x}) + \mathbf{D}(\mathbf{u})$$

where

y = the system output
u = the system input
x = the system state
C defines how the output depends on the state
D defines how the output depends on the input

Df: **sensory processing**
sensory processing is a process by which sensory data interacts with prior knowledge in order to detect or recognize useful information about the world

Sensory processing accepts input data from sensors that measure properties of the external world as well as conditions internal to the system itself. Sensory processing scales, windows, and filters data, computes observed features and attributes, and compares them with predictions from internal models. Correlations between sensed observations and internally generated expectations are used to detect events and recognize entities and situations. Differences between sensed observations and internally generated expectations are used to update internal models. Sensory processing also clusters, or groups, recognized entities and detected events into higher order entities and events, and computes attributes of entities and events.

Df: **value judgment**
value judgment is a process that:
a) computes cost, risk, and benefit of actions and plans,
b) estimates the importance and value of objects, events, and situations,
c) assesses the reliability of information,
d) calculates the rewarding or punishing effects of perceived states and events.

Value judgment evaluates perceived and planned situations thereby enabling behavior generation to select goals and set priorities. It computes what is important (for attention), and what is rewarding or punishing (for learning).

Df: **world modeling**
world modeling is a process that constructs and maintains a world model that can be used for feedback control as well as for the generation of predictions and the simulation of plans.

World modeling performs four principal functions:

1. It provides a best estimate of the state of the world to be used as feedback to servo behavior to desired plans, and to provide the latest status information on which to base planning operations.
2. It maintains a knowledge database about objects, agents, situations, and relationships, including knowledge of task skills and laws of nature, and maintains statistics over time in order to update generic models.
3. It predicts (possibly with several hypotheses) sensory observations based on knowledge in the knowledge database. Predicted signals can be used by sensory processing to configure filters, masks, windows, and templates for correlation, model matching, and recursive estimation.
4. It simulates results of possible future plans. Simulated results are evaluated by the value judgment system in order to select the best plan for execution.

Df: **world model**
an internal representation of the real world

The world model may include models of portions of the environment, as well as models of objects and agents, and a system model that includes the intelligent system itself. The world model is stored in a dynamic distributed knowledge database that is maintained by the world modeling process.

Df: **system model**
a set of differential equations (for a continuous system) or difference equations (for a discrete system) that predict how a system will respond to a given input

A system model is typically of the form

$$\dot{\mathbf{x}} = \mathbf{A}(\mathbf{x}) + \mathbf{B}(\mathbf{u})$$

where
x = the state of the system
$\dot{\mathbf{x}}$ = the rate of change in the system state
u = the input
A defines how the state evolves over time
B defines how the input effects the state

Learning may enable the world model to acquire system models.

Df: **knowledge database**
the data structures and the static and dynamic information that collectively form the intelligent system's world model

The knowledge database stores information about the world in the form of structural and dynamic models, state variables, symbolic entities, symbolic events, rules and equations, task knowledge, images, and maps.

The knowledge database has two parts:
a) A long term memory containing symbolic representations of all the generic and specific objects, events, and rules that are known to the intelligent system.
b) A short term memory containing both planned and realized states and attributes, as well as iconic, topological, and symbolic representations of those entities that are the subject of current attention.

Axiom 2: *The functional elements and knowledge database of an intelligent system can be distributed over a set of computational nodes, and be represented within each computational node by a set of modules interconnected by a communication system that transfers information between them.*

Df: **node**
a part of a control system that processes sensory information, maintains the generic and dynamic world model (including possibly several hypotheses) computes values, and plans and executes tasks

A typical 4-D/RCS node contains a Behavior Generation (BG) module, a World Modeling (WM) module, a Sensory Processing (SP) module, a Value Judgment (VJ) module, and a Knowledge Database (KD). All the modules in an 4-D/RCS node may have input and output connections to an Operator Interface.

Figure 1 illustrates the relationships in a single node of the 4-D/RCS architecture. The interconnections between sensory processing, world modeling, and behavior generation modules close a reactive feedback control loop between the observed input and the commanded action. The interconnections between behavior generation, world modeling, and value judgment modules enable deliberative planning and reasoning about future actions. The interconnections between sensory processing, world modeling, and value judgment modules enable knowledge acquisition, situation evaluation, and learning.

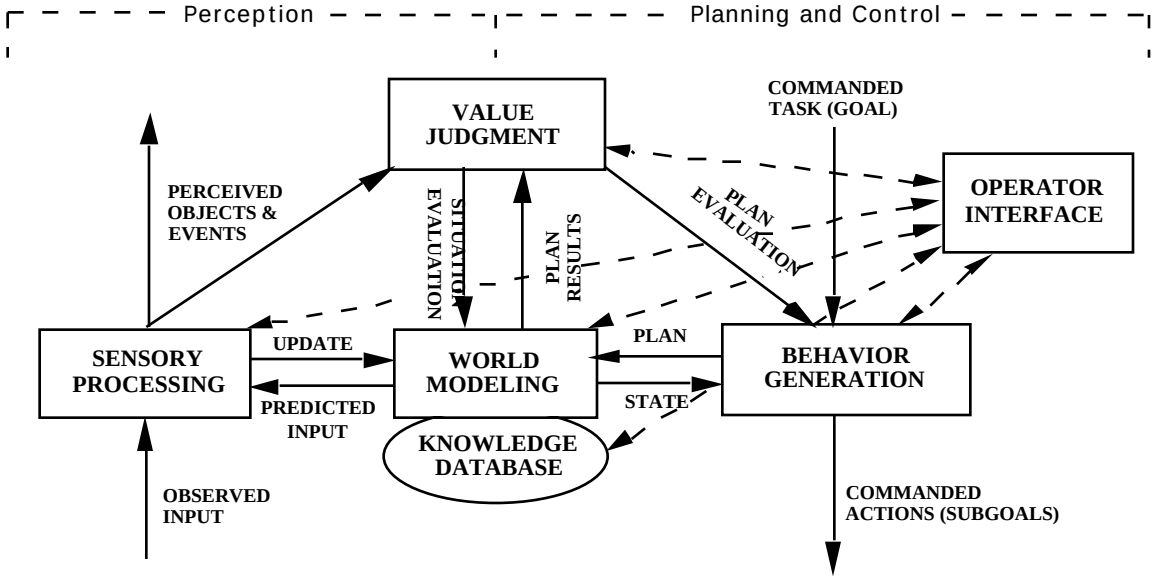


Figure 1. A node in the 4-D/RCS reference model architecture. The functional elements of an intelligent system are behavior generation (task desomposition and control), sensory processing (filtering, detection, recognition, grouping) world modeling (store and retrieve knowledge and predict future states), and value judgment (compute cost, benefit, importance, and uncertainty). These are supported by a knowledge database, and a communication system that interconnects the functional modules and the knowledge database. This collection of modules and their interconnections make up a generic node in the 4-D/RCS reference model architecture. Each module in the node may have an operator interface.

The connections to the Operator Interface enable a human supervisor to input commands, to override or modify system behavior, to perform various types of teleoperation, to switch control modes (e.g., automatic, teleoperation, single step, pause), and to observe the values of state variables, images, maps, and entity attributes. The Operator Interface can also be used for system maintenance, programming, and debugging.

The generic node illustrated in Figure 1 can be used to construct a distributed hierarchical reference model architecture. Depending on where the generic node resides in the distributed hierarchical structure of the 4-D/RCS architecture, it might serve in different capacities. These include an intelligent controller for an actuator, a subsystem, a vehicle, or as a squad, or platoon control unit, or a command center for a company, battalion, or higher level unit. A generic node, or a set of modules within a node, might be implemented

as an intelligent supervised-autonomy controller, or as a human person, or as a management unit at any level in the military command and control structure.

Axiom 3: *The complexity inherent in intelligent systems can be managed through hierarchical layering.*

Intelligent systems are inherently complex. Hierarchical layering is a common method for organizing complex systems that has been used in many different types of organizations throughout history for effectiveness and efficiency of command and control. In a hierarchical control system, higher level nodes have broader scope and longer time horizons with less concern for detail. Lower level nodes have narrower scope and shorter time horizons with more focus on detail. At no level does a node have to cope with both broad scope and high level of detail. This enables the design of systems of arbitrary complexity, without computational overload in any node and any level.

Hierarchical layering makes the 4-D/RCS architecture extensible and portable. Generic nodes can be designed and customized for use at many different levels by embedding different functional algorithms and knowledge. In the 4-D/RCS reference architecture, behavior generation modules in nodes at the upper levels in the hierarchy make long range strategic plans consisting of major milestones, while lower level behavior generation modules successively decompose the long range plans into short range tactical plans with detailed activity goals. Sensory processing modules at lower levels process data over local neighborhoods and short time intervals, while at higher levels, they integrate data over long time intervals and large spatial regions. At low levels, the knowledge database is short term and fine grained, while at higher levels it is broad in scope and generalized. At every level, feedback loops are closed to provide reactive behavior, with high-bandwidth fast-response loops at lower levels, and slower more deliberative reactions at higher levels.

At each level, state variables, entities, events, and maps are maintained to the resolution in space and time that is appropriate to that level. At each successively lower level in the hierarchy, as detail is geometrically increased, the range of computation is geometrically decreased. Also, as temporal resolution is increased, the span of interest decreases. This produces a ratio that remains relative constant throughout the hierarchy. As a result, at each level, behavior generation functions make plans of roughly the same number of steps. At higher levels, the space of planning options is larger and world modeling simulations are more complex, but there is more time available between replanning intervals for planning processes to search for an acceptable or optimal plan. Thus, hierarchical layering keeps the amount of computing resources needed for behavior generation in each node within manageable limits.

Also at each level, lower level entities are grouped into higher level entities. The effect is to geometrically increase the scope and encapsulate the detail of entities and events observed in the world. Thus, at each level, sensory processing functions compute entities that contain roughly the same number of sub entities.

Along the time line from the present ($t = 0$), short term memory is much more detailed than long-term memory, and plans for the immediate future are much more detailed than plans for the long term.

This kind of layering is typical of the military command and control hierarchy. At the top, strategic objectives are chosen and priorities defined that influence the selection of goals and the prioritization of tasks throughout the entire hierarchy. The details of execution are left to subordinates.

At intermediate levels, tasks with goals and priorities are received from the level above, and sub tasks with sub goals and attention priorities are output to the level below. In the intelligent vehicle environment, intermediate level tasks might be of the form <go to position at map coordinates x,y>, <advance in formation along line z>, <engage enemy units at time t>, etc. The details of execution are left to subordinates.

At each level in the 4-D/RCS hierarchy, higher level more global tasks are decomposed and focused into concurrent strings of more narrow and finer resolution tasks. The effect of each hierarchical level is thus to geometrically refine the detail of the task and limit the view of the world, so as to keep computational loads within limits that can be handled by individual intelligent agents, such as 4-D/RCS nodes or ordinary human beings.

Axiom 4: *The complexity of the real world environment can be managed through focusing attention.*

Intelligent systems must operate in a real world environment which is infinitely rich with detail. The real world environment contains a practically infinite variety of real objects, such as the ground, rocks, grass, sand, mud, trees, bushes, buildings, posts, ravines, rivers, roads, enemy and friendly positions, vehicles, weapons, and people. The environment also contains elements of nature, such as wind, rain, snow, sunlight, and darkness. All of these objects and elements have states and may cause, or be part of, events and situations. The environment also contains a practically infinite regression of detail, and the world itself extends indefinitely far in every direction.

Yet, the computational resources available to any intelligent system are finite. No matter how fast and powerful computers become, the amount of computational resources that can be embedded in any practical system will be limited. Therefore, it is imperative that the intelligent system focus the available computing resources on what is important, and ignore what is irrelevant.

Fortunately, at any point in time and space, most of the detail in the environment is irrelevant to the immediate task of the intelligent system. Therefore, the key to building practical intelligent systems lies in understanding how to focus the available computing resources on what is important and ignore what is irrelevant. The problem of distinguishing what is important from what is irrelevant must be addressed from two perspectives: top down and bottom up.

Top down, what is important is defined by behavioral goals. The intelligent system is driven by high-level goals and priorities to focus attention on objects specified by the task, using resources identified by task knowledge as necessary for successfully accomplishing given goals. Top down goals and high-level perceptions generate expectations of what objects and event might be encountered during the evolution of the task and which are important to achieving the goal.

Bottom up, what is important is the unexpected, unexplained, unusual, or out-of-limits. At each level, sensory processing functions detect errors between what is expected and what is observed. Error signals are processed at lower levels first. Control laws in lower level behavior generation modules generate corrective actions designed to correct the errors and bring the process back to the plan. However, if low level reactive control laws are incapable of correcting the differences between expectations and observations, errors filter up to higher levels where plans may be revised and goals restructured. The lower levels are thus the first to compute attributes of signals or images that indicate problems or emergency conditions, such as limits being exceeded on position, velocity, acceleration,

vibration, pressure, force, current, voltage, or temperature. The lower levels of control are also the first to act to correct, or compensate for errors.

In either case, hierarchical layering provides a mechanism for focusing the computational resources of the lower levels on particular regions of time and space. Higher level nodes with broad perspective and long planning horizon determine what is important, while the lower levels detect anomalies and attend to details of correcting errors and following plans. In each node at each level, computing resources are focused on issues relevant to the decisions that must be made within the scope of control and time horizon of that node.

Focusing of attention can be accomplished through masking, windowing, and filtering based on object and feature hypotheses and task goals. It can also be accomplished by pointing high resolution regions of sensors at objects-of-attention. For example in the human eye, the visual field is sampled at high resolution in the foveal region, and lower resolution in the periphery. Similarly, tactile sensors are closely spaced to produce high resolution in the finger tips, lips, and tongue with much lower resolution in other regions of the skin. The foveal area of the eyes and the high resolution tactile sensory regions of the fingers and lips are behaviorally positioned so as to apply the maximum number of sensors to objects of attention. High resolution sensors are scanned over the world to explore the regions of highest interest to the goals of the task.

At each level in the 4-D/RCS sensory processing hierarchy, attention is used to mask, filter, and window sensory data and to focus computational resources on objects and events that are important to the mission goal. This keeps the computational load of processing sensory data within manageable limits at all levels of the hierarchy.

The 4-D/RCS Hierarchy

An example of an 4-D/RCS reference model architecture for a single individual intelligent vehicle is shown in Figure 2. This diagram consists of a hierarchy of control nodes, each of which contain modules corresponding to the functional elements illustrated in Figure 1. Each node consists of a behavior generation (BG), world modeling (WM), and sensory processing (SP), and knowledge database (KD) module (not shown in Figure 2). Most nodes also contain a value judgment (VJ) module (hidden behind the WM module in Figure 2). Each of the nodes can therefore function as an intelligent controller. An operator interface may access modules in all nodes at all levels.

Figure 2 illustrates a vehicle system with four subsystems: locomotion, mission package, communication, and attention. Each of the four subsystems have one or more mechanisms, each of which have one or more actuators and sensors. For example, the locomotion subsystem may consist of a navigation and driving controller with several subordinate controllers for steering, braking, throttle, and gear shift, plus ignition, lights, horn, and turn signals, each of which has one or more actuators and sensors. The communication subsystem might consist of a message encoding subsystem, a protocol syntax generator, and communications bus interface, plus antenna pointing and band selection actuators. The attention subsystem might consist of mechanisms that use cameras, LADARs, FLIRs, radar, and acoustic sensors to detect and track objects, surfaces, edges and points, and compute trajectories for laser range finders, or pan, tilt, and focus actuators. The vehicle control system should be able to incorporate a variety of modular mission packages, each of which may contain a number of sensors and actuators. For example, a weapons mission package might have loading, aiming, and firing subsystems each with a number of sensors and actuators. A reconnaissance, surveillance, and target acquisition (RSTA) mission package might contain a variety of sensors and

sensor pointing actuators all of which need to be coordinated to successfully achieve behavioral goals.

The operator interface (OI) provides the capability for the operator to interact with the system at any time at a number of different levels -- to adjust parameters, to change speed, to select or verify targets, or to authorize the use of weapons. The OI provides a means to insert commands, change missions, halt the system, alter priorities, perform identification friend-or-foe (IFF), or monitor any of the system functions. The OI can send commands or requests to any BG module, or display information from any SP, WM, or VJ module. It can display any of the state variables in the KD at a rate and latency dictated by the communications bandwidth. Using the OI, a human operator can view situational maps with topographic features and both friendly and

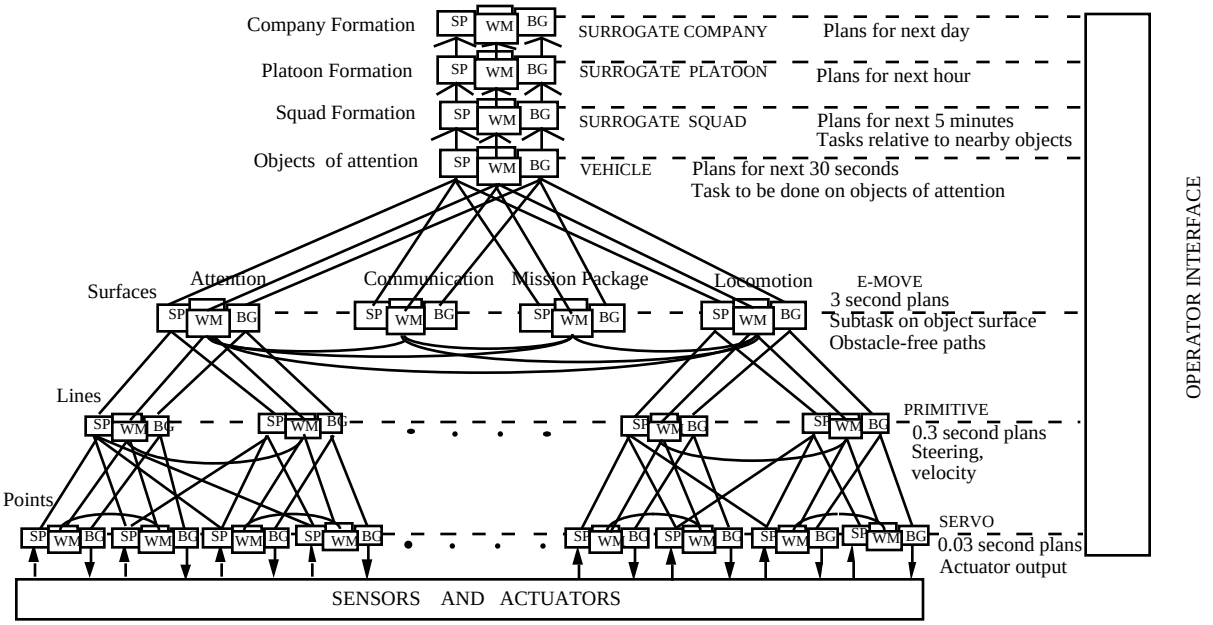


Figure 2. A 4-D/RCS reference model architecture for an individual vehicle. Processing nodes are organized such that the behavior generation (BG) modules form a command tree. Information in the knowledge database (KD) is shared between world modeling (WM) modules in nodes within the same subtree. KD modules are not shown in this figure. On the right, are examples of the functional characteristics of the behavior generation (BG) modules at each level. On the left, are examples of the type of entities recognized by the sensory processing (SP) modules and stored by the WM in the KD knowledge database at each level. Sensory data paths flowing up the hierarchy typically form a graph, not a tree. Value judgment (VJ) modules are hidden behind WM modules. A control loop may be closed at every node. An operator interface may provide input to, and output from, modules in every node.

enemy forces indicated with overlays. The operator may use the OI to generate graphics images of motion paths, or display control programs (plans) in advance, or while they are being executed. The OI may also provide a mechanism to run diagnostic programs in the case of system malfunctions.

In Figure 2, three levels of control are shown above the node representing the individual vehicle. These three additional levels represent a surrogate chain of command that, in principle, exists above the individual vehicle. However, because each vehicle is semi-autonomous, it carries a copy of the control nodes that otherwise would exist in its superiors if those superiors were tightly coupled in an integrated control structure. Because

the individual vehicles are physically separated, and may be only occasionally in contact with their superiors through a low bandwidth and often unreliable communication channel, it is necessary for each vehicle to carry a surrogate chain of command that performs the functions of its superiors in the command chain.

The surrogate chain of command serves four functions. First, it provides each vehicle with an estimate of what its superiors would command it to do if they were in direct communication. Second, it enables, any vehicle to assume the duties of any of its superiors in the event this should become necessary. Third, it provides a natural interface for human commanders at the squad, platoon, or company level to interface with the vehicle at a level relevant to the task being addressed. Fourth, it enables each vehicle to dedicate a separate node to handle each layer of higher level tasks. In this example, the surrogate chain of command consists of three levels with three different planning horizons (five minutes, one hour, and one day). These three levels deal with external objects and maps at three different scales and ranges. There, of course, may be more than three levels above the vehicle.

In Figure 2, the nodes at the upper levels may have even greater degree of branching than exists at the lower levels, but this is not shown in order to simplify the diagram. The horizontal curved lines between WM modules represent the sharing of state information in the knowledge database between nodes within sub trees in order to synchronize related tasks.

The functionality of each level in the 4-D/RCS reference model hierarchy is defined by the functionality, characteristic timing, bandwidth, and algorithms chosen for decomposing tasks and goals at each level. Typically these are design choices that depend on the dynamics of the processes being controlled. The numbers shown in Figure 2 represent planning horizons appropriate for a vehicle. For other types of systems, different numerical values would be derived from design parameters.

A 4-D/RCS Example

To illustrate the types of issues that will be addressed by the 4-D/RCS reference model architecture, an example of a seven level 4-D/RCS hierarchy for an armored company is given below. The specific numbers and functions given in this example are illustrative only. They are meant only to illustrate how the generic structure and function of the 4-D/RCS reference model architecture might be instantiated. Quantitative instances of an actual system design will be specified later in the section on implementation.

4-D/RCS Level 7 -- Company

A company is a unit that consists of a group of platoons. A 4-D/RCS node at level 7 corresponds to a company command headquarters, with a company commander and a group of platoon leaders, plus support staff. (Any or all of these could be humans or intelligent agent software modules.)

The company command headquarters unit plans activities and allocates resources for the platoons in the company. Incoming orders to the company are decomposed into assignments for platoons, resources and assets are allocated to each platoon, and a schedule is generated for each platoon to maneuver through a sequence of positions. Together, these assignments, allocations, and schedules comprise a plan. The plan may be devised by the company commander alone, or in consultation with his platoon leaders. The company level planning process may consider the exposure of each platoon's movements to enemy observation, and the traversability of roads and cross-country routes. The company commander typically defines the rules of engagement for the platoon and works with his

platoon leaders to develop a schedule that meets the objectives of the commanded task given to the company. In the 4-D/RCS company node, plans are computed for a period of about a day and recomputed at least once per hour, or more often if necessary. Desired positions of the platoons at about one hour intervals are computed.

The surrogate company node in a single 4-D/RCS vehicle performs the functions of the company command level when the vehicle is not in direct communication with the company headquarters. The node plans activities for the vehicle on a company level time scale and estimates what platoon level maneuvers should be executed to follow that plan. The surrogate company node considers the exposure of vehicle movements to enemy observations, and the traversability of roads and cross-country routes.

At the company level, the 4-D/RCS world model maintains a knowledge database containing a copy of the company level knowledge database that is relevant to that vehicle. It contains names and attributes of friendly and enemy forces and of the force levels required to engage them. Maps have a range of 1000 km (i.e. somewhat more than the distance that a company is likely to travel in a 24 hour day) with a resolution of about 300 m. Maps describe the terrain and location of friendly and enemy forces (to the extent that they are known), and roads and obstacles such as mountains, rivers, and woods. 4-D/RCS sensory processing at the company level computes information about the movement of forces, the level of supplies, and the operational status of all the platoons in the company, plus intelligence about enemy units in the area of concern to the company. This information is used to update maps and lists in the knowledge database so as to keep it accurate and current.

The surrogate company node also contains the value judgment functions for computing the cost and benefit of various tactical options such as calculating the risk of casualties. To the extent that the information in the surrogate knowledge database is identical with that in the real company knowledge database, the surrogate company node will make the same decisions as the real company node.

An operator interface allows human operators (either on-site or remotely) to visualize the deployment and movement of forces, the availability of ammunition and the overall situation within the scope of attention of the company commander. The operator can intervene to change priorities, alter tactics, or redirect the allocation of resources. Platoon leaders are expected to issue commands to their respective platoons, monitor how well their platoons are following the company plan, and make adjustments as necessary to keep on plan. Output from the company level through the platoon leaders comprise input commands to the platoon level. Output commands may be issued at any time, but typically are planned to change only about once every hour.

*****Note *****
It is important to clearly distinguish between the organizational units in the 4-D/RCS architecture and the intelligent agents that belong to those units. Each organizational unit consists of a commander agent (that is also a subordinate in a higher level unit) and subordinate agents (that are also commanders of lower level units). For example, a company unit consists of a company commander who is also a member of the battalion level staff and a group of platoon leaders who are also commanders of platoon level units. Conversely, each agent belongs to two organizational units. Each agent is a subordinate in a unit at one level and a commander of a unit at the next lower level.
****

Level 6—Platoon

A platoon is a unit that consists of a group of squads. A 4-D/RCS node at the Platoon level corresponds to a platoon commander plus his/her squad leaders. (Again, any of these could be humans or intelligent agent software modules, in any combination.) The platoon commander and squad leaders plan activities and allocate resources for the squads in the platoon. Orders are decomposed into job assignment for each squad, resources are allocated, and a schedule of activities is generated for each squad. Movements are planned relative to major terrain features and other vehicles within the platoon. Inter-squad formations are selected on the basis of tactical goals, stealth requirements, and other priorities. At the platoon level, plans are computed for a period of about an hour into the future, and replanning is done about every five minutes, or more often if necessary. Squad waypoints about five minutes apart are computed.

The surrogate platoon node in each vehicle performs the functions of the platoon command unit when the vehicle is not in direct communication with the platoon commander. It plans activities for the vehicle on a platoon level time scale and estimates what vehicle level maneuvers should be executed in order to follow that plan. Movements are planned relative to major terrain features and other vehicles within the platoon.

At the platoon level, the 4-D/RCS world model symbolic database contains names and attributes of targets, and the weapons and ammunition necessary to attack them. Maps with a range of about 100 km (i.e. more than the distance a platoon is likely to travel in one hour) and resolution of about 30 meters describe the location of objectives, and routing between them. Sensory processing integrates intelligence about the location and status of friendly and enemy forces. Value judgment evaluates tactical options for achieving squad objectives. An operator interface allows human operators to visualize the status of operations and the movement of vehicles within the squad formation. Operators can intervene to change priorities and reorder the plan of operations. Squad leaders are expected to sequence commands to their respective squads, monitor how well their squads are following the platoon plan, and make adjustments as necessary to keep on plan. The output from the platoon level through the squad leaders are commands issued to squads to perform maneuvers and engage enemy units in particular sectors of the battlefield. Output commands may be issued at any time, but typically are planned to change only about once every five minutes.

Level 5—Squad

A squad is a unit that consists of a group of individual vehicles, such as tanks, Bradley vehicles, or HMMWVs. An 4-D/RCS node at the Squad level corresponds to a squad leader plus his vehicle commanders (humans or intelligent software). This command team assigns duties to vehicles and schedules the activities of each vehicle within a squad. Orders are decomposed into assignments for each vehicle, and a schedule is developed for vehicles to maneuver in formation relative to enemy forces and large obstacles. Plans are developed to engage enemy units, or to conduct coordinated reconnaissance, surveillance, or target acquisition functions. At the squad level, plans are computed for about five minute into the future, and replanning is done about every 30 s, or more often if necessary. Vehicle waypoints about 30 s apart are computed.

The surrogate squad node in a single 4-D/RCS vehicle performs the functions of the squad command level when the vehicle is not in direct communication with the squad commander. The node plans activities for the vehicle on a squad level time scale and estimates what vehicle level maneuvers should be executed in order to follow that plan.

At the squad level, the 4-D/RCS world model symbolic database contains names, coordinates, and other attributes of other vehicles within the squad, other squads, and potential enemy targets. Maps with a range of about 10 km and a resolution of about 3 meters (if available) describe the location of vehicles, targets, landmarks, and local terrain features such as buildings, roads, woods, fields, fences, ponds, etc. Sensory processing determines the position of landmarks and terrain features, and tracks the motion of groups of vehicles and targets. Value judgment evaluates plans and computes cost, risk, and payoff of various alternatives. An operator interface allows human operators to visualize the status of the battlefield within the scope of the squad, or to intervene to change priorities and reorder the sequence of operations or selection of targets. Vehicle commanders issue commands to their respective vehicles, monitor how well plans are being followed, and make adjustments as necessary to keep on plan. Output commands to individual vehicles to engage targets or maneuver relative to landmarks or other vehicles may be issued at any time, but typically are planned to change only about once every 30 s.

Level 4—Individual vehicle

The vehicle is a unit that consists of a group of subsystems, such as locomotion, attention, communication, and mission package. Thus, an 4-D/RCS node at the vehicle level corresponds to a vehicle commander plus subsystem controllers. The vehicle commander assigns[1] jobs to subsystems and (possibly in collaboration with subsystem controllers) schedules the activities of all the subsystems within the vehicle. A schedule of waypoints is developed by the locomotion subsystem to avoid obstacles, maintain position relative to nearby vehicles, and achieve desired vehicle heading and speed along the desired path on roads or cross-country. A schedule of tracking activities is generated for the attention subsystem to track obstacles, other vehicles, and targets. A schedule of activities is generated for the mission package and the communication subsystems. Waypoints about 3 s apart out to a planning horizon of 30 s are planned every 3 s, or more often if necessary.

At the vehicle level, the world model symbolic database contains names (identifiers) and attributes of objects -- for example, the size, shape, and material characteristics of road surface, ground cover, or objects such as rocks, trees, bushes, mud, and water. Maps are generated from on-board sensors with a range of about 1 km and resolution of 30 centimeters. Maps represent object positions (relative to the vehicle) and dimensions of road surfaces, buildings, trees, craters, and ditches. Sensory processing measures object dimensions and distances, and computes relative motion. Value judgment evaluates trajectory planning and sensor dwell time sequences. An operator interface allows a human operator to visualize the status of operations of the vehicle, and to intervene to change priorities or steer the vehicle through difficult situations. Subsystem controller executors sequence commands to subsystems, monitor how well plans are being followed and modify parameters as necessary to keep on plan. Output commands to subsystems may be issued at any time, but typically are planned to change only about once every 3 s.

Level 3—Subsystem level

Each subsystem is a unit consisting of a group of related primitive systems, such as steering and braking, engine and transmission, sensor stabilization and pointing, message encoding and decoding, and weapons loading and aiming. An 4-D/RCS node at the Subsystem Level assigns jobs to each of its primitive systems and coordinates the activities of all the primitive systems within each subsystem. A schedule of steering and braking commands is developed to avoid obstacles. A schedule of power train adjustments is

[1] At the vehicle level and below, the assignment of jobs and resources to agents may be fixed by the system design.

developed for the engine and transmission to maintain desired vehicle speed. A schedule of pointing commands is generated for aiming cameras and sensors. A schedule of messages is generated for communications, and a schedule of actions is developed for loading and aiming weapons. For each primitive system, a plan consisting of trajectory of way points about 300 ms apart is generated out to a planning horizon of about 3 s in the future. A new plan is generated about every 300 ms.

At the Subsystem level, the world model symbolic database contains names and attributes of environmental features such as road edges, holes, obstacles, ditches, and targets. Maps generated from sensor data with a range of about 30 meters and a resolution of 30 centimeters are maintained of the shape and location of targets and obstacle boundaries. Sensory processing measures position, range, motion, and dimensions of object features, and computes surface properties such as surface area, orientation, texture, and motion. Value judgment supports planning of steering and aiming computations, and evaluates sensor data quality. An operator interface allows a human operator to visualize the state of the vehicle, or to intervene to change mode or interrupt the sequence of operations. Subsystem executors sequence commands to primitive systems, monitor how well plans are being followed, and modify parameters as necessary to keep on plan. Output commands to primitive systems may be issued at any time, but typically are planned to change about once every 300 ms.

Level 2— Primitive level

Each node at the primitive level is a unit consisting of a group of controllers that plan and execute velocities and accelerations to optimize dynamic performance of components such as steering, braking, acceleration, gear shift, camera pointing, and weapon pointing, taking into consideration dynamical interaction between mass, stiffness, force, and time. Communication messages are encoded into words and strings of symbols. Velocity and acceleration set points are planned every 30 ms out to a planning horizon of 300 ms.

The world model symbolic database contains names and attributes of state variables and features such as target trajectories and edges of objects. Maps consist of camera images with overlays of trajectories and linear image features. (See definition of map on page 53.) Sensory processing computes linear image features such as occluding edges, boundaries, and vertices and detects strings of events. Value judgment cost functions support dynamic trajectory optimization. An operator interface allows a human operator to visualize the state of each controller, and to intervene to change mode or override velocities. Primitive level executors keep track of how well plans are being followed, and modify parameters as necessary to keep within tolerance. Output commands are issued to the Servo level to adjust velocity and acceleration set points for vehicle steering or for pointing sensors or weapons platforms. Output commands are issued every 30 ms.

Level 1—Servo level

Each node at the servo level is a unit consisting of a group of controllers that plan and execute actuator motions and forces, and generate discrete outputs. Communication message bit streams are produced. The servo level transforms commands from component to actuator coordinates and computes motion or torque commands for each actuator. Desired forces, velocities, and discrete outputs are planned for 3 millisecond intervals out to a planning horizon of 30 ms.

The world model symbolic database contains values of state variables such as actuator positions, velocities, and forces, pressure sensor readings, position of switches, and gear shift settings. Maps consist of camera images with overlays of point features and displays of sensor readings. Sensory processing computes point features in images such as spatial and temporal gradients, stereo disparity, range, color, and image flow. Sensory

processing also detects events and scales and filters data from individual sensors that measure position, velocity, force, torque, and pressure. An operator interface allows a human operator to visualize the state of the machine, or to intervene to change mode, set switches, or jog individual actuators. Executors servo actuators and motors to follow planned trajectories. Position, velocity, or force servoing may be implemented, and in various combinations. Motion output commands to power amplifiers specify desired actuator torque or power every 3 ms. Discrete output commands produce switch closures and activate relays and solenoids.

This example illustrates how the 4-D/RCS multilevel hierarchical architecture assigns different responsibilities and duties to various levels of the hierarchy with different range and resolution in time and space at each level. At each level, sensory data is processed, entities are recognized, world model representations are maintained, and tasks are deliberately decomposed into parallel and sequential subtasks, to be performed by cooperating sets of agents. At each level, feedback from sensors reactively closes a control loop allowing each agent to respond and react to unexpected events.

At each level, there is a characteristic range and resolution in space and time, a characteristic bandwidth and response time, and a characteristic planning horizon and level of detail in plans. The 4-D/RCS architecture thus organizes the planning of behavior, the control of action, and the focusing of computational resources such that functional modules at each level have a limited amount of responsibility and manageable level of complexity.

The Computational Modules: Behavior Generation

Df: **Behavior Generation (BG) module**
a functional module for decomposing tasks into jobs for a set of agents within the module, for generating coordinated plans for those agents, and for executing those plans so as to produce sub tasks for subordinate BG modules.

A BG module corresponds to an operational unit that plans, coordinates and executes behavior designed to accomplish tasks. The set of intelligent agents in the BG module may correspond to a team of persons within an organizational unit or a set of coordinated processes in an intelligent control system. For any BG module, there exists a set of skills, or a collection of behaviors that the BG module may evoke to accomplish task goals. The knowledge required to perform this collection of behaviors is stored in a set of task frames in the KD. If the BG module is given a command to do a task that it is capable of performing, it uses the knowledge in the corresponding task frame to accomplish the task. If it is given a command to do a task that it is incapable of performing, (i.e. for which there is no task knowledge stored in a task frame) it responds to its supervisor with an error message = <can't do>.

A BG module contains four types of sub modules:

1. A Job Assignor (JA)
2. A set of plan action SChedulers (SC)
3. A Plan Selector (PS)
4. A set of control EXecutors (EX)

Within the BG module, the JA, SC, PS, and EX sub modules are arranged in the configuration shown in Figure 3. The SP, WM, and VJ modules that provide the BG sub modules with the information necessary for decision making and control are not shown in Figure 3.

Df: **Job Assignor (JA)**
a sub module of BG that decomposes input tasks into job assignments for each agent within the BG module

The Job Assignor (JA) performs four functions:
1. JA accepts input task commands from a higher level BG module, including at least two steps in the current higher level plan.
2. JA decomposes commanded input tasks into a set of jobs which are assigned to agents within the BG module[2].
3. JA allocates resources to the agents to enable them to accomplish their assigned jobs.
4. JA transforms each job assignment into a coordinate frame of reference which is appropriate for the performing agent.

The JA sub module functions as the supervisor of the agents within the BG operational unit.

Df: **SCheduler (SC)**
a submodule of BG that accepts a job assignment and computes a schedule for the agent of which it is a part

[2] At the vehicle level and below, the assignment of jobs and resources to agents may be fixed by the system design.

There is a SC submodule for each agent within the BG module. Each SC accepts its assignment from the JA and has access to the current plan in the higher level BG module.

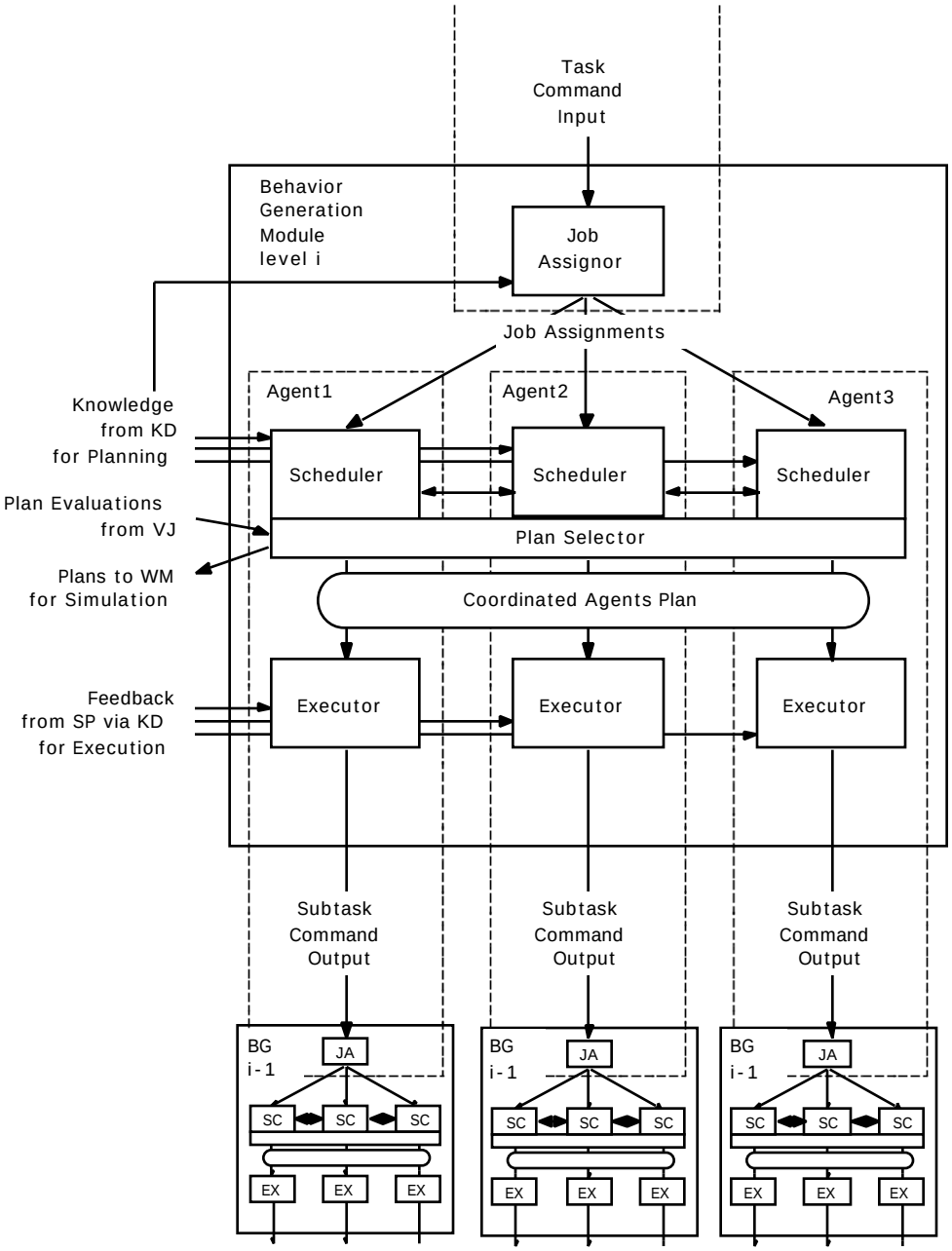


Figure 3. Internal structure of Behavior Generation (BG) modules at two levels with three agents at each level. A BG module is an organizational unit consisting of a Job Assignor, a set of SChedulers, a Plan Selector, a plan buffer that contains the coordinated agents plan, and a set of EXecutors. The SChedulers, EXecutors, and Job Assignors comprise agents within the BG units. Each agent is part of two BG units at two different levels. At level i, an agent is a team member, or peer. At level i-1, the same agent is a supervisor.

Each SC computes a sequence of activities for its agent that accomplishes the assigned job. The set of SC submodules within the BG module typically need to communicate with each other to coordinate the sequences of activities between the agents in the BG module or with agents in other BG modules. The SC modules may sometimes need to negotiate with the JA module or with its peer SC modules to reallocate resources among the agents to resolve conflicts for shared resources and to optimize the overall coordination of actions between all the agents in the BG module.

Within the 4-D/RCS architecture, a system designer can implement different management styles by different distribution of duties and responsibilities to the JA and SC modules within a BG module. For example, an autocratic, or micro-management style can be achieved by giving both job assignment and most scheduling responsibilities to the JA submodule of the supervisor agent, leaving very little discretion to the SC submodules of the subordinate agents. On the other hand, a democratic management style can be achieved by giving the JA submodule of the supervisor agent only general oversight responsibilities and allowing the subordinate agents to negotiate among themselves for job assignments and resources. This gives responsibility to the subordinates for scheduling their own activities.

In either case, the JA and SC submodules together have responsibility for producing a tentative plan that uses the resources of the set of agents within the BG module to accomplish the commanded task. The resultant tentative plan is submitted to the Plan Selector for simulation and evaluation.

Df: **Plan Selector (PS)**
a submodule of BG that works with a WM plan simulator and VJ plan evaluator to select the best overall plan (i.e. job assignment, resource allocation, and coordinated schedule) for all the agents in the BG module

The Plan Selector places the first tentative plan in a plan buffer. If a new tentative plan is evaluated as better than the plan in the buffer, it will replace the plan in the buffer. Many tentative plans may be generated before a "best" plan is selected. However, as soon as the first tentative plan is generated, there exists at least an acceptable plan in the plan buffer. Thus, the EXecutors almost always have a plan that can be executed, even though it might not be the "best" plan.

Df: **EXecutor (EX)**
a sub module of BG that executes its portion of the selected plan, coordinating actions (when required), and correcting for errors between the plan and the evolution of the world state reported by the world model

There is an EX sub module for each agent in the BG module. Each EX submodule closes a feedback control loop for its agent. For continuous systems, each EX submodule detects errors between its plan and the observed state of the world and issues output designed to null the errors. For discrete event systems, each EX sub module increments its controller from step to step along the plan as events are detected by the SP modules and reported to the EX by the WM modules, modifying parameters as necessary to correct errors between observed states and the plan. Each EX submodule produces output subtask commands that become input task commands to subordinate BG modules.

Df: **agent**
an entity that plans and executes jobs (possibly in coordination with other agents)

An agent is typically a peer member of a BG unit at one level and a commander of a BG unit at the next lower level. As such, an agent bridges the gap between two BG operational units at two levels of the organization.

Each agent contains four sub modules:

1. A SCheduler (SC)
2. A Plan Selector (PS)
3. An EXecutor (EX)
4. A Job Assignor (JA)

An agent may be a person or a computer control module. An agent within a BG module at one level may negotiate with the JA sub module of the supervisor agent, and bid against, or cooperate with, peer agents for job assignments. The SCheduler submodules of an agent and its peers work together with the Job Assignor submodule of its supervisor to formulate tentative plans which may be simulated and evaluated. The Plan Selector picks the best of the tentative plans. The EXecutor submodule of an agent carries out the plan and the JA submodule of an agent gives assignments and works with subordinates to formulate plans for the subordinate BG unit.

Each agent within the BG operational unit may compete with, or cooperate with, other agents within the same BG module. The BG module integrates the actions of the separate agents so as to coordinate their planning and execution functions within an operational unit. The set of agents within the BG module typically work together as a team to accomplish tasks assigned to the BG module of which they are a part. The grouping of agents within a BG module implies tight coupling between the Job Assignment and agent scheduling functions as is required for teamwork between supervisor and subordinates in planning and execution.

The JA submodule that is the supervisor of a BG operational unit at one level is part of an agent at the next higher level. The JA submodules that (along with SC, PS, and EX submodules) belong to peer agents within a BG operational unit, act as supervisors of BG operational units at the next lower level.

*****Note*****
The 4-D/RCS convention defines agents as separate from the SP, WM, VJ, and KD modules that support them within a node. In most of the artificial intelligence literature, the definition of an agent typically includes the SP, WM, VJ, and KD functions within the agent itself. There is intended to be no effective operational difference between the 4-D/RCS and the AI conventions. The 4-D/RCS convention is chosen because it is useful to treat the SP, WM, VJ, and KD modules separately.
****

Df: **task**
There are two types of task:

1. a piece of work to be done by a team of agents in a BG module on a set of objects to achieve a task goal.
2. an activity to be performed by a team of agents in a BG module to achieve or maintain a task goal state.

For type 1 tasks, achieving the task goal terminates the task. For example, a type 1 task may be <to drive to a location>, <to look for enemy tanks>, <to assemble an engine>, or <to shop for groceries>.

For type 2 tasks, the objective is to maintain the task goal until a different task is commanded. For example, a type 2 task may be <to maintain a condition> such as a temperature, velocity, or altitude.

A task implies action. In natural language, verbs are used to connote action. In the above definition of task, the infinitive form of the verb is used (e.g. task = <to drive to a location>). The application of the verb <do> to the <task> transforms the task verb from its infinitive to its active form. It puts the task verb into the form of a command. For example, <do_<to look for enemy tanks>> becomes a command to <look for enemy tanks>, where <look for> is the action verb, and <enemy tanks> is the object upon which the action is done.

Df: **do_task**
a command to perform a task

For any task, there can be assigned an action verb that is a command to perform the task. The set of tasks that a BG module is capable of performing, therefore, defines a vocabulary of action verbs or commands that the BG module is capable of accepting.

A task such as <to drive to a location>, may be performed by a number of agents (drivers) on many objects (e.g., tanks, trucks, HMMWVs, Bradley vehicles). A task such as assemble an engine may be performed by a number of agents (assembly line workers) on a number of objects (e.g., pistons, bearings, gaskets, head, oil pan). At the input of the BG module, a task command such as <look for tanks> or <assemble an engine> is encoded as a single command to do a single activity to achieve a single goal. The JA sub module at the input of the BG module performs the job assignment function of decomposing the task into a set of jobs for a set of agents (a suite of RSTA sensors, or an assembly line team). For each agent, a SCheduling sub module schedules a sequence of subtasks, which are executed by the EX submodule within each agent so as to accomplish the commanded task.

Df: **task command**
a command to a BG module to do a task, or to modify a previous task command

A task command can be represented as an imperative sentence in a message language that directs a BG module to do a task, and specifies the object(s), the task goal, and the task parameters. A task command has a specific interface defined by a task command frame.

A BG module accepts task commands with goals and priorities, formulates and/or selects plans, and controls action. The BG module develops or selects plans by using a priori task knowledge, heuristics, and value judgment functions combined with real-time information and simulation capabilities provided by world modeling functions to find the best assignment of tools and resources to agents, and to find the best schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). Behavior generation controls action by comparing current state feedback with the desired state specified by the plan, and using a control law to compute the best action to null the difference. Learning may enable behavior generation to acquire system models and optimize control laws.

Df: **task command frame**
all the information necessary for commanding a BG module to do a task

A task command frame includes:
1. task name (from the vocabulary of tasks the receiving BG module can perform)
2. task identifier (unique id for each task commanded)

3. task goal (a desired state to be achieved or maintained by the task)
4. object or objects (on which the task is to be performed)
5. task parameters (such as speed, force, priority, constraints, coordination requirements)
6. plan (in which the commanded task is the first step)

Item #6 in the task command frame means that the current task command presented to the BG module at level i-1 is not context free. It is but the first in a sequence of task commands that represent the current plan at the level i.

Df: **interface between BG modules at level i and level i-1**
the interface between BG modules at level i and level i-1 consists of the task command frame, plus the status response which returns to level i either directly, or through the world model

For any task to be successfully accomplished, there must exist task knowledge that describes how to do the task. Such task knowledge may include a list of the tools and materials necessary to accomplish the task, a list of conditions necessary to start or continue the task activity, a set of constraints on the operations that must be performed, and a set of information, procedures, and skills required to successfully execute the task, including error correction procedures and control laws that describe how to respond to error conditions. This task knowledge, together with information supplied by the task command, can be represented in a task frame.

Df: **task frame**
a data structure specifying the all the information necessary for accomplishment of a task

A task frame may include:
1. task name (from the vocabulary of tasks the receiving BG module can perform)
2. task identifier (unique id for each task commanded)
3. task goal (a desired state to be achieved or maintained by the task)
4. task objects (on which the task is to be performed)
5. task parameters (that specify, or modulate, how the task should be performed)
6. agents (which will be responsible for executing the task)
7. task requirements (tools, resources, conditions, state information)
8. task constraints (upon the performance of the task)
9. task procedures (plans for accomplishing the task, or procedures for generating plans)
10. control laws and error correction procedures

A task frame is essentially a recipe that specifies the materials, tools, and procedures for accomplishing a task.

Df: **task object**
a thing in the world that is acted upon by an agent to accomplish a task. For any object, there can be assigned a word (noun) which is the name of the object upon which the task is performed

Df: **task parameters**
modifiers that specify, identify, or prioritize a task, or modulate the manner in which a task is to be performed

Df: **tool**
a device that may be used to perform a task

Df: **task decomposition**
a process by which a task given to a BG module at one level (i) is decomposed into a set of sequences of sub tasks to be given to a set of subordinate BG modules at the next lower level (i-1)

Task decomposition consists of six generic functions:

1. a job assignment function is performed by a JA sub module whereby:
   a) a task is divided into jobs to be performed by agents,
   b) resources are allocated to agents
   c) the coordinate frame in which the jobs are described is specified
2. scheduling functions are performed by SC sub modules whereby a job for each agent is decomposed into a sequence of sub tasks (possibly coordinated with other sequences of sub tasks for other agents).
3. plan simulation functions are performed by a WM module whereby the results of various alternative plans are predicted
4. evaluation functions are performed by a VJ module on the results of simulations using a cost/benefit formula
5. a plan selection function is performed by a PS sub module in selecting the "best" of the various alternative plans for execution
6. execution of the best plan generated by functions 1-5 is performed by EX sub modules. The plan consists of a set of desired actions and a set of desired subgoals. The set of desired actions form a set of feedforward reference trajectories. The set of desired subgoals are compared with observed state trajectories and differences are treated as error signals that are submitted to a control law to compute feedback compensation. Feedforward and feedback compensation are combined to produce sequences of sub task commands to BG modules at the next lower level in the control hierarchy.

The result of behavior generation is that each task (represented by a single task command) presented to a single BG module generates a behavior defined by a set of subtask command sequences presented to a set of subordinate BG modules.

Df: **job**
an activity assigned to an agent within an organizational unit intended to transform a starting state into a goal state

A job is the output of a Job Assignor, and the input to an agent. A job consists of a description of the work to be done, the job goal, the resources and tools necessary to carry out the job, and whatever constraints and requirements that exist for cooperation with other agents. In the 4-D/RCS reference architecture, the only difference between a job and a task is that a task is something to be done by a BG module, and a job is something to be done by an agent. A job can have a job goal, a job command, a job command frame which are entirely analogous to a task goal, task command, and task command frame.

Df: **plan**
set of subtasks and subgoals that are designed to accomplish a task or job, or a sequence (or set of sequences for a set of agents) of subtasks intended to generate a sequence of subgoals leading from the starting state to the goal state

Figure 4 illustrates a task plan consisting of three job plans for three agents. Each job plan can be represented as a state graph where subgoals are nodes and subtasks are edges that lead from one subgoal to the next. A job plan can also be represented as a set of way points on a map, such that the waypoints are subgoals and the paths between waypoints are subtasks.

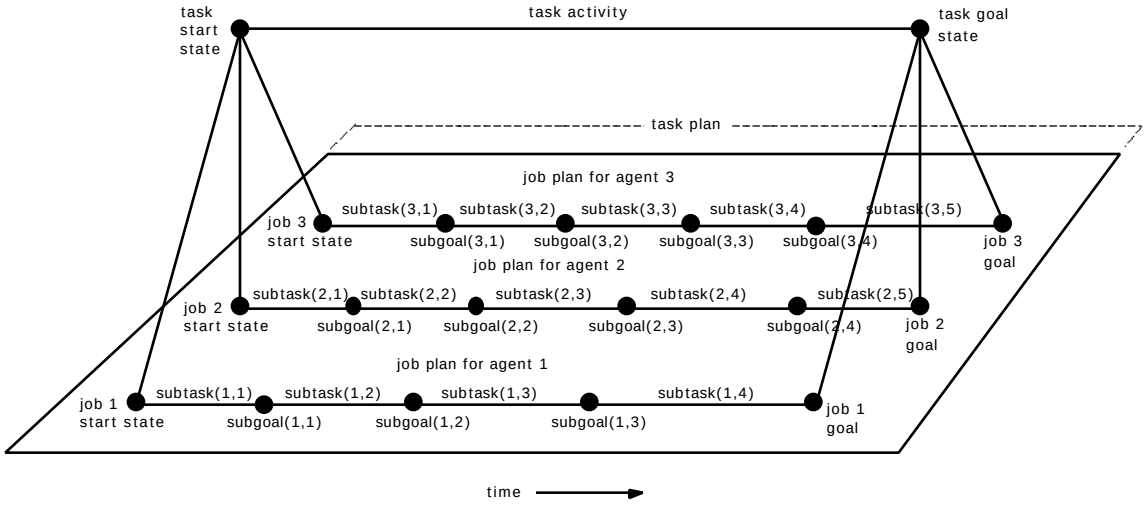


Figure 4. A task plan for three agents. The task is decomposed into three jobs for three agents. Each job consists of a string of subtasks and subgoals. The entire set of subtasks and subgoals is a task plan for accomplishing the task. In this example, a task plan consists of three parallel job plans.

A plan may be represented as a path through state-space from an anticipated starting state to a goal state. A plan may consist of a set of sequences of intended steps, or actions to be done, or sub tasks to be performed by a single agent (a job plan), or by a set of possibly coordinated agents (task plan). A plan may be represented as an augmented state graph, a Pert Chart, a Gantt Chart, or a set of reference trajectories through state space. Alternatively, a plan can be represented as a program, or flow chart, that generates a set of sequences of sub task commands for a set of agents so as to define a set of actions that drive a controlled system through state space from a start state to a goal state while using resources and satisfying constraints.

In general, a task plan consists of:

1) an assignment of job responsibilities to a set of agents,
2) an allocation of resources to the agents,
3) a set of (possibly coordinated) schedules that define a set of sequences of activities for the agents designed to achieve or maintain a goal state while satisfying a set of constraints.

The three job plan state graphs in Figure 4 are simple linear lists of subtasks and subgoals. In practice, each job plan state graph could have a number of branching conditions that represent alternative actions that may be triggered by objects or events detected by sensors. For example, each job plan might contain emergency action sequences that would be triggered by out-of-range conditions.

Coordination between agents can be achieved by making state transitions in the state graph of one agent conditional upon states or transitions in states of other agents, or by making state transitions of coordinated agents dependent on states of the world.

Df: **schedule**
the timing specifications for a plan. A schedule can be represented as a time labeled, or event labled, sequence of activities or events

Df: **planning**
a process of generating and/or selecting a plan

In general, planning consists of the following elements:

1. assigning jobs to agents and transforming coordinates into agent frames
2. allocating resources to agents
3. generating a tentative schedule of concurrent and possibly coordinated actions for each of the agents
4. simulating the likely results of this tentative schedule of actions
5. evaluating the cost/benefit value of these likely results
6. selecting the tentative schedule with the best evaluation as a plan to be executed.

A diagram of a typical planning process is shown in Figure 5. Planning elements 1, 2, and 3 are performed by JA, which assigns jobs and resources to agents and transforms coordinates from task to job coordinates (e.g. from way point coordinates to steering coordinates, or from map coordinates to camera image coordinates).

Element 4 is performed by SC submodules, which compute a temporal schedule of sub tasks for each agent and coordinates the schedules between cooperating agents (e.g. coordinated steering and throttle/brake trajectories to generate a desired velocity trajectory, or coordinated motion between vehicles to maintain formation).

Together, the assignment of jobs and resources to agents, the transformation of coordinates, and the development of a coordinated schedule for each agent, constitutes the synthesis of a plan. Therefore, output from the JA and SC is a tentative alternative set of subtask actions. JA and SC may generate many tentative alternative subtask action plans. Which of these is best (i.e., most efficient, or most effective, or lowest cost, or highest payoff) can be determined by simulating the results of various alternative tentative plans in the World Model (element 5). Each alternative sequence of actions will produce an expected sequence of states, or subgoals leading to the task goal. The value judgment module evaluates the cost and benefits of the sequence of subtask actions and/or the resulting subgoal states (element 6). Evaluations produced by the VJ module are returned to the Plan Selector (PS) sub module for a decision as to the best plan of action (element 7).

This process can be iteratively repeated until the entire universe of possible plans is searched. Heuristic search methods are typically used to reduce the number of alternatives that are actually examined. Eventually, the best plan is submitted to the EXecutors for execution.

These seven elements need not always occur in the order in which they are listed. Specifically, elements 1-4 may be reordered, or intermingled, and iterated many times to generate many alternative assignments of jobs or resources, or alternative schedules, each combination of which needs to be simulated and evaluated by elements 5 and 6 before a plan emerges from element 7.

Planning can be accomplished through a variety of methodologies. These can range from simple table look-up of precomputed plans (or scripts), to dynamic programming, heuristic AI search in state space or configuration space, or game theoretic algorithms for multi-agent systems. Planning techniques may differ at different levels in the 4-D/RCS hierarchy. At

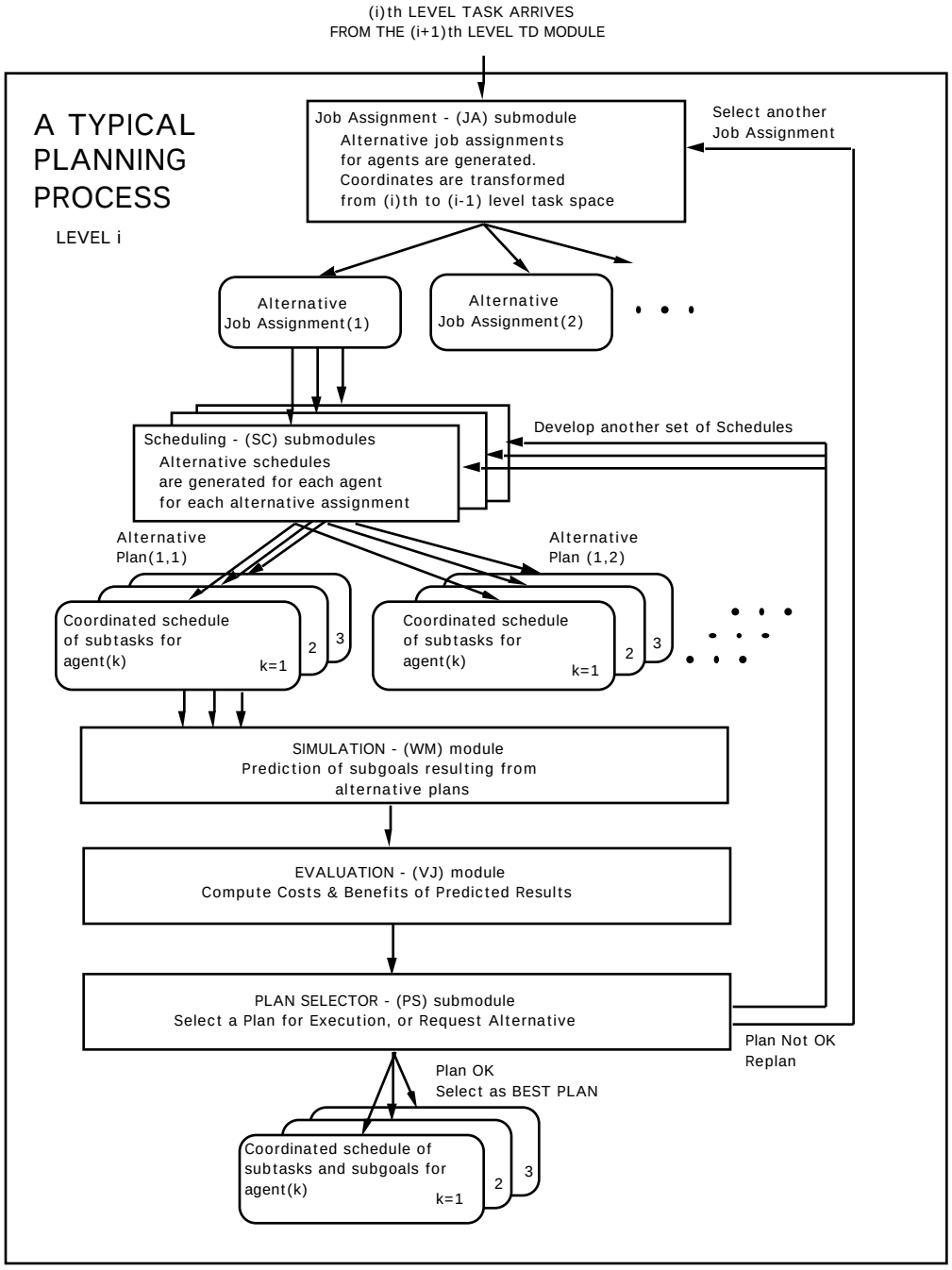


Figure 5. The diagram of planning operations within a 4-D/RCS node. Boxes with square corners represent computational functions. Boxes with rounded corners represent data structures.

the lowest levels, plans may be simple linear or spline interpolation between goal points. At the vehicle level, path plans may be generated by recording GPS and inertial data, or by graphic map-based path planning tools. At higher levels, mission planning tools and terrain fly-through methods may be used to generate tactical maneuvers and logistics scheduling.

Current practice is that planning is done well in advance of execution, and plans are selected at execution time by simple rules. Modifications, if necessary, are done by manual ad hoc methods that depend on the skills and intuition of officers and soldiers in the field. On the battlefield, prearranged plans are typically good only until the first shot is fired. Plans need to be modified often and quickly to adapt to the flow of battle, to take advantage of openings in the enemy defense, to compensate for losses, or to deal with unanticipated actions by the enemy.

Planning may be a distributed process that is done by many different planners working in many different places at different times. For example, strategic planning typically is done at a different hierarchical level by different operational units than tactical planning and long before the battle begins. Typically, each operational unit is required to develop its own plans, scaled to its own time horizon and region of terrain, based on the plans developed at levels above, and constrained by the resources available and conditions on the ground.

Regardless of how, where, or when planning is done, in each stage of planning, the seven elements listed above typically come into play. Regardless of how plans are synthesized, a plan eventually must specify the decomposition of tasks for operational units into sets of job assignments for agents, an allocation of resources, and a schedule of subtasks for each agent ordered along the time line. This may need to be done iteratively and hierarchically through many levels and by many different agents in different organizations before the plans are ready for execution.

In all cases, plans must be synthesized prior to execution. In highly structured and predictable environments, plans may be computed off-line long before execution. For example, in manufacturing, completely precomputed plans, such as NC machine tool code can be developed months or years before execution. In this case, real-time planning consists simply of loading the appropriate programs at the proper time. However, this only works for processes where there are no unexpected events that require replanning, or where human operators can intervene to make the necessary adjustments. As the uncertainty in the environment increases and the demands for agility grow, plans need to be computed nearer to the time when they will be executed, and be recomputed as execution proceeds to address unexpected events and to take advantage of unpredicted availability of resources. For military operations that depend on the state of many factors that cannot be known far ahead of time, operational plans need to be computed close to execution time, and recomputed whenever conditions require.

The repetition rate of replanning must be at least such that a new plan is generated before the old plan is completed. However, in highly uncertain environments, it is best if the planner generates a new plan before the EXecutor completes the first step or two of the old plan. Of course, the ability to react is also a function of the Executor. As intelligent controllers incorporate more and more on-line sensing and in-process measurement, the importance of EXecutor reactive measures will increase.

4-D/RCS is specifically structured to support real-time planning. Planning within the 4-D/RCS architecture is distributed at many hierarchical levels with different planning horizons at different levels. This facilitates the ability to replan fast enough to keep up with the demands of a rapidly changing environment. At each hierarchical level, the planning functions compute plans that extend from the current state out to a planning horizon that is characteristic of that level. On average, planning horizons shrink by about an order of magnitude at each lower level, and the number of subtasks to move from the current state to the planning horizon remains relatively constant (on average ten). Thus, the temporal resolution of subtasks increases about an order of magnitude at each lower level.

Planning horizons at high levels may span months or years, while the planning horizon at the bottom level typically spans around 30 ms. The number of levels required in a 4-D/RCS hierarchy is approximately equal to the logarithm of the ratio between the planning horizons at the highest and lowest levels.

The 4-D/RCS can accommodate either precomputed plans, or planning operations that recompute plans on demand, or on repetitive cyclical intervals. For example, a string of GPS coordinates can be treated as a stored path plan, and called when appropriate. The 4-D/RCS generic planning capability can also accommodate partially completed plans. For example, route plans computed in advance can be used by 4-D/RCS real-time planners to provide constraints for real-time generation of obstacle detour routing plans.

Regardless of how they are derived, plans must be expressed in a vocabulary of subtask commands and subgoal results that can be accepted as input by the EXecutors at that level.

Df: **plan execution**
a process of executing a plan

Planning is a process that looks into the future and anticipates what needs to be done in the interval between the present and the planning horizon. Plan execution is a process that acts in the present to follow the plan and correct for errors.

For each agent(j) in the BG module, there is a EXecutor EX(j) that executes and monitors the plan for that agent. The generic structure of an EXecutor is shown in Figure 6. Each EXecutor receives as input from the plan selector a plan consisting of both a trajectory of subtask commands and a corresponding trajectory of planned subgoals. In 4-D/RCS, plans are expressed in the form of an augmented state graph, where the planned subgoals are nodes and planned subtasks are the edges that connect the nodes as illustrated in Figure 4.

Planned subtask commands are essentially feedforward actions that enable the EXecutor to anticipate the behavior of the system being controlled so as to produce the desired output as closely as possible. This tends to reduce feedback error to a low level. Feedforward control is particularly important for systems with inherent delays (such as the delay between turning the steering wheel and a change in vehicle motion direction.) For systems with delay, feedback signals may arrive too late to compensate for some errors, or worse may cause the system to become unstable.

In addition to the plan, each EXecutor also receives estimated state feedback via Sensory Processing and World Modeling modules from sensors that monitor the state of the world and the internal state of the system. Each EX module has a compare and monitor submodule that computes the difference between the estimated state feedback and the current planned subgoal. This computation results in three types of output:

First, the feedback error is used by the control law to compute error compensation that is combined with the feedforward action to instantiate the output command to the next lower level BG module. This enables each EXecutor function as a servo control loop, steering its agent to follow the desired reference trajectory defined by the plan.

Second, the compare and monitor submodule monitors progress in achieving the current subgoal. When it determines that the estimated feedback state has come within an acceptable neighborhood of the current planned subgoal, it emits a <subgoal_done> signal to the subtask sequencer. This causes the next subtask and subgoal in the plan to be selected for execution.

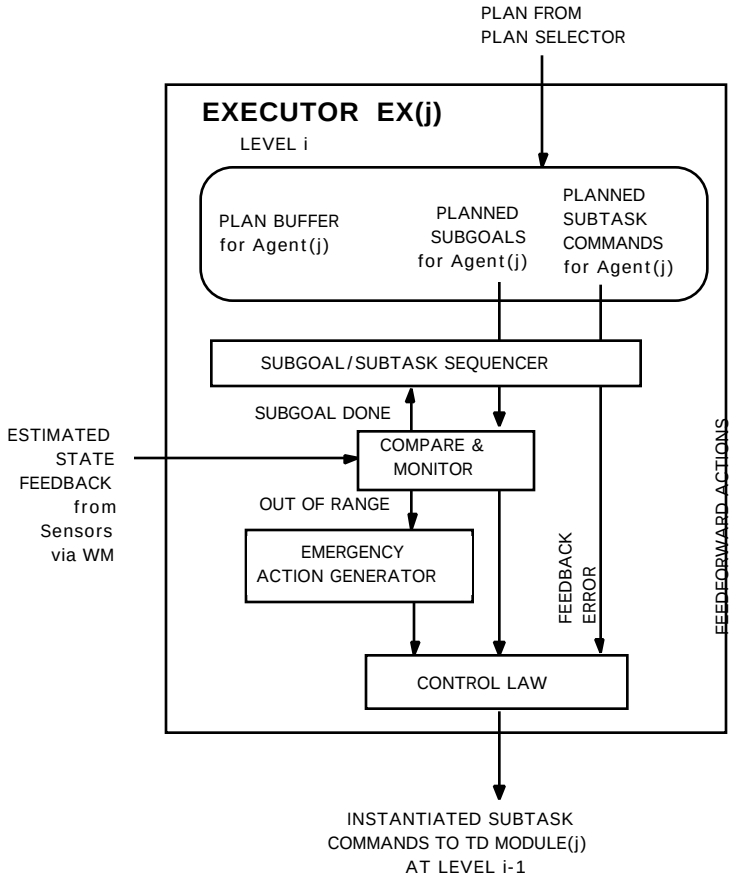


Figure 6. The inner structure of the EXecutor (EX) sub module.

Third, the compare and monitor submodule detects out-of-range conditions. When this occurs, an emergency action generator is triggered to immediately substitute an appropriate emergency action plan for the current plan, and the planner is asked to generate a new plan.

In 4-D/RCS, feedback errors are detected and addressed at the lowest levels first, where the response can be the quickest. If the lowest level error correction processes are successful, the upper levels can continue according to plan. However, if errors cannot be corrected by low level responses, then higher level nodes must become engaged to change tactics or strategy to solve the problem. Thus, error correction ripples up the hierarchy from EXecutor to planner at each level, until either a solution is found, or the entire system is unable to cope, and a mission failure occurs.

For each agent, the EX submodule provides the first line of defense against failure. The EX acts immediately and reflexively, as soon as an error condition is detected, to correct it by feedback compensation, if possible, or by a preplanned emergency action, if available. If this is successful, the error is corrected, and the agent proceeds with the current plan. However, if both error compensation and preplanned remedial actions are unsuccessful,

and the error persists or exceeds limits, then replanning is required. While the planner takes time to generate a new plan, the emergency action generator executes a preplanned emergency action (such as stop, or take cover) until the new plan is ready.

At all except the lowest level, outputs from EXecutors become input task commands to Job Assignors in BG modules at the next lower level. At the lowest level, outputs from EXecutors go to actuator drive mechanisms.

An Example of Planning and EXecutor Functions and Timing for Unmanned Ground Vehicles

Planner function

In Figure 7, a driving task <Go_to_point(a)> requiring about an hour is input as a command to the Platoon level. The planner at the Platoon level looks ahead to a planning horizon about an hour in the future, and decomposes the task into a plan <Go_along_path(b)> from a starting point (S) to a goal point (a). The path is defined by a set of way points, or subgoals (a1), (a2), (a3), . . (a9) leading to the goal point (a). If the waypoints are equally spaced, they will be about five minutes apart.

Figure 7 shows the planning and execution activity at all levels of the 4-D/RCS hierarchy as this task is carried out. At each level (i), the planner looks ahead to a planning horizon a time T(i) in the future. It computes a plan consisting of about ten way-points from the current state to that planning horizon. Thus the plan generated at each level (i) will have steps approximately T(i)/10 apart in time. Each of the way-points in this plan will eventually become a goal for the planner at the next lower level (i-1). Thus, at each lower level of the 4-D/RCS hierarchy, the planning horizon shrinks by a factor of ten, and the distance between way-points in the plan decreases by a factor of ten.

In this simple example, a plan is expressed as a simple linear string of waypoints such as might be encountered while driving on a road. More complex plans might be expressed as a state graph with a number of conditional branch points, such as might be encountered when driving through a network of city streets. For cross-country driving, it may be appropriate to develop plans by computing vector fields which direct the vehicle away from obstacles, but allow freedom of choice to lower levels of behavior generation while the vehicle is in open areas. A vector field is simply a means for generating planned waypoints by gradient descent.

In the example of Figure 7, at each level(i), ten about equally spaced way-points are computed between t=0 and a planning horizon at t=T(i). At a given velocity, this yields discrete way-points on the ground in front of the vehicle with a moving goal at a distance from the moving vehicle proportional to the velocity. As the vehicle moves along its planned path, the planning horizon also moves ahead. For example in Figure 8, when the vehicle is at the start (S), the planning horizon at level (4) looks forward 30 s (a distance of 300 meters at a speed of 36 km per hour) to the next goal point from level (5) which is (a51). It generates a planned path(4) consisting of (a41), (a42), (a43), . . . , (a49) and ending in (a51). As the vehicle begins to move, about 3 s later it reaches (a41). The planning horizon is now beyond the first goal point (a51). The planner must now begin planning for reaching the second goal point (a52). The planning horizon is indicated in Figure 8 by a rectangular box whose left side is at the present (t = 0), and whose right side is at the planning horizon (t = T(4)).

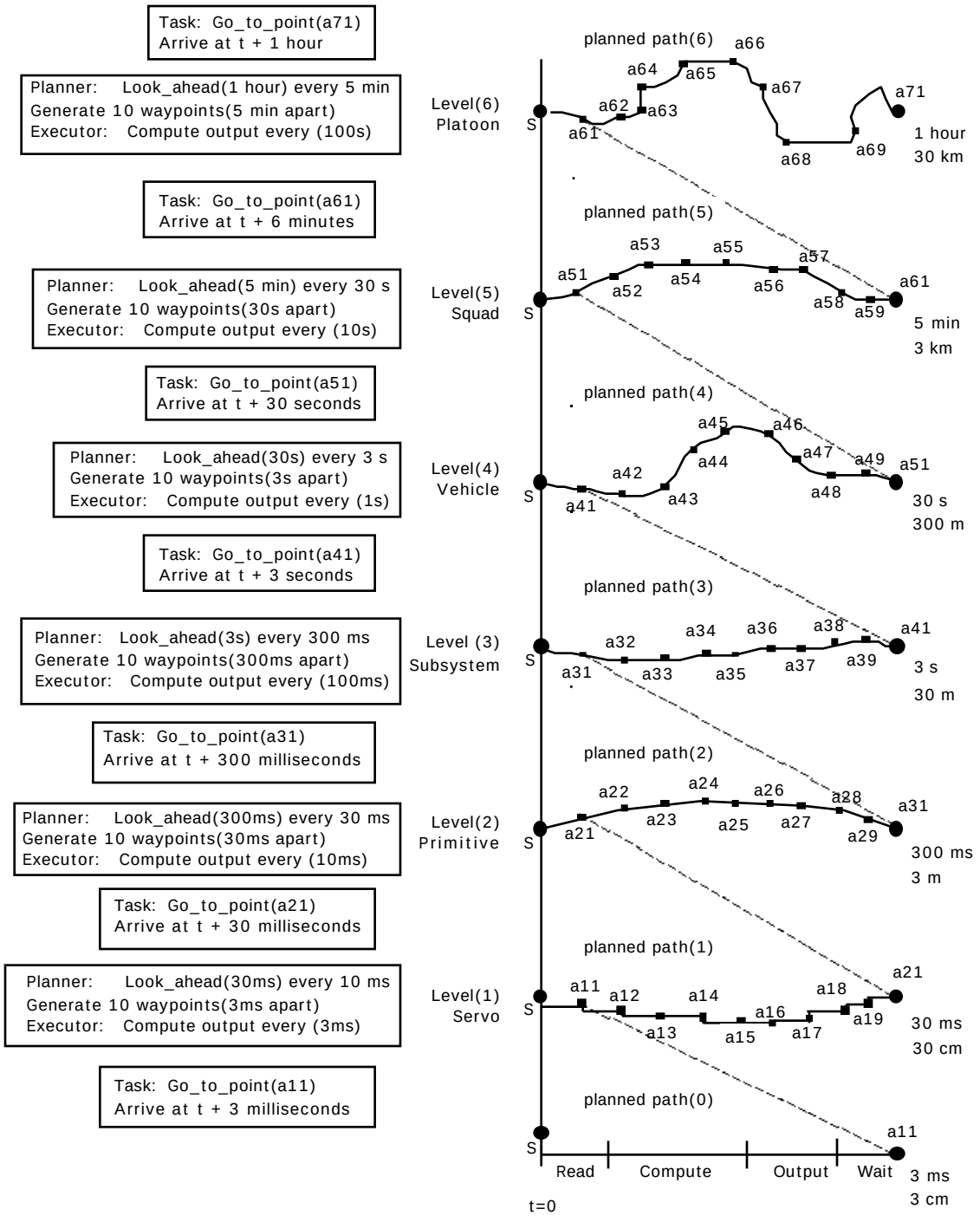


Figure 7. A 4-D/RCS timing diagram. On the right, a typical planning horizon and a planned path defined by about ten waypoints is shown for each level. At the bottom, a single servo executor cycle (read, compute, output, wait) is shown. On the left, an example of a typical task command frame and the functions performed by the planner and executor is shown for each level.

As can be seen in Figure 8, in order for a planner at level (4) to plan to a moving horizon T(4) in the future, it typically must have access to at least the next way-point beyond T(4) in the plan of the higher level(5). In Figure 8, the first waypoint in the Squad level(5) plan is (a51). The next waypoint beyond T(4) is the second waypoint in the Squad level(5) plan which is (a52). Typically, the planner at a level(i) does not require access to more than the next two waypoints at level(i+1).

Planner timing

For real-time planning, the planner at a level must be able to generate a new plan in less time than it takes to execute the old plan (i.e. <T(i)). Otherwise the system must periodically stop and wait for the planner to complete its work. This type of behavior was typical of early efforts in real-time planning, such as the Shaky robot at SRI. However, pausing periodically to wait for a planner is unacceptable for most practical vehicle applications.

In dynamic unpredictable environments, such as the battlefield where the world and the goal may be changing rapidly, the planner should be able to replan in the time it takes to execute the first way-point in the old plan. In this case, the planner will run every T(i)/10, and must run at 10x real-time. This is illustrated in Figure 8 for path planning at level(4), the Vehicle level. In Figure 8, a series of four planning cycles is shown. The planned path from the next higher level, level(5) Squad, is shown as input to the level(4) Vehicle task planner. This input plan consists of waypoints (a51) and (a52). The level(4) Vehicle planner takes this input and generates a planned path(4) that extends out to the level(4) planning horizon T(4) = 30 s. When the vehicle is at position S, a level(4) Vehicle planned path is generated from S to a51. This consists of waypoints (a41), (a42), . . . , (a49). By the time the vehicle has reached the first waypoint (a41), another plan is generated again out to the planning horizon T(4) = 30 s, which now is beyond the point (a51). All the planned waypoints are shifted left (a41 <- a42, a42 <- a43, . . . , a48<- a49) and a new way point is generated for (a49). As this procedure is repeated, the vehicle moves along the planned path. As it passes each waypoint on its planned path, it shifts its planned waypoints left in the planned list, and adds a new waypoint at the planning horizon. At each level(i), the planner thus always has a plan that extends out to its planning horizon at T(i). As it achieves each step in its current plan, it computes a new plan with an additional step at the planning horizon. Of course, in an uncertain environment, conditions may change as the vehicle moves along its planned path. This may require that replanning process alter some of the intermediate waypoints, as well as add a new waypoint at the planning horizon.

It may be possible to conserve computational resources if the planner generates plans with logarithmic resolution, with the first steps in the plan at intervals of T(i)/10, and steps near the planning horizon at longer intervals. Replanning less frequently is a second alternative for conserving computational resources. If the replanning interval is T(i)/3, a new plan will be generated only every three or four way-points. The disadvantage with this second alternative is that the system cannot respond as quickly to changes in the environment.

Executor timing

The executor should supply a new goal point whenever the planner of the next lower level is ready to begin its planning cycle. It serves no purpose for the executor to supply new goal points faster than the next lower level planner can accept them. Therefore, the executor timing is based on the replanning cycle at the next lower level. If the replanning cycle at each level is T(i)/10, then the executor at level (i) must produce an output every T(i)/100. (The planning horizon at level(i-1) is T(i)/10, and the planner at level(i-1) replans in 1/10 of its planning horizon). If the planning cycle at each level is T/3, then the executor

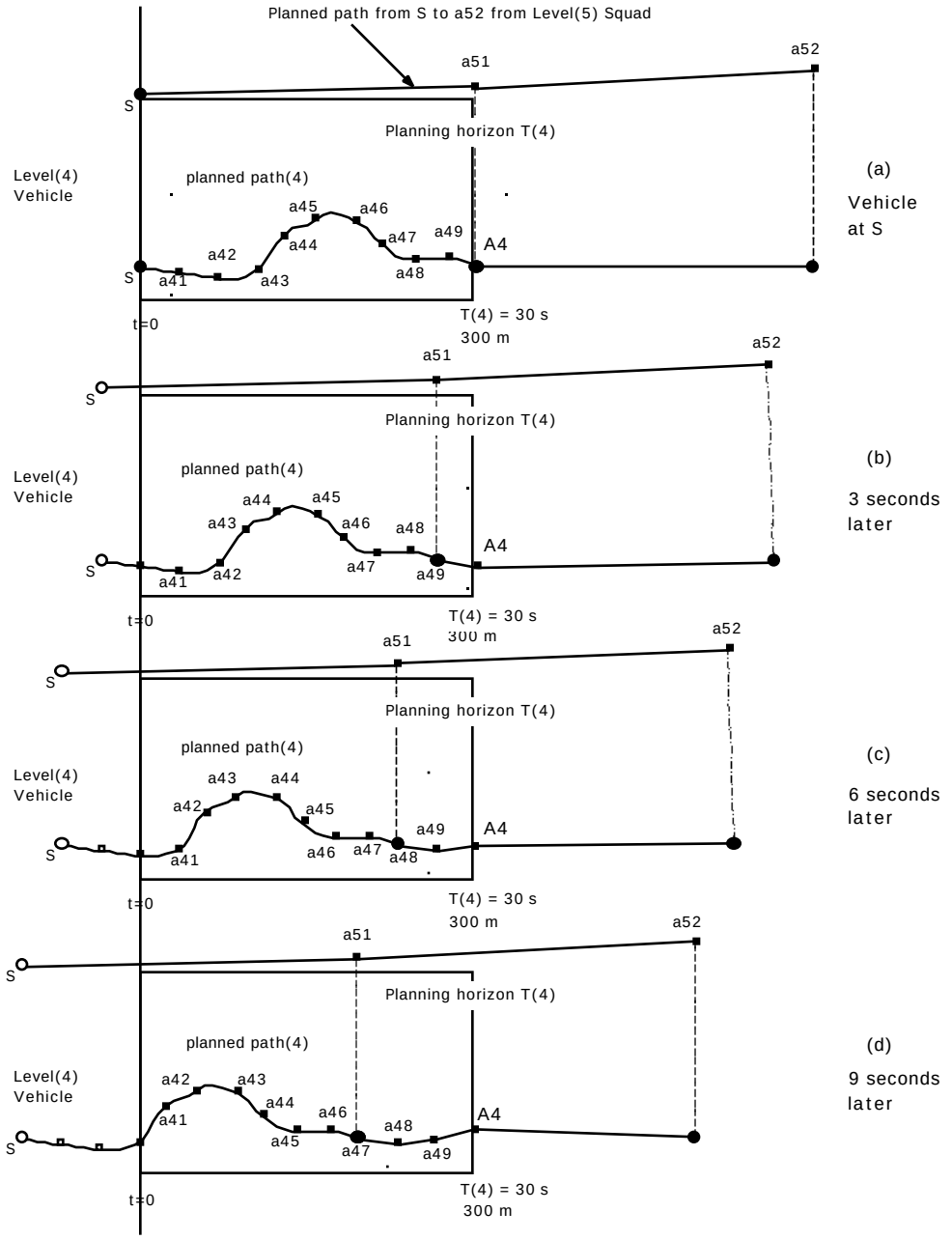


Figure 8. Planning at T(i)/10 intervals. At time (a), when the vehicle is at point S, input to level(4) is a command <Go-to a51> with a low resolution Squad plan to go from S through waypoint a51 to a52. At time (a), the level(4) Vehicle planner generates a higher resolution Vehicle plan (a41, a42, a43, . . . , a49) from S to a planning horizon 30 s in the future. 3 s later, at time (b), about when the vehicle has reached waypoint a41, the level(4) planner has generated a new plan. The waypoints are shifted down in the plan, and a new waypoint is added at the end. This replanning process is repeated again every 3 s, at time (c) and (d).

must produce a new output every T/30. (The planning horizon at level(i-1) is T(i)/10, and the planner at level(i-1) replans in 1/3 of its planning horizon.)

The executor cycle timing is also a function of the filter integration period in the recursive estimation processes at the same hierarchical level. It makes no sense for the executor to sample the output of an estimated state variable at a rate faster than the state estimate can change.

World Modeling (WM) modules

The WM modules perform four basic functions illustrated in Figure 9:

1) Maintenance and updating of information in the Knowledge Database (KD).
• In each node the WM module maintains the KD, keeping it current and consistent.
• The WM module updates the state estimates in the KD based on correlations and variance between world model predictions and sensory observations at each node. This may be part of a recursive estimation loop that operates at the data sampling rate.
• The WM module updates both images and symbolic frame representations and performs transformations from image to symbolic representations, and vice versa.
• The WM module enters new entities into the KD, and deletes entities that are observed to no longer exist in the world.
• The WM module maintains the pointers between KD data structures that define semantic and pragmatic relationships between entities, events, images, and maps.

2) Prediction.
• The WM module generates short term predictions of expected sensory observations that enable sensory processing (SP) modules to perform correlation and predictive filtering.
• The WM module uses symbolic representations, iconic images, masks, and windows to support visualization, attention, and model matching.

Examples of predicted images at lower levels might be what a particular camera would be expected to produce, or what a series of tactile sensor readings might measure. At higher levels, predicted images might be digital terrain maps showing obstacles, buildings, roads, bridges, woods, and other terrain features. At still higher levels, predicted images might be maps of large regions.

3) Response to queries for information required by other modules.
• The WM module responds to “What Is?” queries from the TD (PL and EX sub modules) regarding the state of the world or the state of the controller.
• The WM module acts as a question answering system to compute responses for information that is not explicitly stored in the KD.
• The WM module transforms information into the coordinate system requested by the query.

4) Simulation.
• The WM module performs simulations in response to “What If?” queries in order to support the planning functions of the TD modules.
• The WM module uses structural and dynamic models and rules of physics and logic to simulate the results of current and future actions.

Examples of simulation at lower levels might be of steering trajectories for high speed cornering. At higher levels, path plans for obstacle avoidance might be simulated. At

still higher levels, various routes from one location to another or various formations for tactical maneuvers might be simulated.

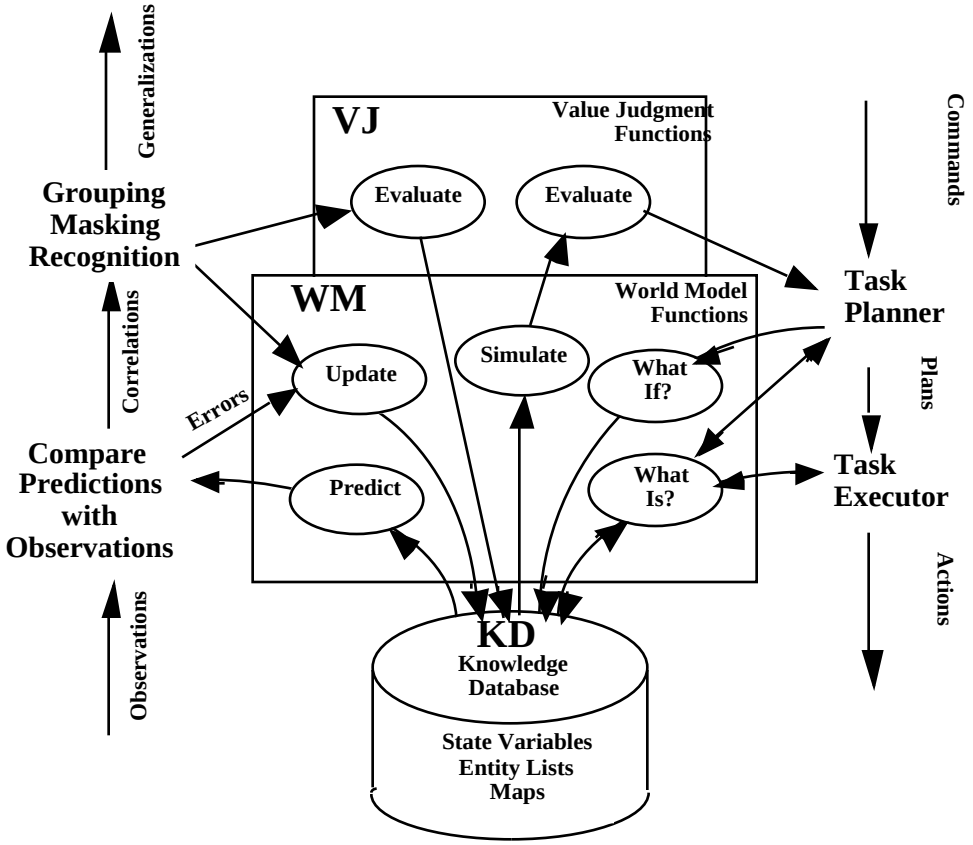


Figure 9. World Modeling (WM) and Value Judgment (VJ) modules. WM and VJ modules typically exist in every node of the 4-D/RCS architecture.

Value Judgment (VJ) modules

Figure 9 shows a block diagram of the functional operations and data flow pathways for interactions between world modeling and value judgment. VJ modules evaluate plan results simulated by the WM modules. VJ modules contain algorithms for the following: (1) computing the cost, risk, and benefit of actions and plans, (2) estimating the importance and value of objects, events, and situations, (3) assessing the reliability of information, and (4) calculating the rewarding or punishing effects of perceived states and events.

VJ modules compute the cost functions that enable intelligent behavioral choices. VJ modules define priorities, set limits on risk, and decide how aggressive or conservative a system should be in pursuing its behavioral goals. VJ modules assign values to objects, events, and situations recognized by SP modules. These assigned values may be used to compute whether it is worthwhile to defend a position, or to attack a target. Values placed on friend objects may define how valuable they are, how vulnerable to attack, and how

much in need of defense or rescue. Values placed on foe objects may define how dangerous they are, how aggressive they are toward the self vehicle. Values placed on regions of space may define how safe or risky it is to be in that space, how valuable a region of space is, how worth defending or attempting to occupy.

Values placed on objects or regions of space can be overlaid on images or maps. This enables image processing techniques to be applied to cost values similar to intensity values. Values of cost and risk in an image can be analyzed to find the lowest cost regions for path planning. Gradients of cost and risk can be analyzed to generate commands for steering, or aiming.VJ modules may compute statistics on the reliability of information about the world based on the correlation and variance between observations and predictions. Confidence values can be assigned to state variables and regions of the visual field. VJ modules evaluate perceived and planned situations to enable TD modules to select goals and set priorities. VJ modules also compute what is important (for attention) and what is rewarding or punishing (for learning).

VJ modules compute different types of values at different hierarchical levels. For example, at levels one and two, VJ may compute of cost functions for path planning that minimize jerk or acceleration. At levels three and four, VJ may minimize energy or time. At levels five and six, VJ may minimize risk or exposure to enemy action.

Knowledge Database (KD)

KD modules are the data structures and the static and dynamic information that collectively form a model of the world. The KD is the information needed to support the TD, SP, WM, and VJ processes in each node. Types of data structures include scalars, vectors, arrays, symbols, strings, lists, frames, and graphs. Information in the KD includes signals, state variables, images, maps, symbolic entities, symbolic events, attributes that describe entities and events, semantic and pragmatic relationships between entities and events, rules, equations, structural and dynamic models that describe how the world behaves, and task knowledge that describes how to do tasks, what tools to use, what resources are needed, and what information is required.

Df: **signal**
output from a sensor that measures a phenomenon in the environment

Df: **state variable**
a numeric or symbolic variable that represents the current estimated value of a property or attribute of an object or event in world

Df: **symbolic entity**
a data structure that represents an entity that exists in the world.

A symbolic entity can be represented by a list, or frame, consisting of a list head, a set of attribute-value pairs, and a set of pointers that define relationships with other symbolic entities.

Df: **attribute**
a property of an entity or a pixel

An entity may have attributes such as color, texture, size, shape, position, orientation, and motion. A pixel may have attributes such as intensity, color, spatial or temporal gradients, range, position in the image, and motion.

There are three types of attributes:

1. Observed attributes that are derived directly from sensors, or computed from other observed attributes.
2. Estimated attributes that are computed from observed attributes by recursive estimation, or another filtering technique.
3. Predicted attributes that are computed from estimated attributes using knowledge of system dynamics, geometry, and planned actions.

Df: **symbolic event**
a data structure that represents a state change, or a set of states or situations that occur at a particular time and place, or of a sequence of states, or situations, that transpire over an interval of time and space in the world

A symbolic event can be represented by a list, or frame, consisting of a list head, a set of attribute-value pairs, and a set of pointers that define relationships with other symbolic events and entities.

Df: **rules and equations**
symbolic representations that express physical and mathematical laws that describe the way the world works and how things relate to each other in time, space, causality, and probability

Examples include If/Then rules, formulae in the predicate calculus, differential equations, control laws, geometrical theorems, and system models.

Df: **structural models**
rules and equations that describe how physical structures are kinematically connected and how forces and stresses are distributed

Df: **dynamic models**
rules and equations that describe how forces asnd inertias interact with each other in time and space

Df: **task knowledge**
knowledge of how to act in order to accomplish task goals

Task knowledge includes skills and abilities required for manipulation, locomotion, communication, and attention. Task knowledge may also include a list of equipment, materials, and information required to perform a task, as well as conditions required to begin or continue a task. For example, task knowledge may describe how to steer around an obstacle, how to follow a road, how to navigate from one map location to another, how to avoid being observed from a particular location, how to cross a creek or gully, how to encode a message, or how to load, aim, and fire a weapon. Task knowledge is used by the Task Decomposition module to plan and execute tasks.

Df: entity frame
a data structure that consists of a name, a list of attribute-value pairs; a set of pointers, and a set of criteria for recognition

An entity name is an address or index by which an entity frame can be accessed in a database. The list of attribute-value pairs describe properties or characteristics of the entity. The set of pointers define relationships between the entity and other entities in events and situations, and relationships between entity frames and images or maps. Entity relationships can represent class membership, or semantic, pragmatic, or causal meaning.

The criteria for recognition are the distinguishing characteristics of the topological, generic, or specific class which the entity represents.

Figure 10 shows the structure of a typical entity frame. The "entity name" is an identifier

NAME -- name of entity

attribute - value pairs

type -- topological, generic, or specific
topology -- point, edge, vertex, surface patch, boundary, surface, object, or group
position -- x, y, r [of c.g.], (uncertainty)
orientation -- roll, pitch, yaw [about c.g.], (uncertainty)
velocity -- vx, vy, vr, vroll, vpitch, vyaw, (uncertainty)
acceleration -- ax, ay, ar, aroll, apitch, ayaw, (uncertainty)
trajectory -- sequence of positions
geometry -- shape, size, axis of symmetry, etc.
properties -- mass, color, substance, etc.
behavior -- responds_to, acts_like, etc.
capabilities -- speed, range, armaments,
IFF -- friend or foe
value -- as_a_target or worth_to_defend

pointers to other entities

belongs_to -- parent entity
has_part -- subentity1
has_part -- subentity2
has_part -- subentity3
. .
. .

criteria for recognition

distinguishing characteristics of the class to which the entity belongs

Figure 10. The structure of a typical entity frame. Each entity frame consists of a name, a list of attributes, and a set of pointers to other entities. The list of "has_part" pointers may include pointers to the pixels in an image that are part of the entity, or a single pointer to an entity image (as illustrated in Figure 13).

of the entity defined by the frame. It can be used as an address, or index in a library or dictionary of entities. Entity frames exist in the knowledge database. Entity attributes may be derived from sensory data, or from a priori information. Entity attributes are the world model's best estimate of the state of real world entities. Entity attributes allow the world model to make hypotheses and predict what sensory data to expect

There are three types of entities:

1. Topological entities that are defined by topological class properties such as points, lines, surfaces, volumes, and groups.
2. Generic entities that are defined by generic class membership, such as a road, a tree, a bush, a building.
3. Specific entities that are defined by specific class membership, such as a particular road, a specific tree, a unique building.

Topological Entity Classes

4-D/RCS has the following set of topological entity classes: point, line, surface, object, group.

Point entity classes

Point (or pixel) entity frames contain attributes that can be measured by a single sensor at a single point in time and space, or that can be computed at a single point from a local neighborhood in time and space. Point attributes may describe the properties of a single pixel in an image. Examples of pixel attributes are intensity, color (rd, gn, bl), range, spatial and temporal gradients of intensity or range, flow direction and magnitude. Point attributes may also describe the output of a single sensor, such as a position, velocity, torque, or temperature sensor, at a point in time

Line entity classes

Line entity classes include edge, vertex, and surface_patch entities. Line entity classes consist of sets of contiguous point entities that satisfy some line gestalt grouping hypothesis. For example, an edge may consist of a set of contiguous pixels for which the first or second derivatives of intensity and/or range exceed threshold and are similar in direction. A vertex may consist of two or more edges that intersect, or a single edge with an endpoint. A surface_patch may consist of a set of contiguous pixels with similar first or second derivative of intensity and/or range and similar image flow vectors.

Edge, vertex, and surface_patch entity attributes are computed over the set of points that comprise the entity. For example, edge entity attributes may define the orientation and the length of the edge, the sharpness of the edge, or the magnitude of the discontinuity at the edge, as well as the centroid of the group of points that make up the edge. Vertex attributes may describe the relationship between the set of edges that make up the vertex. For example, vertex attributes may define the type of vertex (e.g. an endpoint, V, T, or Y), the orientation or the vertex, and perhaps the angles between lines forming the vertex. Surface_patch attributes may describe the collective properties of the set of connected points that make up the patch. For example, surface_patch attributes may define the position, and velocity of the surface patch, the texture, and the orientation of the surface patch relative to the viewing point.

Surface entity classes

Surface entity classes include surface and boundary entities. Surface entities consist of sets of contiguous line entities that satisfy some surface gestalt grouping hypothesis. For example, a surface may consist of a set of contiguous surface_patches that have similar range, orientation, texture, color as their immediate neighbors. A boundary may consist of a set of edge entities that are contiguous along their orientation. Surface and boundary attributes are computed over the entire set of points that are included within the entity. Surface attributes may describe the properties of the surface, such as its area, shape, roughness, texture, color, position and velocity of the centroid, etc. Boundary attributes may describe the properties of the boundary between two surfaces computed over the set of points that make up the boundary. For example, boundary attributes may define the shape of the boundary, its orientation, its length, its position, and velocity, which side each surface lies on, etc.

Object entity classes

Object entities consist of sets of contiguous surface and boundary entities that satisfy some object gestalt group hypothesis. For example, an object may consist of a set of surfaces that have roughly the same range and velocity, and are contiguous along their shared boundaries. Object attributes are computed over the entire set of points that are included within the object. Object attributes may describe the properties of an object, such as its volume, shape, projected size in the image, color, texture, position and velocity of centroid, orientation and rotation about the centroid.

Group entity classes

Groups entity classes consist of sets of objects that have similar attributes, such as proximity, color, texture, or common motion. Group attributes are computed over the entire set of objects that are included within the group. Group attributes may describe the properties of a group, such as the number of its members, size, density, position, velocity, and direction of motion.

The grouping properties of the above entity classes produces a hierarchical layering of inclusion or belonging relationships. Group entities have subentities that are objects. Object entities have subentities that are surfaces and boundaries. Surface and boundary entities have subentities that are edges, vertices, and surface patches. Edge, vertex, and surface patch entities have subentities that are point entities.

This topological hierarchy of entity classes in the world model knowledge database is parallel with, and can be overlaid on, the task decomposition hierarchy in the behavior generation side of the 4-D/RCS hierarchy. For many purposes, segmentation of the world into topological entity classes is sufficient for generating effective behavior. Simply knowing the position, size, shape, and motion of edges, surfaces, objects, and groups is all that is required to avoid collisions, and to compute whether the ground is too rough or too steep to be safely traversed.

In general, however, objects cannot be uniquely defined by topological relationships alone. For example, is a tree an object? Topologically, a tree has surfaces, each of which has edges, and surface patches, each of which consists of points. However, a tree also has parts, such as a trunk, branches, roots, and leaves. Is tree trunk an object? It also has surfaces, edges, and points. If the tree trunk is an object, what is a hole in the trunk? A hole also has surfaces, edges, and points. Thus, the definition of an object cannot be solely in terms of topology.

In general, the definition of an object, like beauty, is in the mind of the beholder. In 4-D/RCS, the definition of an object depends on the task assigned to the vehicle. For example, if the vehicle task is to approach a tree, then the tree is the object. If the vehicle task is to avoid hitting the trunk of a tree, then the tree trunk is the object. If the vehicle task is to inspect a hole in the tree trunk, the hole is the object. On the other hand, if the vehicle task is to avoid the woods, the group of trees comprising “the woods”, is an object with surfaces, boundaries, edges, and points.

Therefore, 4-D/RCS defines an object as follows:

Df: **object**
the entity upon which a task at the vehicle level of the 4-D/RCS hierarchy is performed

Thus, object entities are the object of tasks at the vehicle level(4) of the 4-D/RCS hierarchy. Group entities are the object of tasks at the Squad level(5) of the 4-D/RCS hierarchy. Surface and boundary entities are the object of tasks at the Subsystem level of the 4-D/RCS

hierarchy. Edge, vertex, and surface_patch entities may be the object of tasks at the Primitive level of the 4-D/RCS hierarchy. And point entities are the object of tasks at the Servo level.

Therefore, the layers in the KD entity class hierarchy correspond to, and are defined by, layers in the task decomposition process in the TD hierarchy.

Generic Entity Classes

A generic entity class is a set of entities that are similar to each other and have the same class name. For example, a generic entity may be a building, a tree, or a person. A generic entity class has attributes that define class membership. The set of attributes in a generic entity frame exemplify the class.

Specific Entity Classes

A specific entity is a particular instance of a thing. A specific entity has unique attributes that distinguish it from all others. For example, a specific object may be a specific building, a specific tree, or a specific person. A specific entity frame has attributes that uniquely define that entity.

There is an taxonomy of classes and hence an taxonomy of entity frames. For example, a specific tree may be a member of the generic class of oak trees, which is a member of the generic class of deciduous trees, which is a member of the generic class of trees, which is a member of the generic class of plants, which is a member of the generic class of living things. The taxonomy of classes can be represented in the knowledge database by a set of pointers, or links, between more generic and more specific classes. A pointer to a more generic entity frame is a “is_a” or “belongs_to” link. Pointers to more specific classes are “an_example_of” or “contains” links.

Additional attributes

In Figure 10, the attributes of position and orientation describe the pose of the entity (in a specified coordinate reference system. The velocity and acceleration attributes describe the 6 degree of freedom motion of the entity. The trajectory attributes define a sequence of poses over the recent past. Geometry attributes describe shape, size, symmetry, boundaries, etc. Properties define such things as mass, color, the substance from which the entity is made. Behavior describes how the object is likely to behave under certain conditions or in response to certain stimuli. Capabilities describes the relevant information as to range speed, armaments, etc. IFF defines whether the entity is a friend or foe. The value attribute defines how much an object is worth as a target (if a foe), or how valuable an object is to the friendly side (if friend). “Has_part” is a pointer to a subentity. “Belongs_to” is a pointer to the parent entity.

Situational and geometric relationships between symbolic entities can be described by semantic networks consisting of additional pointers such as “on-top-of,” “beneath,” ‘to-the-right-of,” “inside-of,” etc. Relationships such as “is-used-for,” “uses,” etc. can be used to define pragmatics. Relationships such as “causes,” is-caused-by,” etc. can be used to define causality.

Criteria for recognition

The criteria for recognition in the entity frame of Figure 10 define the range of attribute values that tend to distinguish the entity type. By comparing the attributes of an entity measured from an image with the criteria for recognition stored in the entity frame, an observed entity can be (at least tentatively) recognized as matching a stored entity. More details about the recognition process are contained in the section on Sensory Processing

Iconic Entities

An iconic entity is an image, or array of spatially distributed attribute values, that represent an entity. Typically, any entity can be represented in the world model knowledge database either by an entity frame, or by an iconic image, or both. For example, Figure 11 shows both an entity frame and an image representation for a particular object -- a tank. Entity frames and iconic images can exist both for generic classes of objects, and for specific

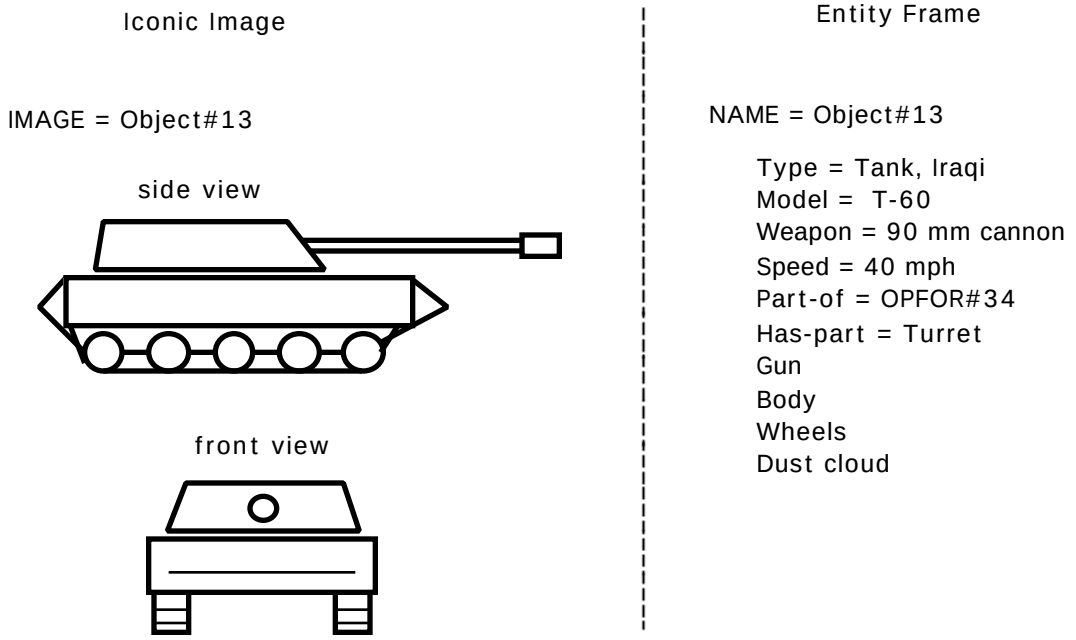


Figure 11. An image and entity frame representation of a tank. The image may be derived from a camera, or generated by a graphics engine from internal data. The entity frame provides a description of the entity that may include information that is inherited from a generic class and not available from sensory input. For example, the range of the weapon may be known from the class of the tank, but cannot be measured from sensory data.

instances of objects. A specific instance of an object can be classified as a member of a class by the attributes it shares in common with the generic class. Once classified, the specific object inherits the attributes of the class, possibly with some exceptions and unique attributes.

Df: **image**
a two-dimensional array of attribute values

Images may be generated in a number of ways. For example, an image may be formed by a map or sketch on a sheet of paper, or by the projection of electrons on the face of a cathode ray tube, or by the optical projection of light from a scene in the world through a lens onto an array of photoreceptors such as a CCD TV camera. An image may also consist of attributes such as spatial or temporal gradients, stereo disparity, range, or flow that are computed from other images over spatial windows and temporal intervals. An image may also be generated by internal mechanisms (such as a computer graphics engine) from information stored in symbolic entity frames.

There can be at least two different types of images:

1. Attribute images wherein each pixel defines a location that contains the value of an attribute.
2. Entity images wherein each pixel defines a location that contains the name of the entity to which it belongs.

Df: **map**
two-dimensional array of attributes that are registered with known locations in the world

Map pixel attributes may include icons or names of symbolic entities of which the pixel is a part.

Df: **pixel**
a picture element

A pixel is the smallest distinguishable region in an image (i.e., the region within a pixel has no internal structure). A pixel may have attributes, such as the output of a photoreceptor in a CCD camera that corresponds to brightness or color. Pixel attributes may also include spatial or temporal gradients of brightness, range, stereo disparity, image flow magnitude and direction, surface orientation, etc. integrated over the area of the pixel, or over a small neighborhood centered on the pixel.

Pixel attributes may also include the name of an entity that occurs within that pixel. For example, pixel attributes may include the name of an edge, surface, or object that lies within the pixel. For each attribute that a pixel can have, an attribute image can be constructed. Examples of attribute images are shown in Figure 12.

Df: **pixel attribute vector**
an ordered set of attributes of a pixel

Df: **entity image**
an attribute image consisting of pixels that are labeled with the names of the entities to which they belong (i.e., into which they are grouped according to some grouping criteria).

For each pixel in an entity image, the entity name is a pointer to an entity frame that contains the entity attributes. There are two ways in which the knowledge database can be structured to accomplish this. Figure 13a is an example of a knowledge database structured so that there is a single surface entity image for many surface entity frames. In this example, there are three surfaces in the image. At each pixel in the surface entity image there is a pointer that points to the appropriate surface entity frame. Figure 13b is an example of a knowledge database structured so that there is a separate surface entity image for each surface entity frame. Each surface entity image is uniquely related to a unique surface entity frame. In Figure 13b, each pixel in each surface entity frame needs only a binary value indicating whether that pixel is a member of the surface entity or not.

In either case, the entity image provides a list of all the pixels that belong to the entity frame. Thus, either type of entity image can be used as a mask or window for correlation or for integrating entity attributes. Pixel attributes integrated over the window provide entity attributes.

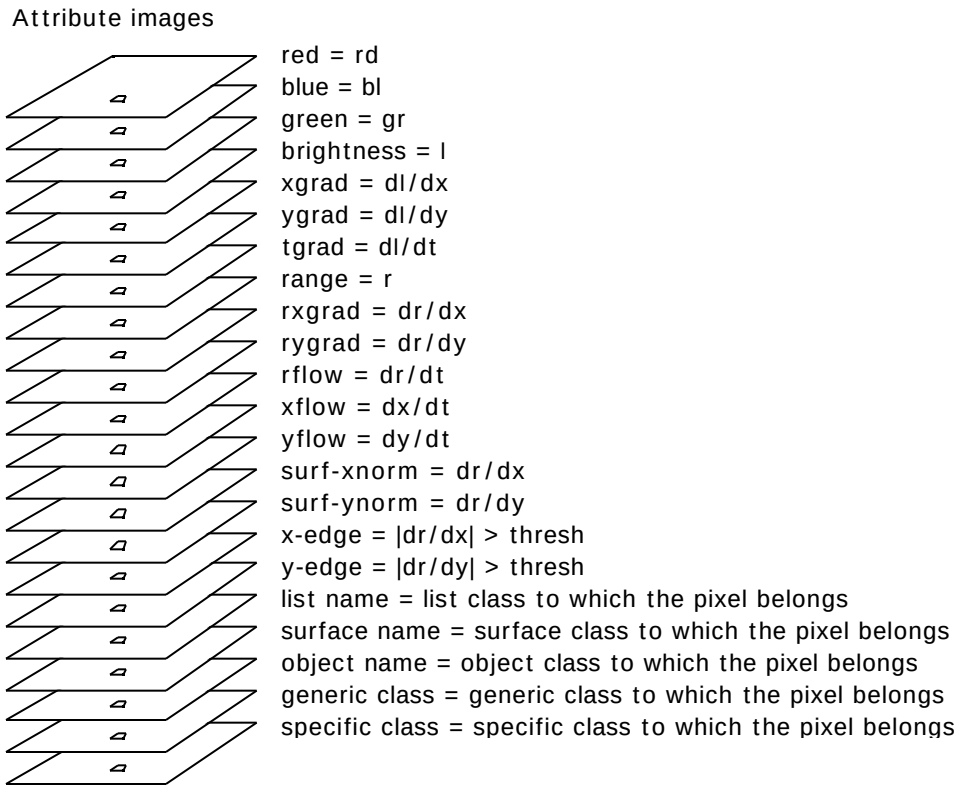
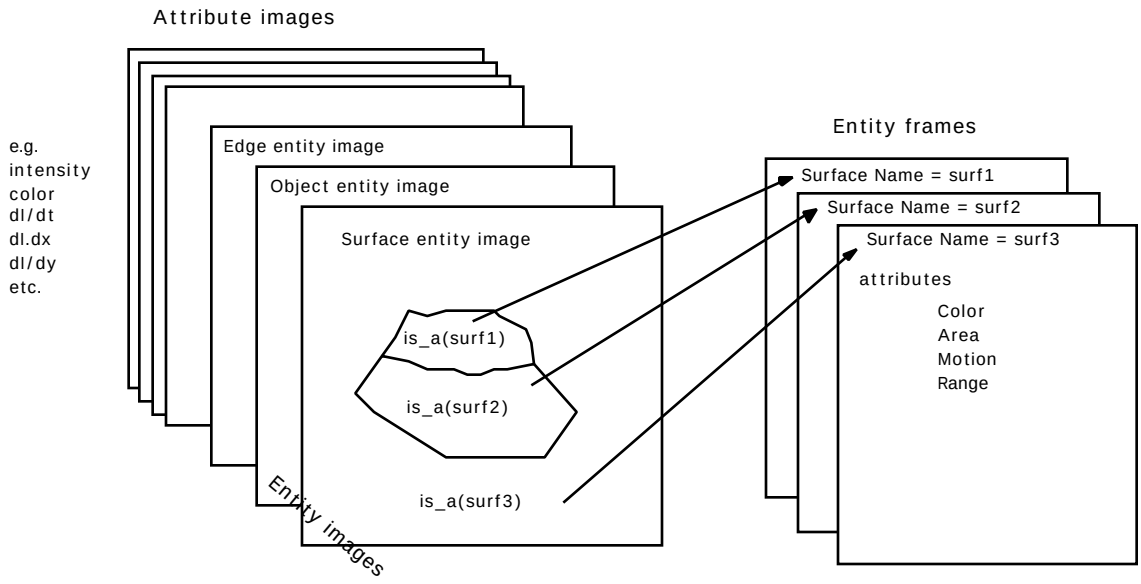
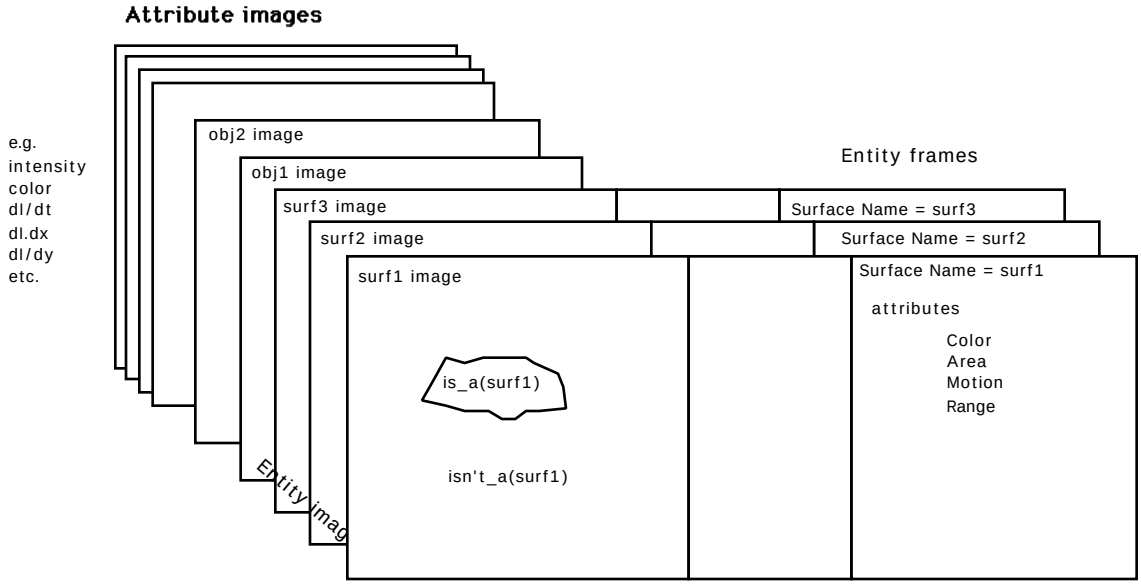


Figure 12. A set of attribute images. An image is a two-dimensional array of attribute values. The set of registered attribute images shown in this figure form a three-dimensional array of attribute values. A single pixel in each of the images corresponds to a column in the 3-D array. The set of attributes of the pixel column corresponds to a pixel attribute vector.

Entity attributes can be computed from pixel attributes only after the image is segmented and a window is defined over which the attributes can be integrated. For example, the length of a boundary, or the area, texture, shape, and size of a surface can be computed from an image only after the image is segmented into regions that belong to the entity and those that do not. Entity attributes are stored in entity frames. Attributes in an entity frame can be used to generate windows and masks for focusing attention in images, or for correlation, classification, and grouping. (This is discussed later in the section on Sensory Processing (SP) modules.)



(a)



(b)

Figure 13. Relationship between entity frames and attribute images. In (a), pointers in the pixels indicate which surface each pixel belongs to. In (b), each surface entity has a binary image that defines which pixels belong to it. In both cases, the entity frame contains a list of attributes of the entity, and the entity image comprises a list of the pixels that belong to the entity. Thus, the entity image can serve as a mask or window for processing the image. The labels in the entity image are pointers to the entity frame.

Examples of 4-D/RCS Entities

The collection of entities stored in long term memory are the set of entities that are relevant to the tasks that the system may be required to perform. Some object entities that are relevant to Demo III tasks are [ground], [sky], [hill crest], [ditch], [rock], [tree], [bush], [road], [water], [vehicle], [human], [animal]. Each of these objects has attributes and subentities such as surfaces, and edges that can be observed in, and computed from, the image.

For example, the ground consists of one or more surfaces, including the surface directly under the vehicle and extending outward from beneath the vehicle. Each of these surfaces has attributes of slope, roughness, and ground cover. Each surface has boundaries, or edges that may be fuzzy or sharp. The slope of the ground directly under the vehicle can be measured by a tilt meter in the vehicle. Directly in front of the vehicle, the slope of the ground can be measured by the LADAR camera, by vergence and stereo disparity, or by image flow calculations. Ground surfaces may blend smoothly into each other, or be separated by walls, embankments, fences, ditches, hill crests, or other discontinuities.

An entity frame for the ground might have the form:

name = ground
- *Subentities*
  - surface#1 (beneath the vehicle)
  - surface#2
  - surface#3
  - etc.
- *Criteria for recognition*
  - elevation = from straight down to near the horizon
  - dr/dy = roughly a horizontal surface

Each of the ground subentities has the form:

name = surface#1
- *Attributes*
  - slope
  - roughness
  - ground cover
  - shape
  - size
  - map position (in egosphere coordinates)
  - map position (in world coordinates)
  - map orientation (with respect to vehicle motion)
  - map orientation (in world coordinates)
- *Subentities*
  - boundaries
  - surface patches

name = surface#2
- *Attributes*
  - slope
  - roughness
  - ground cover
  - shape
  - size

map position (in egosphere coordinates)
map position (in world coordinates)
map orientation (with respect to vehicle motion)
map orientation (in world coordinates)
Subentities
boundaries
surface patches

name = surface#3
etc.

An entity frame for the sky might have the form:

name = sky
Attributes
color
cloud types
Criterion for recognition
elevation = above the horizon
range = infinite
color = blue or gray (by day), black (at night)
may contain clouds of various shapes

The criteria for recognition enable the SP modules to recognize regions in the image as belonging to the [sky] entity.

An entity frame for a hill crest or ditch might have the form:

name = hill crest
Attributes
dr/dy > threshold
elevation of discontinuity =
velocity of discontinuity =
Criteria for recognition
dr/dy = large
magnitude of dr/dy decreases as vehicle approaches
map position of dr/dy recedes as vehicle approaches

Both a [hill crest] entity and a [ditch] entity have an attribute of a large dr/dy discontinuity. The criteria for recognition enable the SP modules to distinguish a [hill crest] from a [ditch].

name = ditch
Attributes
dr/dy > threshold
elevation of discontinuity =
velocity of discontinuity =
depth =
slope of banks =
type of bottom = (water | rocks | dirt)
map position = (in egosphere coordinates)
map position = (in world coordinates)
map orientation of near edge = (with respect to vehicle motion)
map orientation of near edge = (in world coordinates)

Subentities
bank surfaces
bottom surface
near edge
far edge
etc.
Criteria for recognition
dr/dy = medium to large
elevation of dr/dy = near the minimum ground elevation
magnitude of dr/dy = constant as the vehicle approaches
map position of dr/dy remains fixed as the vehicle approaches

Entity frames for other entities relevant to an unmanned vehicle might have the form:

name = rock
Attributes
height
width
map position = (in egosphere coordinates)
map position = (in world coordinates)
shape =
color =
etc.
Subentities
surfaces
edges
Criteria for recognition
range texture
color

name = tree
Attributes
shape
size
diameter of trunk =
clearance under lowest branch =
map position = (in egosphere coordinates)
map position = (in world coordinates)
etc.
Subentities
trunk
branches
foliage

name = bush
Attributes
width =
height =
density =
map position = (in egosphere coordinates)
map position = (in world coordinates)
color =

Subentities
foliage

name = road
Attributes
surface roughness =
color =
width =
shape = (curved or straight, banked, etc.)
map position = (in egosphere coordinates)
map position = (in world coordinates)
map orientation = (with respect to vehicle motion)
map orientation = (in world coordinates)
Subentities
lanes
lane markings

name = water
Attributes
surface roughness =
width =
length =
slope of banks =
map position of nearest edge = (in egosphere coordinates)
map position of nearest edge = (in world coordinates)
map orientation of nearest edge = (with respect to vehicle motion)
map orientation of nearest edge = (in world coordinates)
etc.
Subentities
bank surfaces
water surface
near edge
far edge
etc.

name = vehicle
Attributes
height =
length =
width =
orientation =
color =
map position = (in egosphere coordinates)
map position = (in world coordinates)
velocity = (in egosphere coordinates)
velocity = (in world coordinates)
Subentities
surfaces

name = human
Attributes
height =
width =
orientation =

color =
map position = (in egosphere coordinates)
map position = (in world coordinates)
velocity = (in egosphere coordinates)
velocity = (in world coordinates)
Subentities
surfaces

name = animal
Attributes
height =
width =
orientation =
color =
map position = (in egosphere coordinates)
map position = (in world coordinates)
velocity = (in egosphere coordinates)
velocity = (in world coordinates)
Subentities
surfaces

Each of the attributes defined for the above entities can, in most cases, be computed from the image.

The above list is only an illustrative example. A complete knowledge database of entities and their attributes will be compiled as the Demo III project progresses.

Short- vs. Long-Term Memory

The 4-D/RCS knowledge database has two parts:
1. A short term memory containing both symbolic and iconic representations of those entities that are the subject of current attention.
2. A long term memory containing symbolic representations of all the entities, events, and rules that are known to the intelligent system, as well as topographical maps and iconic templates of selected objects.

1. Short-Term Memory

Short-term memory consists of data structures and memory management software and hardware that are specially designed to support real-time computation for planning, control, and interpretation of sensory input. Short-term memory is analogous to a computer cache memory. It is loaded with data that is relevant to the current task, and provides very efficient high speed service to support the computations required by that task. As soon as the current task is complete, or interrupted by a higher priority task, the short term memory is flushed, and new data is loaded to support the next task. Short-term memory contains images, maps, entity frames, and event frames.

Short-term symbolic entity frames represent current entities-of-attention. Short term entity and event frames contain pointers to other short term entity and event frames, to short term iconic images, and to long-term symbolic entity and event frames stored in long term memory.

Df: **entities-of-attention**
entities that have been specified by task decomposition as important by the task or have been flagged by sensory processing as particularly note-worthy

Entities-of-attention are loaded into short-term memory.

Short-term iconic images can consist of attributes generated directly, or by recursive estimation, from sensory observations, or can be generated internally from data resident in short term symbolic frames. Figure 12 is an example of a set of short-term iconic attribute images. Short-term iconic images can be used to filter, mask, or window incoming data, or to correlate incoming sensory observations with internally generated images. The observed and predicted attribute images and the error images shown in Figures 17 and 18 are examples of short term iconic images. Short-term iconic images persist in memory only so long as they are refreshed by incoming sensory data or by internally generated images.

Entity attributes can be used to recognize (or establish correspondence between entities-of-attention and) entities in long-term memory

The short-term KD is implemented in a distributed fashion with representations at each node supporting the task decomposition and sensory processing functions currently planned or being executed out in each node. The long-term KD may be implemented in either a distributed or a centralized database.

2. Long-Term Memory

Long-term memory contains the entire dictionary of entities that the intelligent system has information about. Typically, only entities at the level of object and higher are contained in long term memory. Long-term symbolic entity and event frames may be transferred into short-term memory, or vice versa. When a long-term symbolic entity is specified by a task command, that entity frame is transferred into the list of short-term entities-of-attention. Alternatively, if a match is recognized between a current entity-of-attention and a long-term symbolic entity, newly observed attributes from the short-term entity can be used to update the attributes of the long-term entity, and attributes of the long-term memory symbolic entity can be added to the short-term entity. If nothing in long-term memory is recognized as corresponding to what is observed, and if the observed entity is judged noteworthy by the Value Judgment function, then the short-term symbolic entity can be entered as a new entity into long-term memory.

Long-term memory consists mostly of symbolic data structures such as entity and event frames and their relational pointers, although it may contain some iconic templates such as illustrated in Figure 11 to be used for recognition. Long-term memory also contains digital terrain maps, with overlays for entities, objects, and terrain features such as ground cover, and traversibility. Terrain maps may be multiresolutional and maps that are related to a particular level in the hierarchy, such as the vehicle level, may be used to update the lower resolution maps of the higher levels. Map modifications may be shared with peer vehicles, or with human operators or battlefield digitization databases.

Df: **symbol grounding**
the establishment of correspondence between symbolic entities in the KD and physical entities in the external world

Df: **physical entity**
a group, object, surface, edge, boundary, vertex, region, or point that exists in the world

Symbol grounding is achieved by Sensory Processing.

Sensory Processing (SP) modules

Within each node of the 4-D/RCS architecture, Sensory Processing (SP) modules process data from sensors to extract information about the world. The goal of the SP modules is to compute the information needed to maintain the knowledge database (KD) as a current and accurate estimate of the state of the world, including the state of the vehicle itself.

Sensory processing is organized around sensors. Sensors provide input signals that are windowed, filtered, recognized, and grouped into entities, events, and situations in a world model knowledge database that correspond in a meaningful and useful way to entities, events, and situations in the real world. Each sensor produces a signal that varies in time as the physical phenomena that it measures varies in time.

An intelligent vehicle may have many sensors, including visual, infrared, radar, laser ranging, acoustic, vibration, sonar, tactile, position, velocity, acceleration, force, torque, temperature, pressure, magnetic, electrical, nuclear radiation, and chemical sensors. These sensors may be grouped into a variety of sensory subsystems.

The vision subsystem may include black and white, or color TV cameras, FLIRs, and laser range imaging (LADAR) cameras. SP modules in the visual subsystems may process images to compute brightness, color, spatial and temporal gradients, stereo disparity, range, texture, shape, and motion. This information may be used to detect edges, boundaries, and measure the shape, position, orientation, of surfaces. The objective may be to detect obstacles, to track targets, and to recognize objects, events and situations. Camera platform encoders may measure the pointing direction of vision sensors relative to the vehicle, and inertial sensors may enable the vision system to stabilize images and determine absolute camera pointing direction and tracking velocity. This enables the SP modules to determine the position and motion of objects in the world. Characteristic size, shape, color, thermal signature, and motion enables the recognition of objects, such as roads, ditches, fences, rocks, trees, bushes, dirt, sand, mud, concertino wire, fire, smoke, buildings, bodies of water, tanks, trucks, and people.

The acoustic subsystem may include microphone arrays and sonar sensors. SP modules in the acoustic subsystem may process signals to detect frequency components, compare time and frequency patterns with predicted acoustic entities, measure correlations and differences, close phase-lock loops and tracking filters. This information may be used to track and recognize acoustic signatures from sources such as helicopters, jets, trucks, tanks, sniper fire, machine guns, heavy weapons, exploding ordinance, and sounds of footsteps.

The mobility subsystem may include accelerometers, tilt meters, gyros, global positioning satellite (GPS) receivers, odometers and speedometers to estimate position, velocity, and heading. Internal vehicle sensors may provide information about fuel levels, engine temperature, rpm, oil pressure, and vibration.

Each of these sensory subsystems has its own SP, WM, and VJ modules that extract and evaluate information from the sensory data stream that is relevant to the behavioral task being planned and executed in the TD modules. As sensory data is processed, it is filtered, matched against known classes, windowed, and combined, or grouped, with data from other sensors into entities. In the process, signals are transformed into symbols, and images are transformed into maps with identified regions that correspond to entities and

classes in the world. These symbols and maps represent the information needed by the TD modules to plan and execute behavior that maximizes the intelligent system's probability of success in accomplishing its behavioral goals. This is illustrated in Figure 14.

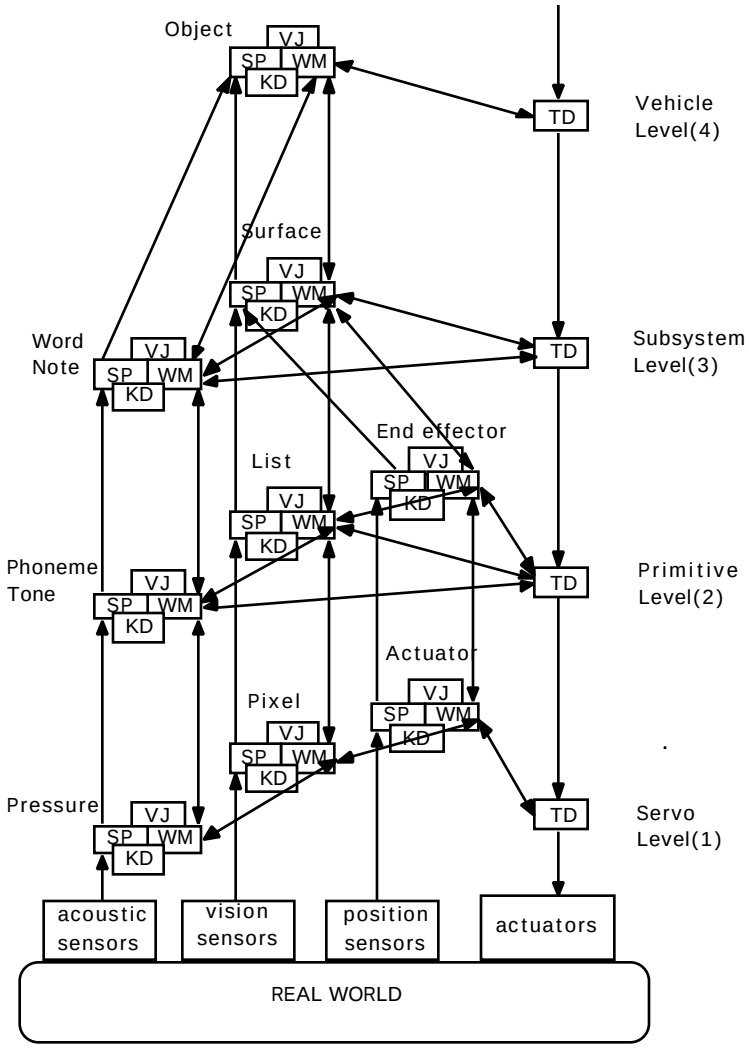


Figure 14. Three SP/WM/VJ/KD hierarchies that provide information about the world to a TD hierarchy. At each successively higher level, information from different sensory inputs is integrated and grouped into higher level more global entities and situations. Note that the acoustic and vision systems do not interact with the TD hierarchy at the Servo level.

In all subsystems and at all levels of the 4-D/RCS hierarchy, SP modules compare observed data from sensors with predicted data from the world model. Within each node of the 4-D/RCS architecture, interactions between WM and SP modules can generate a variety of predictive filtering, detection, and model matching processes, such as Kalman filtering and recursive estimation. Interactions between SP and WM modules can also enable recognition, segmentation, scene understanding, and learning.

The interaction between the SP and WM modules maintains a correspondence between the state of the world and the knowledge in the Knowledge Database (KD). SP modules process input from sensors and combine this input with knowledge already in the KD. Therefore, the knowledge in the KD at time k is a combination of knowledge at time k-1, plus observations at time k from sensors and control signals sent to the output. This can be expressed as

$$\text{Knowledge}(k) = \text{Knowledge}(k-1) + \text{Observation}(k)$$

where

k is the time index

and

Observation(k) = Sensory input(k) + Control output(k)

The KD is thus a product of a priori information, plus the history of all past observations, including the string of tasks and commands that brought the system to its current state, plus the current sensory input and the current task. It is this real-time combination of past and present information that allows the system to behave in an intelligent manner. At each computational cycle, sensory input is processed in the context of the current task and the current knowledge database. Models of the world stored in the KD provide the basis for Kalman filtering of noisy data. Knowledge in the KD provides information for directing attention and masking, or windowing, input from sensors. SP modules compare what is observed by sensors with what is expected by the WM. Good correlation confirms what is predicted, and variance can be used to update information that already exists in the KD. Poor correlation contradicts what is predicted, and causes the system to reconsider its predictions.

The sum total of the knowledge in the KD enables the WM modules to provide the TD modules with the information needed to plan and control behavior. Using knowledge stored in the KD, the TD modules can plan for the future and extrapolate through periods when sensory data is unavailable or unreliable.

Hypothesis and Test

Interaction between the SP and WM modules is a form of hypothesize and test as shown in Figure 15. The WM module hypothesizes a prediction based on the set of estimated state variables that reside in the knowledge database. WM predictions are compared in the SP modules with sensory observations. At the lowest level, sensory observations come directly from sensors. At higher levels, sensory observations come from lower level SP modules. Correlations between sensory observations and WM predictions indicate the degree to which the predictions are correct. Differences (or correlation offsets) indicate the error between the prediction and observation. If correlation exceeds threshold, the hypothesis is verified, or confirmed. When the hypothesis is verified, the focus-of-attention can be narrowed, and residual difference values can be used to update the estimated state variables to improve the hypothesis. This is a recursive estimation process that performs predictive filtering and temporal integration. On the other hand, if correlation falls below threshold, the hypothesis is rejected. In this case, the focus-of-attention is widened, and a new hypothesis is generated.

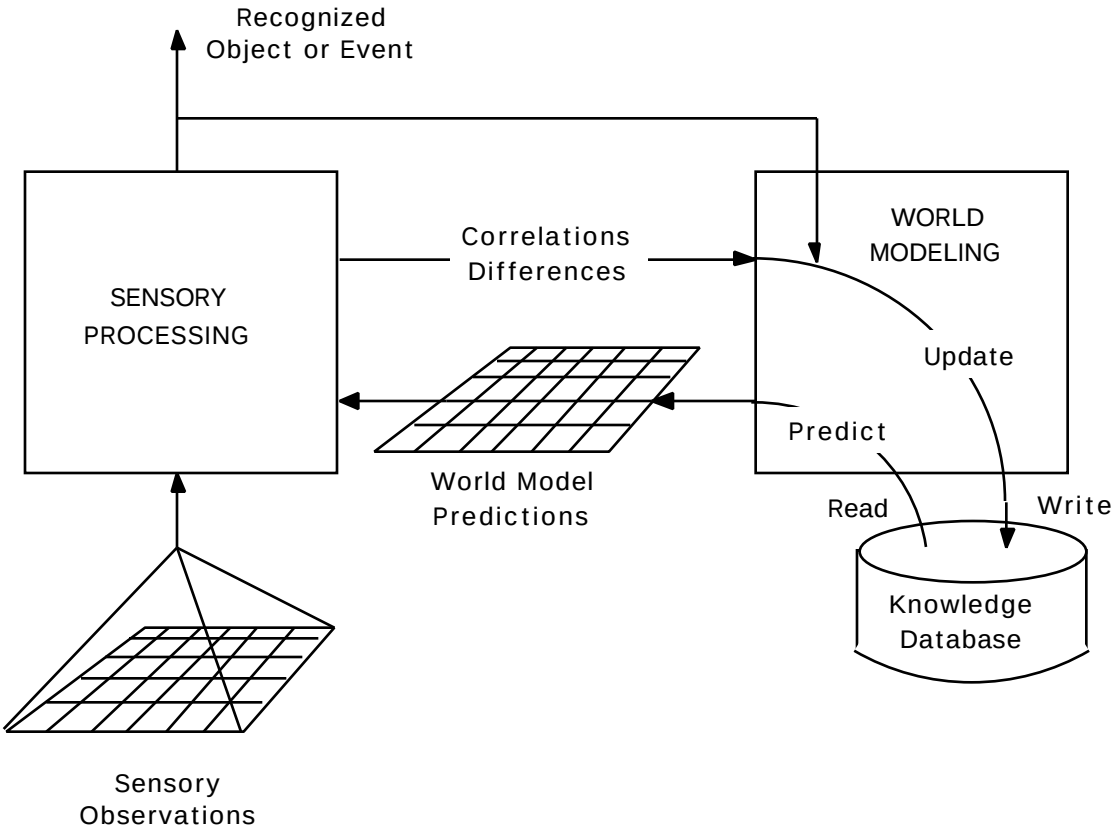


Figure 15. Interaction between sensory processing (SP) and world modeling (WM). SP modules compare sensory observations from sensors or from lower level SP modules with predictions generated by WM modules from knowledge stored in the knowledge database. Differences are used to update state variables in the KD. Correlations that exceed threshold verify the hypotheses used to generate predictions. Hypothesis verification may indicate recognition of entities or detection of events.

Tracking

Hypothesize and test is the fundamental paradigm of the 4-D approach pioneered by Dickmanns and Graefe and refined by Dickmanns et al over the past eight years. [Dickmanns 95, Dickmanns 94, Dickmanns 92a, Dickmanns 92b, Dickmanns and Graefe 88]. Figure 15 shows the resulting coarse overall block diagram of a vision system based on these principles. Dickmanns [92a] explains Figure 16 as follows:

> At the upper left, the real world is shown by a block; control inputs to the self vehicle may lead to changes in the visual appearance of the world either by changing the viewing direction or through egomotion. The continuous changes of objects and their relative position in the world over time are sensed by CCD-sensor arrays (shown as converging lines to the lower center, symbolizing the 3D to 2D data reduction). They record the incoming light intensity from a certain field of view at a fixed sampling rate. By this imaging process the information flow is discretized in two ways: There is a limited spatial resolution in the image plane determined by the pixel spacing in the camera, and a temporal discretization determined by the scan rate of the camera.

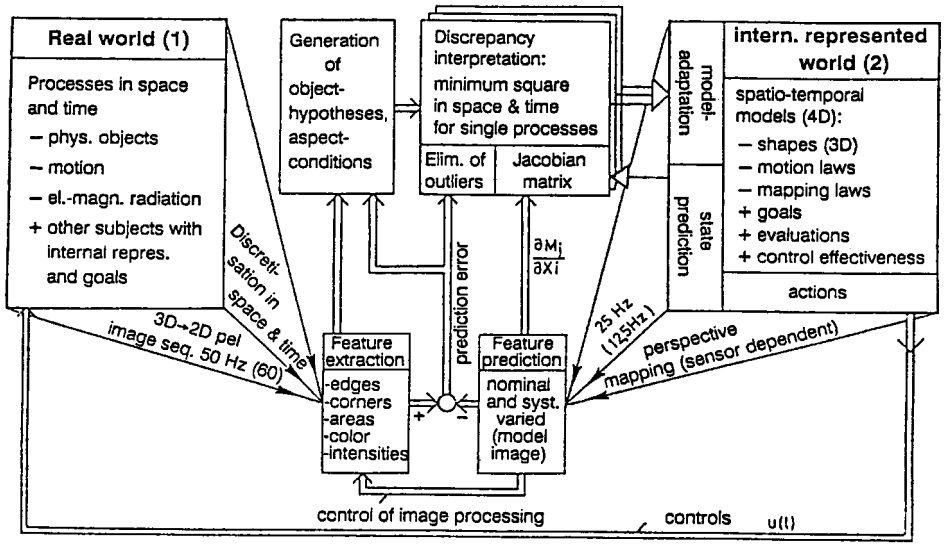


Figure 16. Basic scheme for 4-D image sequence understanding by prediction error minimization. [from Dickmanns 92a]

Instead of trying to invert this image sequence for 3D scene understanding, a different approach of analysis through synthesis has been selected, taking advantage of the available recursive estimation scheme after Kalman. From previous experience, generic models of objects in the 3D-world are known in the interpretation process. This comprises both 3D shape, recognizable by certain feature aggregations given the aspect conditions, and motion behavior over time. In an initialization phase, starting from a collection of features extracted by low level image processing (lower center left in fig. 15), object hypotheses including the aspect conditions and the motion behavior (transition matrices) in space have to be generated (upper center left in Figure 15). They are installed in an internal 'mental' world representation intended to duplicate the outside real world. This is sometimes called 'world 2', as opposed to the real 'world 1'.

Once an aggregation of objects has been instantiated in the world 2, exploiting the dynamical models for those objects allows the prediction of object states for that point in time when the next measurements are going to be taken. By applying the forward perspective projection to those features which will be well visible, using the same mapping conditions as in the CCD sensor, a model image can be generated which should duplicate the measured image if the situation has been understood properly. The situation is thus 'imagined' (right and lower center right in fig. 15). The big advantage of this approach is that due to the internal 4D-model not only the actual situation at the present time but also the Jacobian matrix of the feature positions and orientations with respect to all state component changes can be determined (upper block in center right, lower right corner). This need not necessarily be done by analytical means but maybe achieved by numerical differentiation exploiting the mapping subroutines already implemented for the nominal case.

This rich information is used for bypassing the perspective inversion via recursive least squares filtering through feedback of the prediction errors of the features. More details are available in [Dickmans and Graefe 88].

A more mathematical representation of the recursive estimation loop between the SP and WM modules is shown in Figure 17. In this figure, the entity attributes in the entity frame

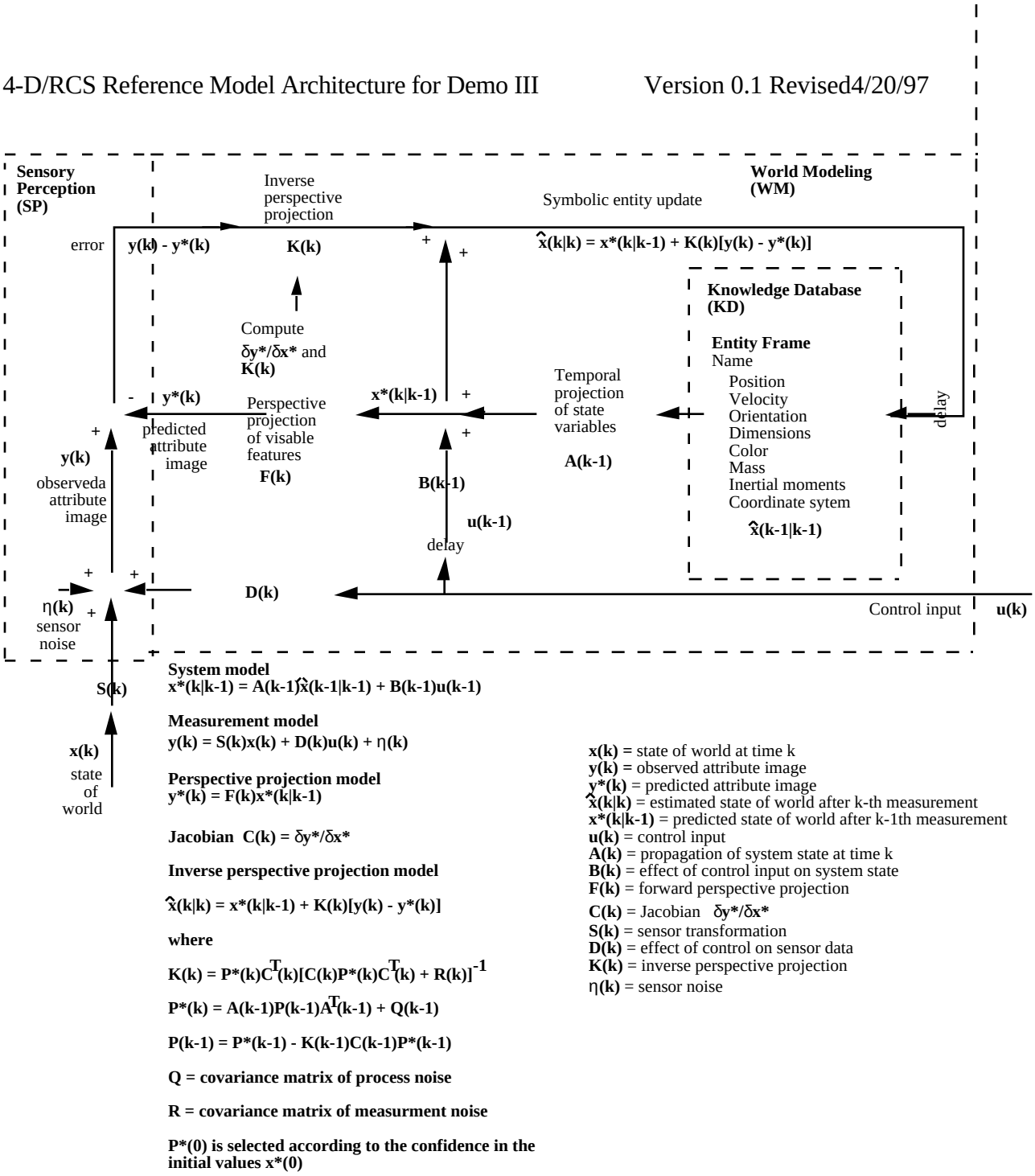


Figure 17. A recursive estimation loop. This loop updates entity attributes in the WM knowledge database (KD) based on differences between predicted images and observed images computed in the SP modules.

format are shown at the right. This is the estimate at time k-1 after the measurement at k-1. This estimate is projected forward in time by the matrix A(k-1), and combined with the effect of the control input u(k-1) to provide a prediction of the entity attributes at time k, based on information from measurements up to and including time k-1. The entity attributes are then transformed into image coordinates through the perspective projection F(k), resulting in a predicted attribute image y*(k). This is compared with the observed attribute image y(k). This comparison may consist of correlation or difference operations.

The result is an error signal y(k) - y*(k). This error is then transformed back into attribute coordinates through the inverse Jacobian K(k). The forward Jacobian can be computed numerically from the forward perspective projection F(k) as described by Dickmanns. The transformed error is then added to the predicted entity attribute vector x*(k|k-1) to produce the new estimated entity attribute vector x*(k|k). The text included in Figure 16 contains mathematical expressions for the system model, the measurement model, and the forward and inverse perspective projection models.

Recognition

A recursive estimation loop requires initialization. The initial choice of entity frames, and initialization of attributes within the entity frames is a recognition problem.

Df: **recognition**
the establishment of a match between an observed entity in the image and a stored entity in the KD

When the attributes of an observed entity correlate with those of a stored entity, the observed entity can be said to match, or be recognized as belonging to the class represented by, the stored entity.

When the attributes of an observed entity fail to correlate with those of a stored entity, the observed entity does not belong to the class represented by the stored entity. In this case, recognition does not occur.

There are at least five degrees of recognition:

First, there is the recognition that a pattern of attributes in a signal or an image matches the attributes of a topological entity at a point in time. This is achieved when the recognition criteria for a topological entity are met.

Second, there is confirmation that the pattern in the signal continues to match the single topological attributes over an extended period of time. This is achieved when the hypothesize and test process of recursive estimation is successful.

Third, there is the confirmation that the signal pattern matches topological entities at all levels in the topological hierarchy simultaneously over an extended period of time. This is achieved when the hypothesize and test process is successful at all hierarchical level simultaneously.

Fourth, there is the recognition that the attributes of the topological object match the attributes of a generic object class over an extended period of time. This is achieved when the hypothesize and test process is successful between the attributes of the topological object and the typical attributes of a generic object class.

Fifth, there is the confirmation that the attributes of a generic object match the attributes of a specific object over an extended period of time. This is achieved when the hypothesize and test process is successful between the attributes of the generic object and the attributes of a specific object.

Recognition is not as time critical as the update rate of the tracking and recursive estimation loop. However, the recognition problem is typically more difficult and less well defined than the tracking problem. In fact, recognition is largely an unsolved problem except for a few special and rather simple cases. In practice, the initialization/recognition problem for

recursive estimation in vision is often solved by a human operator. For example, a human operator may indicate what entity models appear as entities in the image, where they are located, and how they are oriented [Schneiderman and Nashman]. In 4-D/RCS, the recognition problem will be restricted to a limited set of objects that are critical to the vehicle's mission.

Hierarchy of Perception

Sensory processing (SP), world modeling (WM), and value judgement (VJ) are hierarchical processes, and KD is a hierarchical knowledge database that in many ways mirror the task decomposition (TD) hierarchy. While TD/WM/VJ/KD generate behavior by decomposing tasks into subtasks for a number of agents over an extended period of time in the future, SP/WM/VJ/KD integrate information from a number of sensors over an extended period of time in the past. While computations in the TD hierarchy are primarily top-down, decomposing high level conceptual representations of goals and tasks into low level actions by a large number of actuators, computations in the SP hierarchy are primarily bottom-up, integrating low level signals from a multitude of sensors into high level perceptions of situations and concepts.

There are important bottom-up inputs to TD/WM from SP that enable reactive or reflexive behavior. Similarly, there are important top-down inputs to SP/WM from TD that enable focusing of attention, masking of unimportant signals, and selective application of sensory processing algorithms. Just as there are different kinds of planning and control requirements at different levels of the TD/WM/VJ hierarchy, so there are different kinds of computational algorithms and data representations at different levels of the SP/WM/VJ hierarchy. Just as the TD hierarchy is a hierarchy of tasks, the SP/WM/KD hierarchy is a hierarchy of entity classes.

Definition of Levels

The hierarchical layers in the SP/WM/KD hierarchy are intimately related to, and to a large extent defined by, the classes of entities in the knowledge database that are required by the planning and control execution modules at various layers in the TD hierarchy. The SP, WM, KD, VJ, and TD hierarchies are interconnected in the nodes of the 4-D/RCS hierarchy. Each node is built around a TD module, with the SP, WM, KD, and VJ functions necessary to support that TD module. In general, the entity classes that are required in KD (and hence in SP and WM) to support TD functionality at each level are:

Level 1 -- Pixel Entity Classes
The input to level 1 of the SP hierarchy is generated by sensors that detect energy derived from force, torque, acceleration, velocity, position, illumination, reflectance, temperature, and acoustic signature of entities in the environment, such as actuators, tools, materials, objects, interacting in dynamic relationships and situations in the world. Each sensor measures some state, attribute, or condition of the world and the variety of entities in it. For example, encoders measure position, tachometers measure velocity, accelerometers measure acceleration, etc. Each photodector in a CCD camera measure visible radiation imaged on it from a pixel sized region of space. Each photodetector in a FLIR camera measures infrared radiation. Typically, each sensor integrates energy over an exposure time interval.

At level 1, sensory data from each individual sensor may be processed and analyzed entirely with respect to itself and its immediate neighborhood in space and time. Thus, level 1 of the SP hierarchy is the level of pixel entity classes.

Level 2 -- List Entity Classes
At level 2 of the SP hierarchy, lists of a few (10 plus or minus 9) spatially contiguous pixels with attributes that fit some list entity model, or temporal strings of attributes from a single sensor that fit some list entity model, can be grouped, processed, and analyzed as units called list entities. A list entity is thus defined as a list, or group, of points in space and/or time. For example, a list of pixels with contiguous brightness or color gradients might be grouped, processed, and analyzed as an edge, vertex, or surface patch. A temporal signal, or string of intensity attributes, from an acoustic sensor might be grouped as a tone, a frequency, or a phoneme. Attributes of the pixels or strings comprising each list are combined into list entity attributes. Thus, level 2 of the SP hierarchy is the level of list entity classes.

Level 3 -- Surface Entity Classes
At level 3 of the SP hierarchy, sets of spatially contiguous list entities with range, motion, and other attributes that fit some surface entity model, or temporal strings of list attributes from a set of frequency filters that fit some surface entity model, can be grouped, processed, and analyzed as units called surface entities[3]. For example, a group of contiguous surface patch list entities with similar range and motion might be grouped, processed, and analyzed as a spatial surface entity. A spatial string of edge and vertex entities might be grouped, processed, and analyzed as a spatial surface boundary entity. A temporal string of phonemes might be grouped as a word, or a temporal string of outputs from a set of frequency filters as a note or phrase in a song. Attributes of lists entities comprising each surface entity are combined into surface entity attributes. Thus, level 3 of the SP hierarchy is the level of surface entity classes.

Level 4 -- Object Entity Classes
At level 4 of the SP hierarchy, sets of spatially contiguous surface entities with attributes such as range, motion, orientation, color, and texture that fit some object entity model, or temporal strings of surface attributes from a single surface entity that fit some object entity model, can be grouped, processed, and analyzed as units called object entities. For example, a group of surfaces with coincident boundaries and similar or smoothly varying range and velocity attributes might be grouped into an object such as a building, a vehicle, or a tree. A string of words might be grouped into a sentence, or a string of song phrases into a song. Attributes of surface entities comprising each object entity are combined into object entity attributes. Thus, level 4 of the SP hierarchy is the level of object entity classes.

Level 5 -- Squad Entity Classes
At level 5 of the SP hierarchy, sets of spatially related object entities with similar range, motion, and other attributes, or temporal strings of object attributes from a single object entity, can be grouped, processed, and analyzed as a group, collection, or assemblage of objects. In 4-D/RCS, the smallest group is called a squad. A string of sentences, or string of songs can be grouped into a paragraph, or a vocal performance. Attributes of object entities comprising each squad entity are combined into squad entity attributes. Thus, level 5 of the SP hierarchy is the level of squad entity classes.

[3] A surface entity may consist of a two dimensional surface in space, or a two dimensional surface in frequency and time. For example, a sonogram is a two dimensional representation of frequency vs. time.

Processing at Each Level

At each level of the SP hierarchy, there are six basic processing procedures:

1. Windowing (or its inverse, masking) -- selects the regions of space and/or time to be considered. The shape, position, and duration of spatial and temporal windows are determined by the shape, position, and duration of regions in an image, or in the world, that are labeled as worthy of attention. Regions may be worthy of attention either because they are goal related, or because they exhibit properties that are unexpected or dangerous, or because they are otherwise noteworthy. The size of each windows is determined by the level of confidence in the position and velocity attributes of the estimated entity defining the window.

2. Grouping -- integrates or organizes spatially and temporally contiguous subentities with attributes that fit some entity model into entities. The grouping process segments, or partitions, the image into topological regions with entity labels, or names. Any particular grouping is a hypothesis based on gestalt heuristics. A grouping hypothesis is confirmed or rejected based on the confidence factor in the estimated entities resulting from the grouping.

3. Measurement or computation -- measures or computes observed attribute values of entities generated by the grouping hypothesis.

4. Filtering (e.g., by recursive estimation) -- computes a best estimate (over a window of space and time) of entity attribute values based on correlation and difference between predicted and observed entity attribute values. Filtering also computes statistical properties such as confidence factors for observed and estimated attribute values.

5. Recursive estimation filtering requires prediction of the most likely next value for each attribute. Prediction is based on estimated dynamic attributes, planned results from behavior generation, and predicted dynamic attributes of higher level parent entities.

6. Recognition (classification, or detection) -- establishes a correlation or match between estimated attributes of entities observed in the world and attributes of entity classes stored in the system's knowledge database. Both generic and specific entity classes may be stored in the knowledge database, and observed entities may be recognized as belonging to either generic or specific object classes, or both.

WHAT and WHERE Channels

At each hierarchical level, there are two channels of information:

1. A WHERE channel that computes estimated attributes of position and motion of topological entities in the image, or in the external world. Estimated position and motion attributes are used to predict where to expect entities to appear next in the image, or in the external world.

2. A WHAT channel that computes attributes that describe distinguishing characteristics of entities in the image (or in the world), such as brightness, color, size, shape, texture, sound, taste, feel, or smell.

Both WHERE and WHAT attributes can be used to identify, recognize, classify, or name entities in the image, and by projection, in the external world.

Both WHERE and WHAT channels also provide information that is useful in windowing, grouping, and segmentation. WHERE information may be used to group subentities into entities (or regions) based on how subentities are connected, or how they move together in the image. WHAT information may be used to group subentities based on similarity of characteristics or attributes.

Pyramids of scale

At each level of the SP hierarchy, any attribute image may be represented simultaneously at several different scales (or resolutions) in a pyramid of images. The scale or resolution of an image is determined by the size of the spatial and temporal sample collected by each pixel. For any image, a lower resolution image may be derived by subsampling, or by averaging attributes from several higher resolution pixels, to compute an attribute for each lower resolution pixel. This process can be repeated to generate several levels of lower resolution. Each higher level in the pyramid is lower in resolution by a geometric progression that is determined by the ratio of subsampling (i.e., the number of pixels at each level that are averaged at each level to yield a pixel at the next lower resolution.)

In 4-D/RCS, the hierarchy of resolutions provided by pyramid processing is orthogonal to the hierarchy of entity classes and tasks that define levels in the 4-D/RCS system. This means that each image at each level in the hierarchy shown in Figure 2 can have its own pyramid of resolution. Typically, attribute images in the 4-D/RCS architecture are represented at four levels of resolution separated by a factor of two between levels of resolution. This means that each attribute image is represented as a 256x256 pixel image, a 128x128 pixel image, a 64x64 pixel image, and a 32x32 pixel image.

The utility of pyramids of scale is primarily a reduction in the computational expense of correlating two images. To compute a correlation value between a predicted and observed image, the value of each pixel attribute in the predicted image must be multiplied by the value of the corresponding pixel attribute in the observed image and the results summed over the correlation window. To compute the correlation function, a correlation value must be computed for every possible offset between predicted and observed images. The number of calculations required to find the correlation function therefore increases as the product of the number of possible offsets along each degree of freedom multiplied by the size of the correlation window all raised to the power of the number of degrees of freedom. This computational burden can be minimized by reducing the resolution, making the correlation window small, or reducing the number of degrees of freedom.

Pyramid processing minimizes both the resolution and the size of the correlation window at each level of the pyramid, yet maintains high resolution in the resulting correlation function by successive approximations at several levels of scale. In pyramid processing, correlation is performed first at the lowest resolution of the pyramid, and for only zero and plus or minus one pixel offset in x and y directions (a total of 9 offsets). The resulting low resolution correlation function can be used to make a course correction of the correlation offset. Correlation can then be performed at a higher resolution scale, again for only zero and plus or minus one pixel offset. A second more refined correction can then be made. This process can be repeated at each successively finer scale of the pyramid, until the finest scale is reached. This is a form of binary search for the correlation offset peak. By this procedure, the number of computations required to compute the correlation function is

significantly reduced. Correlation by pyramid processing requires only the logarithm[4] of the number of computations required by correlation at the finest scale.

The peak of the correlation function indicates how much the predicted and observed images are shifted relative to each other along each axis of freedom. The amount of shift divided by the temporal separation between the two images provides a measure of the image flow rate along each axis of freedom, i.e.,

$$dx/dt = \Delta x / \Delta t$$
$$dy/dt = \Delta y / \Delta t$$

where dx/dt is the image flow rate along the x-axis
dy/dt is the image flow rate along the y-axis
Δx = correlation peak offset along the x-axis
Δy = correlation peak offset along the y-axis
Δt = temporal separation between images

Pyramid processing techniques have been extensively studied at the David Sarnoff Research Laboratory by Dr. Peter Burt and his associates. The Sarnoff team has developed algorithms and special purpose computing hardware (VFE) to implement pyramid processing for area based correlation in real time. The hardware and software techniques developed by the Sarnoff team vastly improve the speed and efficiency of the computation of correlation and difference values over what can be achieved by conventional methods. The pyramid processing algorithms also provide means for widening and narrowing the focus of attention, and for electronically stabilizing images. [Burt and Adelson 83, Burt 88, Burt et al. 89, Burt and van der Val 90]

The 4-D/RCS architecture integrates the 4-D approach to image processing of Dickmanns et al with the pyramid processing techniques and VFE hardware of Burt et al. within the framework of the RCS reference model architecture developed at NIST.

An Example of Image Sensory Processing and World Modeling in 4-D/RCS

Figure 18 illustrates how signals from cameras and other sensors can be processed and transformed into knowledge that can be used to generate intelligent behavior. At the bottom left of Figure 18, a set of cameras observe a scene in the real world and a set of sensors measure parameters such as position, force, velocity, and acceleration. At each level in the SP hierarchy, each of the six basic procedures of sensory processing (1-windowing, 2-grouping, 3-measurement or computation, 4-filtering, 5- prediction, and 6-recognition) are performed.

Level 1 -- Pixel Entities

Level 1 Windowing
At level 1, windowing defines the region of space that is sampled by each sensor, the duration over which a signal from each sensor is integrated, and how often each sensor is

[4] The base of the logarithm is the ratio of resolutions at successive levels of the pyramid.

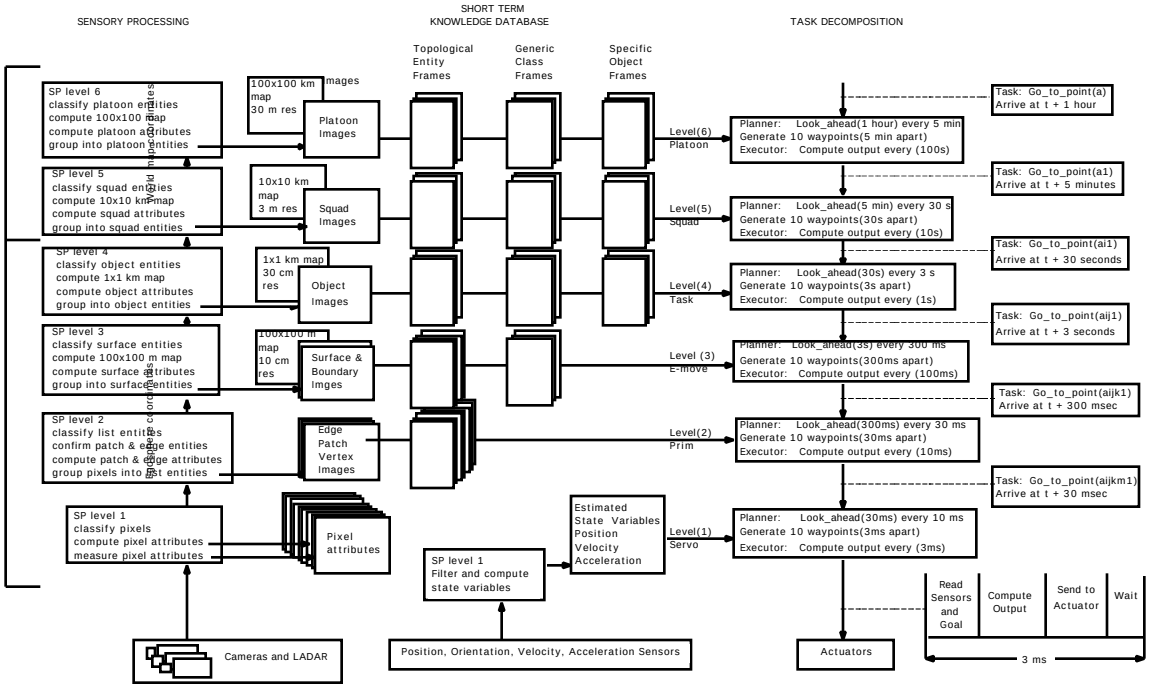


Figure 18. The 4-D/RCS hierarchy of image processing, short term knowledge database, and task decomposition.

sampled. For vision, windowing includes pointing and focusing the camera system to center the highest resolution portion of the photodetector array on the region of the egosphere that is most worthy of attention.

For example, in 4-D/RCS, there are three cameras with three different resolutions. The pan/tilt system moves the camera system to center the highest resolution camera on the most important region in the image while keeping the lower resolution cameras centered around the highest resolution camera. The camera control system functions to track the highest priority entity of attention, or to saccade quickly from one high priority entity to the next as different entities achieve the status of highest priority.

For acoustic or microwave sensors, windowing may be performed by pointing directional microphones or antennae at high priority targets.

For position, orientation, velocity, acceleration, and force sensors, windowing consists of selecting those signals that are relevant to the current servo task command for each actuator.

Level 1 Grouping
At level 1, grouping simply means that each sensor integrates, or groups, all the energy impinging upon it during a sample interval.

Level 1 Measurement
A number of attributes can be directly measured for each pixel. In a black and white TV camera, the intensity (I) of radiation at each pixel is measured. In a color TV camera, the intensity of red, blue, and green light (rd, gn, bl) at each pixel is measured. In a laser range imager (LADAR), or in an imaging radar, a value of range (r) is measured at each

pixel. For a FLIR, the infrared radiation (Ir) at each pixel is measured. A number of cameras can produce a number of images of different type, that may have different resolution and field of view. Pairs of cameras with overlapping fields of view may be displaced so as to provide stereo image pairs.

Level 1 Computation
Additional attributes for each pixel can be computed from measured attributes. For example, the x- and y-intensity gradients (dI/dx, dI/dy) may be computed at each pixel by calculating the difference in intensity between adjacent pixels in the x- and y-directions, or by applying a gradient operator such as a Sobel operator on a neighborhood about a pixel. The temporal intensity gradient (dI/dt) can be computed by subtracting the intensity of a pixel at time t-1 from the intensity at time t, or by applying a temporal gradient operator. Differences in range at adjacent pixels can be used to compute spatial range gradients (dr/dx, dr/dy) and surface roughness, or texture (tx). Spatial range gradients can also be computed from shading (dI/dx, dI/dy) when there is knowledge of the incidence of illumination. Temporal differences in range at a point can yield range rate (dr/dt).

For a pair of stereo images, disparity can be computed at each pixel by a number of algorithms. When combined with camera vergence and knowledge of camera spacing, disparity can be used to compute range for each pixel at every point in time.

In cases where the camera image is stabilized (or camera rotation is precisely known), spatial and temporal gradients can be combined with knowledge of camera motion derived from inertial and speedometer measurements to compute image flow vectors (dx/dt, dy/dt) at each pixel (except where the spatial gradient is zero. In this case, the flow rate sometimes can be inferred from the flow of surrounding pixels.) For stationary objects, time to contact (ttc) can be computed from image flow. Under certain conditions, range at each pixel can also be computed from image flow.

The set of attribute images that may be measured or computed at level 1 can be seen in Figure 18. A set of attribute images may be computed for each camera, and four or more resolution levels for each image may be maintained for pyramid processing. Higher derivatives may also be computed at each pixel. Based on Konderick's analysis, Kent and Luck have suggested that up to fourth order derivatives should be computed at each pixel. In 4-D/RCS, only first order derivatives will be computed, albeit at four different resolutions.

Level 1 Filtering
Each measured or computed attribute image can be filtered by a recursive estimation process that generates an estimated attribute image. Figure 18 illustrates the recursive estimation loop.

Each observed (i.e., measured or computed) attribute image is compared with a predicted attribute image. The prediction process derives information from two sources:
1) estimated image flow rates ($\frac{dx}{dt}$, $\frac{dy}{dt}$) and range rate ($\frac{dr}{dt}$) that describe the motion of entities in the image.
2) commanded camera motions and other actions that affect motion of entities in the image

The computation of image flow at a single pixel at a single point in time is notoriously unreliable and noisy. However, filtering over an interval in time by recursive estimation has been shown to provide considerable improvement in signal-to-noise ratio. Integration over a group of points comprising an entity further improves the signal-to-noise. The computation of image flow from a combination of entities at several different levels, each of which is recursively estimated, can provide a robust estimate of image flow at each pixel. This is

illustrated in Figure 18 where the estimated flow rates $\underline{dx/dt}$ and $\underline{dy/dt}$ are computed from a combination of estimates, including:

1. from the correlation offset between the predicted attribute image and the observed attribute image at each pixel
2. from the list entity flow estimate for the list entity to which the pixel belongs
3. from the surface entity flow estimate for the surface entity to which the pixel belongs
4. from the object entity flow estimate for the object entity to which the pixel belongs

In equation form this can be expressed as:

$$\underline{dx/dt}(x,y,t\text{-}1) = c1 * \underline{dx/dt}(\text{pixel}) + c2 * \underline{dx/dt}(\text{list}) + c3 * \underline{dx/dt}(\text{surface}) + c4 * \underline{dx/dt}(\text{object})$$
$$\underline{dy/dt}(x,y,t\text{-}1) = c1 * \underline{dy/dt}(\text{pixel}) + c2 * \underline{dy/dt}(\text{list}) + c3 * \underline{dy/dt}(\text{surface}) + c4 * \underline{dy/dt}(\text{object})$$

where c1 = confidence in the pixel level flow computation for the pixel at x, y
c2 = confidence in the list level flow computation for the pixel at x, y
c3 = confidence in the surface level flow computation for the pixel at x, y
c4 = confidence in the object level flow computation for the pixel at x, y

and $c1 + c2 + c3 + c4 = 1$

This is illustrated in more detail in Figure 19. In the outer loop of Figure 19, correlation offset at each pixel generates a measure of pixel flow error. This is added to the predicted flow rate to generate an estimate of pixel flow rate. This is then combined with higher level estimates of list entity flow, surface entity flow, and object entity flow to compute a best estimate of flow at each pixel.

Once a reliable estimate of image flow is known, the attribute of a pixel at position (x,y) at time t-1 can be predicted to appear at position (x+dx, y+dy) at time t. This can be expressed as:

$$I^*(x+dx, y+dy, t) = \underline{I}(x, y, t\text{-}1)$$

In Figure 19, a predicted intensity image $I^*(x,y,t)$ is generated from an estimated intensity image $\underline{I}(x,y,t\text{-}1)$ by computing the expected image motion for each pixel attribute based on image flow estimates $\underline{dx/dt}(x,y,t\text{-}1)$ and $\underline{dy/dt}(x,y,t\text{-}1)$ at each pixel, plus the effect of control signals sent to the camera pan/tilt unit, plus other control actions effecting the image.

Comparison between predicted and observed images produces correlation and difference values. As shown in Figure 18, the attribute difference between observed image I(x,y,t) and predicted (with zero offset) $I^*(x,y,t)$ is used to update the attribute estimate. Correlation of the observed image with a set of predicted images with different offsets produces a correlation function. The offset values that produce the maximum correlation value indicate the amount of position error (Δx and Δy) between the observed and predicted attribute images. These position errors divided by the sample period yield the pixel flow error.

A recursive estimation loop is required for each pixel in each attribute image. Thus, a recursive estimation loop similar to the intensity attribute loop in Figure 19 is required for each pixel for each attribute in all the attribute images listed in Figure18.

The combination of image flow estimates illustrated in Figure 19 suggests how recursive estimation provides a means for combining information about a single estimated pixel attribute from several sources. Each source or method of computation produces an observed attribute which can be correlated with the predicted attribute to update the estimated attribute. The blending of information from different sources is modulated by source confidence factors, so that high confidence sources of information will dominate over low confidence sources.

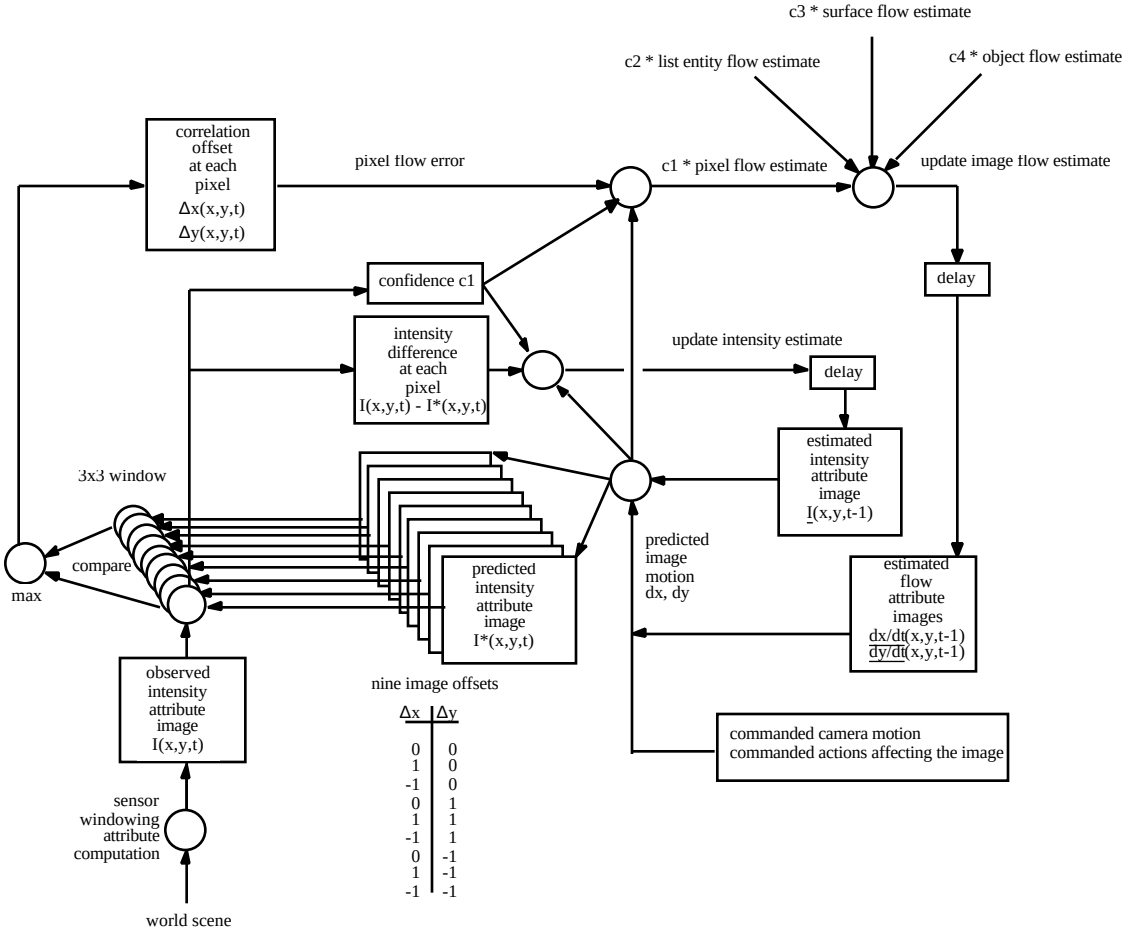


Figure 19. Recursive estimation loops for intensity and flow attribute images. Similar recursive estimation loops may exist for each of the other attribute images shown in Figure 18.

Level 1 Recognition

The set of attributes for each pixel define a pixel attribute vector. At level 1, recognition is achieved when the estimated attribute vector of a pixel matches the attribute vector of a pixel entity class. In 4-D/RCS, there are five pixel entity classes: brightness-discontinuity, color-discontinuity, range-discontinuity, range-gradient-discontinuity, and surface-continuity.

Upon recognition, each pixel acquires a new attribute -- i.e., the name of the pixel class to which it belongs. Thus, the classification process generates a pixel entity image where

each pixel in the entity image has an attribute(s) that is the name(s) of the pixel class(es) to which the pixel belongs.

Summary of level 1:
1. A portion of the egosphere is windowed onto each sensor.
2. The energy falling on each sensor is integrated over a sample interval in space and time.
3. Observed values are computed for a set of attributes for each pixel.
4. Estimated values are computed for each attribute of each pixel by a recursive estimation filter process.
5. Predicted values are computed for each attribute based on dynamic attributes and control input.
6. Each pixel is classified as a member of a topological point entity class, and a new pixel entity image is formed in which each pixel has a pointer (or pointers) to the class (or classes) to which it belongs.

Level 2 -- List Entities

Level 2 Windowing
At level 2, windowing consists of placing windows around regions in the list entity image that are classified as worthy of attention. The selection of which regions to so classify depends on the goal and priorities of the current level 2 task, or on the detection of noteworthy estimated list entities, or on inclusion in a higher level entity of attention. The shape of each window is determined from the recognized list entity image. The size of each window is determined by the confidence factor computed by the level 2 recursive filtering process. If the estimated entity confidence value is high, the window will be narrowed to only slightly larger than the set of pixels in the list entity image. If the estimated entity confidence value is low, the window will be significantly larger than the level 2 list entity image. Until there exists a recognized list entity image, level 2 windows are set wide open.

Level 2 Grouping
At level 2, grouping is a process by which neighboring pixels of the same class with similar attributes can be grouped to form higher level entities. Information about each pixel's class is derived from the recognized pixel entity image. Information about each pixel's orientation and magnitude is derived from the predicted pixel attribute images. Grouping is performed by a heuristic algorithm based on gestalt principles such as contiguity, similarity, proximity, pattern continuity, or symmetry. The choice of which gestalt heuristic to use for grouping may depend on an attention function that is determined by the goal of the current task, and on the confidence factor developed by the recursive estimation process at level 2. Grouping causes an image to be segmented, or partitioned, into regions (or sets of pixels) that correspond to higher order entities.

For example at level 2, contiguous pixels in the same topological class with similar attributes of range, orientation, and flow rate can be grouped into topological list entities such as range-edges, brightness-edges, color-edges, surface-slope-discontinuity-edges, surface-patches, or vertices of various kinds.

Any grouping is essentially a hypothesis that the pixels in the group all lie on the same entity in the world. The grouping process thus generates a hypothesized list entity image in which each group is assigned a label. A grouping hypothesis can be tested by a recursive estimation filtering algorithm applied to each of the hypothesized entities. If the recursive filtering process is successful in predicting the observed behavior of a hypothesized entity in the image, then the grouping hypothesis for that entity is confirmed. On the other hand, if the recursive filtering process is not successful in predicting the behavior of the

hypothesized entity, the grouping hypothesis will be rejected, and another grouping hypothesis must be selected.

In practice, a number of grouping hypotheses may be tested in parallel if the computational resources are available. In this case, the grouping hypothesis with the greatest success in predicting will be selected. In either case, the measure of success in predicting behavior is the confidence factor computed by the filtering algorithm for the estimated entity.

Level 2 Computation
Each of the entities defined by the grouping hypotheses have attributes that can be computed. Entity attributes differ from pixel attributes in that they are integrated over the entire group of pixels comprising the entity. For example, edge entities have attributes such as edge length, edge curvature, edge orientation, position and motion of the edge center of gravity (c.g.), etc. Surface-patch entities have attributes such as area, texture, average range, average surface gradient, average color, position and motion of the c.g.

For each hypothesized list entity, computed entity attributes fill slots in an observed list entity frame.

Level 2 Filtering
At level 2, a recursive estimation process compares observed list entity attributes with predicted list entity attributes that are generated from estimated list entity attributes. Prediction is a function of information from two sources:
1) estimated dynamic attributes such as image flow rates ($\frac{dx}{dt}$, $\frac{dy}{dt}$) and range rate ($\frac{dr}{dt}$) that describe the motion of entities in the image.
2) commanded actions that affect motion of entities in the image

Comparison of observed entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the estimated entity attribute values. Estimates of image flow rates of entities at level 2 are generated from a combination of correlation offsets generated at level 2, plus estimates of flow attributes from higher levels. A confidence level for each estimated entity attribute value is computed as a function of the correlation and difference values. If predictions based on estimated entity attributes successfully track the behavior of observed entity attributes, the confidence level rises. When the confidence level of the recursive estimation filter rises above threshold, the hypothesized list entity grouping is confirmed. If the confidence level of the recursive estimation filter falls below threshold, the hypothesized list entity grouping is rejected, and another grouping hypothesis must be selected.

Level 2 Recognition
The recognition process compares the attributes of each estimated list entity with attributes of generic list entity classes stored in the knowledge database. At level 2, recognition establishes a match between an estimated list entity and a class of generic list entities in the world model knowledge database, such as patch of ground, near edge of ditch, far edge of ditch, crest of hill, edge of tree trunk, edge of building, edge of road, edge of foliage.

Upon recognition, each pixel acquires a new attribute -- i.e., the name of the list entity class to which it belongs. Thus, the recognition process generates a confirmed list entity image with classified regions where each pixel in the list entity image has an attribute (or attributes) that is the name (or names) of the list entity class (or classes) to which the pixel belongs.

Summary of level 2:
1. Portions of the image containing a set of list entities of attention have been windowed.
2. Contiguous pixels with similar attributes have been tentatively grouped into topological list entities.
3. The attribute values of each list entity have been computed.
4. Each attribute of each list entity has been estimated by a recursive filter. Based on confidence factors computed by the recursive filter, the grouping hypotheses are confirmed or rejected.
5. Predicted attributes are computed to be used in recursive estimation.
6. The attributes of each estimated list entity have been compared with the attributes of a set of list entity classes, and each list entity has been assigned to a list entity class. As a result, a recognized entity image is formed wherein each pixel carries the name of (or pointer to) the list entity class to which it belongs.

Level 3 -- Surface Entities

Level 3 Windowing
At level 3, windowing consists of placing windows around regions in the list entity image that are classified as worthy of attention. The selection of which regions to window is made by an attention function that depends on the goal and priorities of the current level 3 task, or on the detection of noteworthy attributes of estimated surface entities, or on inclusion in a higher level entity of attention. The shape of the windows are determined by the recognized regions in the surface entity image. The size of the windows are determined by the confidence factor associated with the degree of match between the predicted surface entities and the observed surface entities.

Level 3 Grouping
At level 3, recognized list entities in the same class are grouped into surface entities based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Surface patches in the same class that are contiguous at their edges and have similar range, velocity, average surface orientation, and average color may be grouped into surface entities. Edges and vertices in the same class that are contiguous along their orientation with similar range and velocity may be grouped into boundary entities. Grouping is a hypothesis that all the pixels in the group have the common property of imaging the same physical surface or boundary in the real world. This grouping hypothesis will be confirmed or rejected by the level 3 filtering function.

Level 3 Computation
Observed surface entities have attributes such as area, shape, position and motion of the center-of-gravity, range, orientation, surface texture, and color. Surface boundary entities have attributes such as boundary type, length, shape, orientation, position and motion of the c.g., and rotation. Boundary types may include intensity, range, texture, color, and slope boundaries. For each surface entity, computed entity attributes fill slots in an observed surface entity frame.

Level 3 Filtering
A recursive estimation process compares observed surface and boundary entity attributes with predicted surface and boundary entity attributes generated from estimated surface entity attributes, planned results, and predicted attributes of parent entities.

Comparison produces correlation and difference values. Difference values are used to update the estimated surface entity attributes. Correlation values are used to update

estimates of entity motion. A confidence level associated with each estimated entity confirms or rejects the surface entity grouping hypothesis that created it.

Level 3 Recognition
The recognition process establishes a match between estimated surface entities and a generic class of surface entities in the world model knowledge database such as ground, tree foliage, tree trunk, side of building, roof of building, road lane, fence, or lake surface. As a result of recognition, a recognized surface entity image is formed in which each pixel has a pointer to (or the name of) the surface entity to which it belongs.

At level 3, the surface entity image can be transformed into a map of the ground in the vicinity of the vehicle. In 4-D/RCS, this map will be a 100 meter by 100 meter vehicle centered map with 10 centimeter resolution. In some cases, this map may be used to update the map provided by the digitized battlefield database.

Summary of level 3:
1. Portions of the image consisting of a set of topological surface and boundary entities of attention have been windowed.
2. Contiguous list entities with similar attributes have been tentatively grouped into topological surface entities.
3. The attribute values of each topological surface and boundary entity have been computed.
4. Each attribute of each topological surface entity has been estimated by a recursive filter. Based on confidence factors computed by the recursive filter, the grouping hypotheses are confirmed or rejected.
5. Predicted attributes of each topological surface entity are computed.
6. The attributes of each topological surface entity have been compared with the attributes of a set of generic surface entity classes, and each surface entity has been assigned to a surface entity class. As a result, a recognized surface entity image and a vehicle centered map have been formed wherein each pixel carries the name of the generic surface class to which it belongs.

Level 4 -- Object Entities

Level 4 Windowing
At level 4, windowing consists of placing windows around regions in the image that are classified as worthy of attention. The selection of which regions to so classify depends on the goal and priorities of the current level 4 task, or on the detection of noteworthy attributes of estimated object entities, or inclusion in higher level entities of attention. The shape of the windows are determined by set of pixels in the recognized object entity image. The size of the windows are determined by the confidence factor generated by the level 4 recursive estimation filter.

Level 4 Grouping
Surface and boundary entities in the same class are grouped into topological object entities based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Surfaces in the same class that are contiguous along their boundaries and have similar range and velocity may be grouped into object entities. Boundaries that separate surfaces with different range, velocity, average surface orientation, and average color are used to distinguish between different objects.

Level 4 grouping is a hypothesis that all the pixels in the group have the common property of imaging the same physical object in the real world. This grouping hypothesis will be confirmed or rejected by the level 4 filtering function.

Level 4 Computation
For each of the hypothesized object entities, entity attributes such as position, range, and motion of the center-of-gravity, average surface texture, motion, average color, solid-model shape, projected area, and estimated volume can be computed. These computed attributes become observed entity attributes in an observed object entity attribute frame.

Level 4 Filtering
A recursive estimation process compares observed object entity attributes with predictions based on estimated object entity attributes. Included in the prediction process are estimated motion of object entities, and expected results of commanded actions. Comparison of observed entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the attribute values in the estimated object entity frames. A confidence level is computed as a function of the correlation and difference values. If the confidence level of the recursive estimation filter rises above threshold, the object entity grouping hypothesis is confirmed.

Level 4 Recognition
The recognition process compares the attributes of each confirmed object entity with attributes of object entity classes stored in the knowledge database. A match causes a confirmed topological object to be classified as a generic, or a specific object entity in the world model knowledge database.

For example in the case of a ground vehicle performing a typical task, a list of generic object classes may include the ground, the sky, the horizon, dirt, grass, sand, water, bush, tree, rock, road, mud, brush, woods, log, ditch, hole, pole, fence, building, truck, tank, etc. There may be several tens, hundreds, or even thousands of generic object classes in the KD. A list of specific object classes may include a specific tree, bush, building, vehicle, road, or bridge.

As a result of recognition, a new object entity image is formed in which each pixel has a pointer to (or the name of) the generic or specific object entity to which it belongs.

At level 4, the object entity image can be transformed into a map of the ground in the vicinity of the vehicle. In 4-D/RCS, this map will be a 1 km by 1 km vehicle centered map with 1 meter resolution. In some cases, this map may be used to update the map provided by the digitized battlefield database.

Summary of level 4:
1. Portions of the image consisting of a set of object entities of attention have been windowed.
2. Contiguous confirmed surface and boundary entities with similar attributes have been tentatively grouped into object entities.
3. The observed attribute values of each object entity have been computed.
4. Each attribute of each object entity has been estimated by a recursive filter. Based on confidence factors computed by the recursive filter, the grouping hypotheses are confirmed or rejected.
5. Predicted attributes of each object entity are computed.
6. The attributes of each object entity have been compared with the attributes of object entity classes, and object entities have been recognized as belonging to generic or specific object classes. As a result, a new entity image and a vehicle centered map have been formed wherein each pixel carries the name of the object class to which it belongs.

Level 5 -- Squad Entities

Level 5 Windowing
At level 5, windowing consists of placing a window around regions in the image that are classified as squad entities-of-attention. The selection of which regions to so classify depends on the task, and on detection of entity attributes that are worthy of attention. The shape of the windows are determined by classified regions in the squad entity image. The size of the windows are determined by the confidence factor associated with the level 5 recursive estimation process.

Level 5 Grouping
Object entities in the same class are grouped into squad entities based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Objects in the same class that are near each other and have similar range and velocity may be grouped into squad entities. These grouping hypotheses will be confirmed or rejected by the level 5 filtering function.

Level 5 Computation
For each of the hypothesized squad entities, entity attributes such as position, range, and motion of the center-of-gravity, density, and shape can be computed. These computed attributes become observed entity attributes in an observed squad entity attribute frame.

Level 5 Filtering
A recursive estimation process compares observed squad entity attributes with predictions based on estimated squad entity attributes, planned results, and predicted attributes of platoon entities. Comparison of observed squad entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the attribute values in the estimated squad entity frames. A confidence level is computed as a function of the correlation and difference values. If the confidence level of the recursive estimation filter rises above threshold, the squad entity grouping hypothesis is confirmed.

Level 5 Recognition
The recognition process compares the attributes of each confirmed squad entity with attributes of squad entity classes stored in the knowledge database. A match causes a recognized topological squad entity to be classified as a generic, or a specific squad entity in the world model knowledge database.

For example in the case of a ground vehicle, a list of generic squad classes may include a woods, a field, a friendly tank squad, a hostile tank squad, a group of trucks, a group of friendly humans, a group of hostile soldiers. There may be several tens, hundreds, or even thousands of generic squad classes in the KD. A list of specific squad classes may include a specific group of humans, trees, bushes, buildings, vehicles, or the intersection of two or more roads.

As a result of recognition, a new entity image is formed in which each pixel has a pointer to (or the name of) the generic or specific squad entity to which it belongs.

At level 5, the recognized squad entity image can be transformed into a map of the ground in the vicinity of the vehicle. In 4-D/RCS, this map will be a 10 km by 10 km vehicle centered map with 3 meter resolution. In some cases, this map may be used to update the map provided by the digitized battlefield database.

Summary of level 5:
1. Portions of the image containing squad entities of attention have been windowed.
2. Contiguous recognized object entities with similar attributes have been tentatively grouped into squad entities.
3. The attribute values of each observed squad entity have been computed.
4. Each attribute of each squad entity has been estimated by a recursive filter. Based on confidence factors computed by the recursive filter, the grouping hypotheses are confirmed or rejected.
5. Predicted attributes of each squad entity are computed.
6. The attributes of each squad entity have been compared with the attributes of squad entity classes, and squad entities have been recognized as belonging to generic or specific squad classes. As a result, a new entity image and a vehicle centered map have been formed wherein each pixel carries the name of the squad class to which it belongs.

Hierarchy of Tasks and Entities

The 4-D/RCS architecture is a hierarchy of tasks and entities. It is based on the decomposition of tasks and the assignment of tasks to operational units. Operational units correspond to TD modules, and the SP and WM modules within the 4-D/RCS nodes are designed to provide the TD modules with the knowledge of the state of task objects necessary to accomplish task goals. The task goals for the various operational units at different levels of the 4-D/RCS hierarchy are different, as are the task objects.

For example, at the servo level, task objects are state variables, each of which can be computed from one or more sensor signals at a single point in space and time. Task goals are desired values of these state variables. State variables are in sensor and actuator coordinates.

At the Trajectory level, task objects are edges, vertices, or trajectories that represent linear features in space and time such as road edges, obstacle edges, etc. Task goals are desired positions, velocities, or orientations of these features in a vehicle coordinate frame.

At the Subsystem/ E-move level, task objects may be surfaces of obstacles or terrain features. Task goals are desired relationships between the vehicle and these task objects expressed in task object surface coordinates.

At the Vehicle level, task objects may be real world objects such as vehicles, buildings, trees, bushes, etc. Task goals are desired relationships between the vehicle and these task objects in task object coordinates.

Thus, the recursive loop illustrated in Figures 14 - 16 is repeated for each entity attribute for each node at each level in the 4-D/RCS hierarchy. Typically, tasks and world model information are expressed in a different coordinate system at each different hierarchical level.

A Generic Processing Node

Details of the relationships and interactions between the TD, WM, KD, SP, and VJ modules in a generic node of the 4-D/RCS architecture are shown in Figure 20. The Task Decomposition (TD) modules contain the Planner (PL) and Executor (EX) sub modules. The Planner contains the Job Assignment (JA), Scheduling (SC) and Plan Selector (PS) sub-submodules. The World Modeling (WM) module contains the Knowledge Database (KD), with both long term and short term symbolic representations and short term iconic

images. In addition, WM contains a Simulator where the alternatives generated by JA and SC are tested for comparison in PS. The Sensory Processing (SP) module contains filtering, detecting, and estimating algorithms, plus mechanisms for comparing predictions generated by the WM module with observations from sensors. It has algorithms for recognizing entities and grouping entities into higher level entities. The Value Judgment (VJ) module evaluates plans and computes confidence factors based on the variance between observed and predicted sensory input.

Each node of the 4-D/RCS hierarchy closes a control loop. Input from sensors is processed through sensory processing (SP) modules and used by the world modeling (WM) modules to update the knowledge database (KD). This provides a current best estimate $\hat{x}$ of the state of the world x. A control law is applied to this feedback signal arriving at the EX sub module. The EX sub module computes the compensation required to minimize the difference between the planned reference trajectory and the trajectory which emerges as a result of the control process. The value of "best estimate" $\hat{x}$ is also used by the JA and SC submodules and by the WM plan simulator to perform their respective planning computations. It is also used in generating a short term iconic image that forms the basis for comparison, recognition, and recursive estimation in the image domain in the sensory processing SP module.

The entities that comprise task objects vary at different levels of the 4-D/RCS hierarchy. Spatial regions where entities are recognized can then be grouped into higher level entities. For example, points where edges are detected can be grouped into boundaries. Points where range is varying smoothly can be grouped into surfaces. Surfaces that are connected may be grouped into an object. Objects that have similar properties can be grouped into groups, etc.

Temporal intervals where events are detected can be grouped into higher level events. For example, periodic acoustic signals can be grouped into frequencies, strings of phonemes can be grouped into words, strings of words into sentences, etc.

Grouping can be accomplished bottom up by gestalt principles of nearness, continuity, symmetry, etc., or top down by windowing provided by world model predictions.

At each level of the 4-D/RCS hierarchy, this process of windowing, comparison, spatial and temporal integration, detection, and grouping is repeated. At each level, lower level entities are recognized and grouped into higher level entities. The result is a sensory processing and world modeling hierarchy of entities that corresponds closely to the task decomposition hierarchy of task decomposition. At each level, entities are recognized and attributes are computed that support the planning and control functions at that level.

This hypothesis and test approach requires an enormous amount of processing to be accomplished in real-time. In order to be practical, it is necessary to focus attention in space and time, and in resolution and scale, through hierarchical layering and windowing, and through pyramid processing techniques, and to employ parallel processing techniques as appropriate.

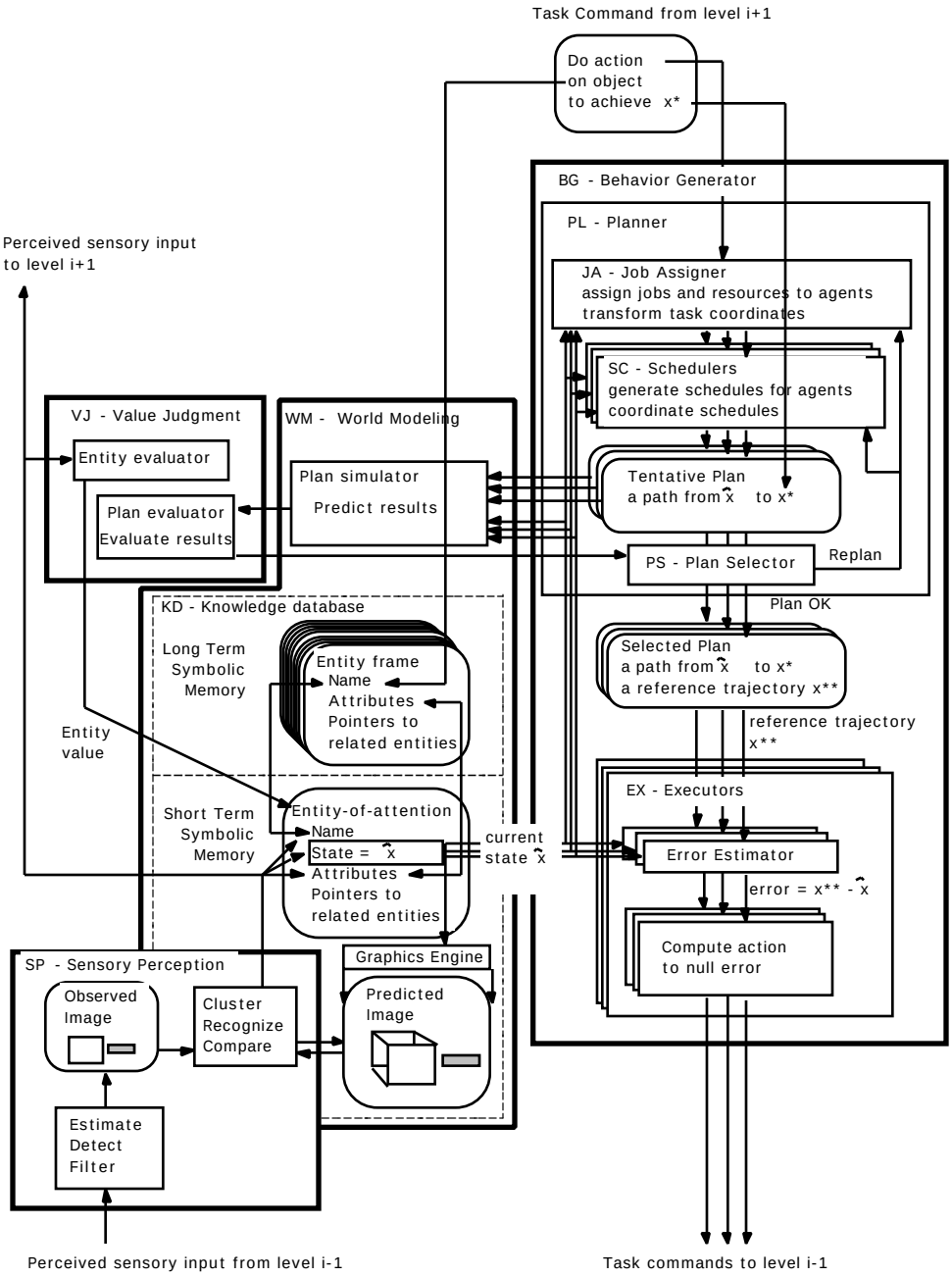


Figure 20. Relationships within a single node of the 4-D/RCS architecture A task command from level i+1 specifies the goal and the object. The object specification selects an entity from long term memory and moves it into short term memory. The TD module generates a plan that leads from the current state to the goal state. The EX sub modules execute the plan using estimated state information in the WM KD short term memory. The SP module processes sensory input from level i-1 to update the WM KD within the node, and sends processed information to level i+1.

III. 4D/RCS as an Implementation Guide for Demo III

System Clock

There should be a system clock for each vehicle system that provides sync pulses to various modules. Each module should have its own clock that allows interpolation between sync pulses. The following sync pulse rates should be provided:

1 microsecond
1 millisecond
4 milliseconds - Control system sample / output rate
32 milliseconds - TV camera frame rate, Servo level command rate
256 milliseconds - Trajectory level command rate
1 second
1 minute
1 hour

The clock should be syncronized to the operator control station to 1 microsecond, or to whatever accuracy is required by the communication and navigation systems.

Vision System for Driving

The 4-D/RCS architecture shall allow a number of camera systems and image processing software and hardware to be integrated into the Demo III vehicles. In particular, the 4-D/RCS architecture shall accomodate the baseline NIST HMMWV vision system consisting of a laser range imager and two CCD cameras mounted on an inertially stabilized camera platform.

The laser range imager is a Dornier system that produces a range image consisting of 128 x 64 pixels with approximately 64 x 32 deg field of regard. This yields an image with about 0.5 deg per pixel as shown in Figure 21.

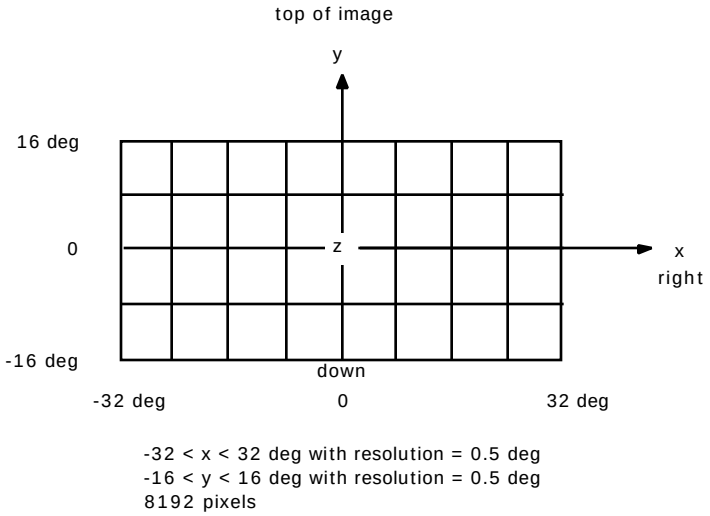
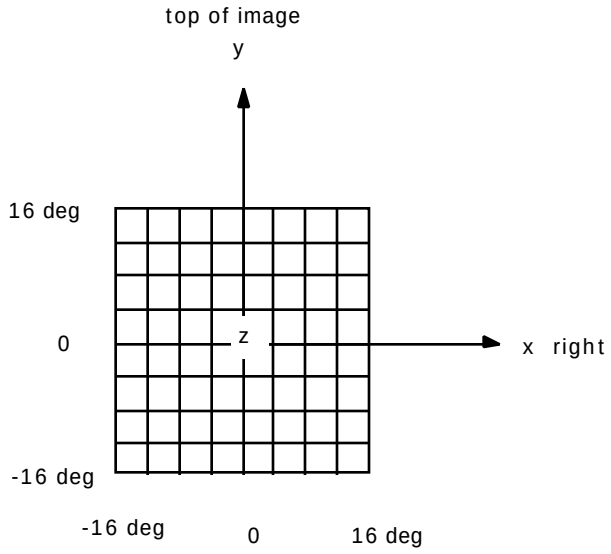


Figure 21. The laser range imager field of view.

The wide angle CCD camera produces a red, blue, green, and intensity image of 256x256 pixels with a 32x32 deg field of view. This yields an image with about 0.125 deg/pixel as shown in Figure 22.



-16 < x < 16 deg with resolution = 0.125 deg
-16 < y < 16 deg with resolution = 0.125 deg
65536 pixels

Figure 22. The wide angle CCD camera field of view.

The foveal CCD camera produces a red, blue, green, and intensity image of 256x256 pixels with a 8x8 deg field of view. This yields an image with about 0.031 deg/pixel as shown in Figure 23. This is slightly lower resolution than normal human foveal vision.

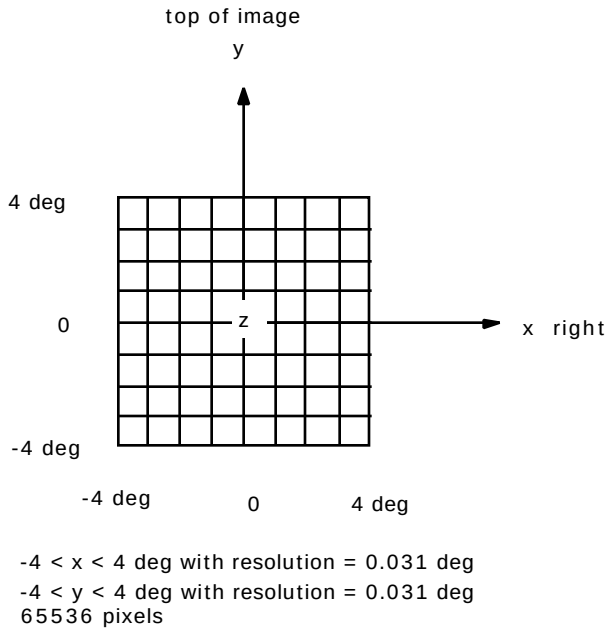


Figure 23. The foveal CCD camera field of view.

Additional cameras may be added or substituted for these. For example, stereo cameras will be required for some applications. FLIR or night vision cameras may be used.

The fields of view (FOV) of the laser range imager, the wide angle, and foveal CCD cameras can be registered and projected onto a Mercator projection of the retinal egosphere as shown in Figure 24.

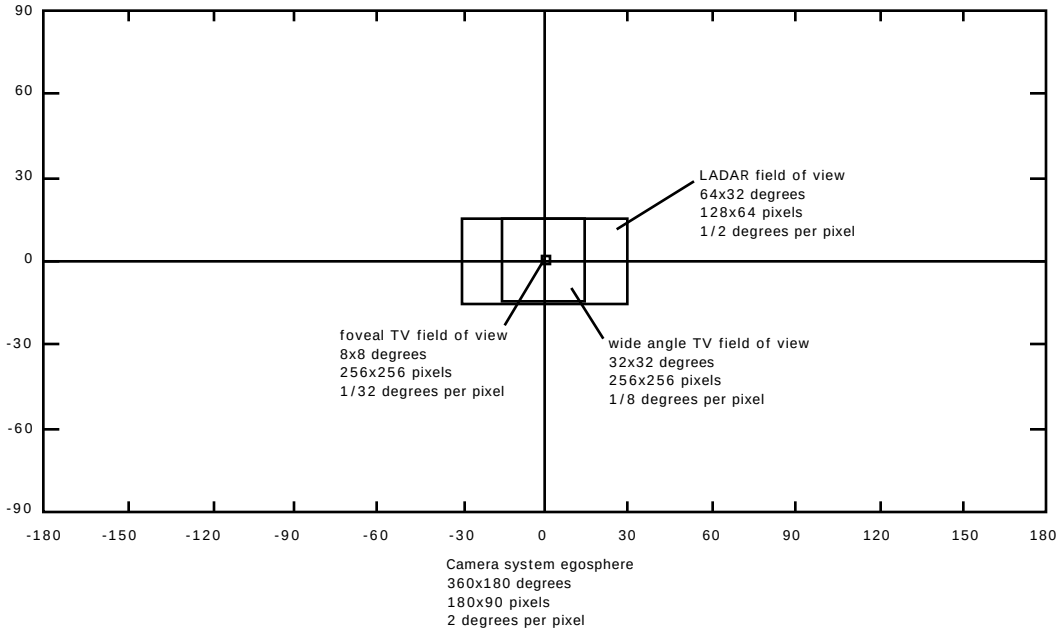


Figure 24. The retinal egosphere in Mercator projection coordinates

A definition of the retinal egosphere is shown in Figure 25.

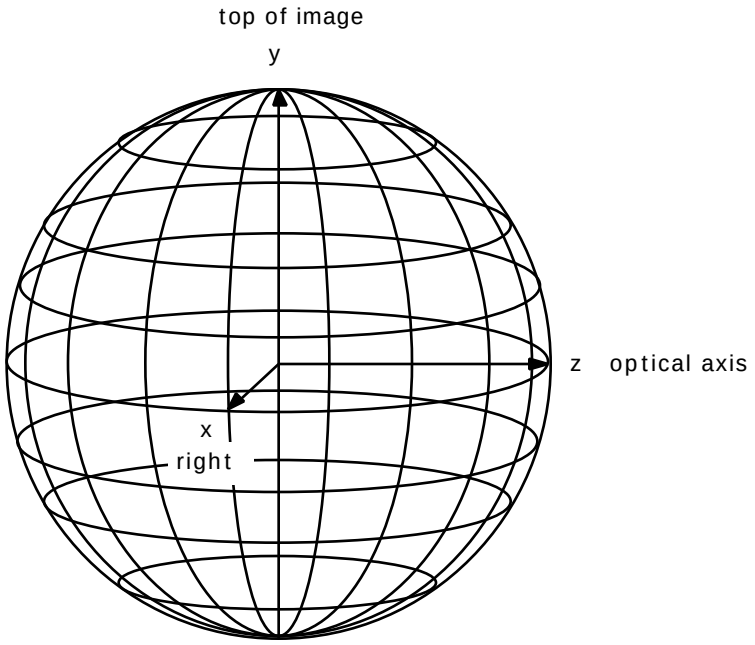


Figure 25. The retinal egosphere in spherical egosphere coordinates

Laser Range Images

For the laser range imager, at least six attribute images will be maintained. These will be:
1) r = range
2) dr/dx = range x gradient, or surface slope along x direction
3) dr/dy = range y gradient, or surface slope along y direction
4) dr/dt = range rate, or surface motion along the pixel line of sight
5) r-edge = |dr/dx| or |dr/dy| > threshold
6) r-edge-angle = orientation of r-edge (0 ˚ - 360 ˚ counter-clockwise around the pixel line of sight)

The ladar image will be analyzed to extract range, range rate, x and y range gradients, and range discontinuities at each pixel.

Pixels in the ladar image will be grouped or clustered into image regions that can be segmented and recognized based on the following attributes:

1) Smooth surfaces where range values vary relatively smoothly from pixel to pixel. Smooth surfaces may consist of the smooth ground, sides of buildings, walls, sand, snow, short grass, rocks, trunks of trees, barrels, boxes, vehicles, etc. For these types of surfaces it is possible to compute surface range and surface slope at each ladar pixel.

2) Surface discontinuities where range values change abruptly from one smooth surface to another. Surface discontinuities may correspond to the edges and roof lines of buildings, sides of tree trunks, or the sides and tops of rocks, barrels, boxes, vehicles, etc. Surface discontinuities also exist at the crest of hills, or the near edges of cliffs or gullies.

3) Surface joins where surface slopes change abruptly from one smooth surface to another. Surface joins may correspond to the bottom edge of walls, curbs, ditches, or where rocks and tree trunks emerge from the ground.

4) Rough surfaces where range values vary abruptly from one pixel to the next. Rough surfaces may correspond to tree foliage, branches, bushes, tall grass, piles of rubble, etc.

5) Thin surfaces where near range values only occasionally are returned. Thin surfaces may correspond to ropes, wires, and single thin tree branches.

6) Specular surfaces where range values are absent altogether. Specular surfaces may correspond to water, glass, or polished metal.

CCD Image Processing

The two CCD camera images will be processed to determine attributes of color, brightness, and spatial and temporal brightness gradients for each pixel. Additional processing will provide estimates of image flow and flow discontinuity values at each pixel. Brightness, color, and image flow discontinuities will be thresholded and correlated to find brightness boundaries.

Combining Ladar and CCD Images

Ladar and CCD attributes will be combined to add color and brightness attributes to the ladar image, and to add range and surface attributes to the CCD image.

There are 4x4=16 CCD pixels per range pixel in the wide angle CCD, and 16x16=256 CCD pixels per range pixel in the foveal camera as shown in Figure 26.

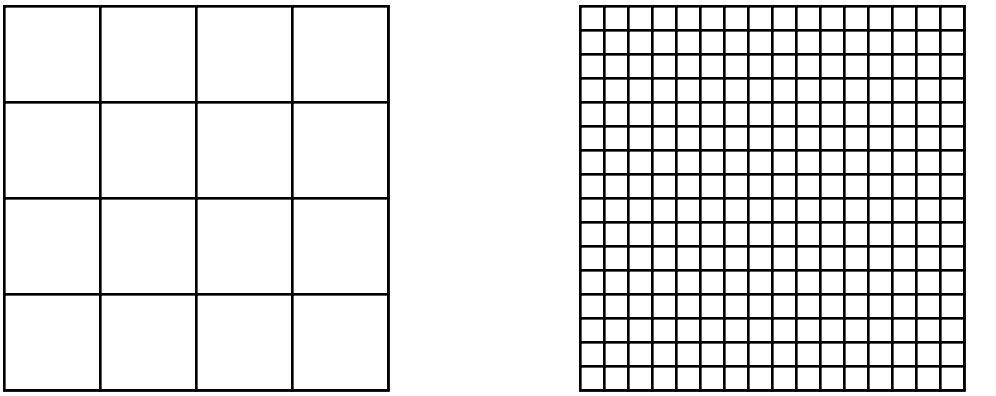


Figure 26. Ratio of CCD pixels to one laser range pixel

Because there are many CCD pixels within each ladar pixel, it is easy to assign color and brightness attributes to the ladar image. Color and brightness information can simply be averaged over all the CCD pixels within the spatial region occupied by the ladar pixel.

On the other hand, adding range attributes to the brightness pixels is more complicated.

For smooth surfaces, the problem is simplest. A first approximation of range for each pixel can be computed from the ladar pixel attributes of range plus x and y range gradients. (For most purposes, this is sufficient. If more detail is desired, shape-from-shading algorithms might be used to refine range estimates for each CCD pixel.)

At surface boundaries, two adjacent ladar pixels have significantly different range values. This implies that somewhere between the centroids of the two ladar pixels there exists a surface boundary. It is quite likely that there will also be a sharp brightness discontinuity at the same surface boundary. Therefore, brightness edges between the two range edge pixels that have the same orientation as the range edge can be assumed to correspond to the surface edge. CCD pixels on the surface side of the brightness edge will be assigned range values corresponding to the surface range value. CCD pixels on the background side of the brightness edge will be assigned range values corresponding to the background range value.

For rough surfaces, such as tree or bush leaves and branches, CCD pixels with color or brightness attributes corresponding to the branches or leaves will be assigned the nearest range values. CCD pixels with color or brightness attributes corresponding to the background will be assigned the more distant range values. Differential CCD image flow rates may also be used in making range assignments to CCD pixels for rough surfaces.

For thin surfaces, CCD pixels with color or brightness attributes corresponding to a rope or wire or narrow branch will be assigned the nearest range values. CCD pixels corresponding to the background will be assigned the more distant range values.

For specular surfaces, CCD pixels at the boundaries of the specular surface will be assigned range values the same as the nearest valid range pixel. The range to interior CCD pixels on specular surfaces will be assumed to vary smoothly over the region between the specular surface boundaries.

Relative Timing

The laser range imager produces a range image every second with 0.5 deg resolution.

The two CCD cameras produce 30 images per second with .125 and .031 deg resolution respectively.

This means that each CCD pixel is updated 30 times for each corresponding laser range imager update. This is illustrated in Figure 27.

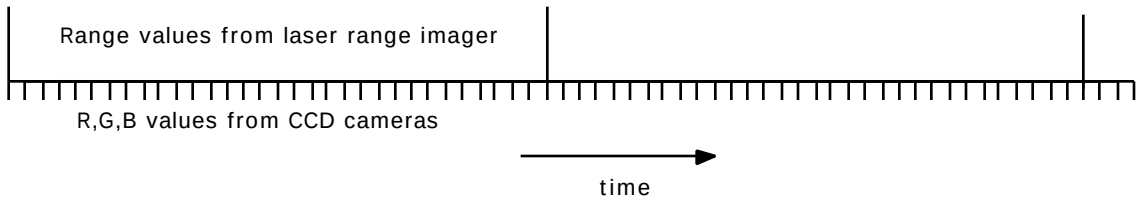


Figure 27. Timing of range images vs. CCD image. For each pixel, a CCD camera pixel update occurs 30 times for every ladar pixel update.

It thus becomes necessary for the CCD camera pixels to flywheel their estimate of range at each pixel through 30 frames of CCD images between each range estimate update from the laser range imager. This can be done using knowledge of camera motion and measured image flow.

Combined Attribute Images

Once the attributes from the Ladar images have been combined with the two CCD camera images, a set of at least 22 attribute images will be computed for the fields of view of the two CCD images. These attribute images will be maintained in registration as shown in Figure 12.

In each attribute image, the estimated value of each attribute will be initially set to some a priori value, and subsequently updated by a recursive estimation procedure of the form:

$$\hat{a}(x, y, t|t) = a^*(x, y, t|t-1) + K(a(x, y, t) - a^*(x, y, t|t-1))$$

where

$\hat{a}(x, y, t|t)$ is the estimated value of the attribute a at pixel (x, y) at time t after the measurement at time t
$a^*(x, y, t|t-1)$ is the predicted value of a at pixel (x, y) at time t after the measurement at t-1
$a(x, y, t)$ is the observed value of a at pixel (x, y) at time t

K(t) is the confidence in the observed value at time t,
with K(t) = 0 => no confidence in the observed value,
and K(t) = 1 => complete reliance on the observed value.
$0 \leq K(t) \leq 1$

Image Segmentation and Entity Recognition

Once a full set of range, brightness, and edge attributes are assigned to each pixel at each resolution, it becomes possible to perform grouping of pixels and segmentation of the image into surface regions based on similarity and contiguity between pixel attributes. Edge pixels can be grouped into boundary entities, and vertex entities. Surface pixels can be grouped into surface entities that are delimited by boundary and vertex entities. Surface, boundary, and vertex entities can be further grouped into object entities. Entity attributes such as size, shape, orientation, position, velocity, and rotation can be computed over the image regions occupied by the entities. Pixels that belong to entities define entity images. Entity images can be used to define masks and windows that facilitate efficient tracking and rapid processing of entity images. An example of an entity image is shown in Figure 13.

Attention

The estimated value of each attribute for every pixel need not be computed every frame. Attention mechanisms driven by the task can control when and where various entity images will be processed and select which processing algorithm will be applied. Windows will be used to mask out portions of the image, based on expectations and task requirements.

There will be a list of entities of attention. Entities will be added to this list on the basis of two factors:

1) Entities in an active task frame will become entities of attention. The world will be searched for these entities, and once they are discovered, they will be tracked until they are no longer in an active task frame.

2) Regions of the image that do not conform to expectations will be labeled unexplained and entered into the list of entities of attention. These entities may become the object of inspection behavior, depending on competing task priorities.

When there are items on the list of entities of attention, windows can be placed around the entities of attention, and detailed processing of all pixels will take place only within the windows. This will speed up processing of items being attended to.

When there are no items on the list of entities of attention, all pixels in the image will be processed. This will take longer, but there is no rush.

SUMMARY AND CONCLUSIONS

This document is in progress. Additional material will be added, and modifications and corrections are on going. It is expected that the Demo III integration contractor will work with the technology providers to modify, extend, and clarify this document where required.

REFERENCES

J. S. Albus, A. M. Meystel, A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems, *International Journal of Intelligent Control and Systems,* Vol.1, No. 1, 1996, pp15-30

P. J. Burt, Multiresolution Techniques for Image Representation, Analysis, and 'Smart' Transmission, *SPIE Conf. 1199: Visual Communications and Image Processing IV,* Philadelphia, Nov. 1989

P. J. Burt, Attention Mechanisms for Vision in a Dynamic World, IEEE, 1988

P. J. Burt, E. H. Adelson, The Laplacian Pyramid as a Compact Image Code, *IEEE Transactions on Communications,* VOL. COM-31, No. 4, April 1983

P.J. Burt, G. van der Wal, An Architecture for Multiresolutional Focal, Image Analysis, *Proceedings of 10th International Conference on Pattern Recognition*, Atlantic City, Nune 16-21, 1990

P. J. Burt, et al., Object Tracking with a Moving Camera: An Application of Dynamic Motion Analysis, *Proceedings of the Workshop on Visual Motion*, Irvine, CA, March 20-22, 1989

E. D. Dickmanns, Parallel Use of Differential and Integral Representations for Realising Efficient Mobile Robots, *7th ISRR*, Munich, Germany, 1995

E. D. Dickmanns, et. al., The Seeing Passenger Car 'VaMoRs-P', *International Symposium on Intelligent Vehicles'94*, Paris, October 24-26, 1994

E. D. Dickmanns, Machine Perception Exploiting High-Level Spatio-Temporal Models, *AGARD Lecture Series 185: Machine Perception*; Hampton, VA; Munich; Madrid, Sept. 1992

E. D. Dickmanns, A General Dynamic Vision Architecture for UGV and UAV, *Journal of Applied Intelligence* 2.251-270 (1992), 1992 Kluwer Academic Publishers, Boston, Manufactured in The Netherlands

E.D. Dickmanns and V. Graefe, a) Dynamic monocular machine vision, b) Application of dynamic monocular machine vision. Journal of Machine Vision & Applications, Springer-Int, Nov. 1988, pp 223-261