The NIST Mobility Testbed

William G. Rippey
Robot Systems Division
National Institute of Standards and Technology (NIST)
Gaithersburg, MD 20899
rippey@cme.nist.gov

Abstract

This paper describes the application of NIST's Real-Time Control System (RCS) reference architecture for intelligent systems to autonomous robotics. The focus is on the Mobility Testbed, a facility which supports interdependent research in two areas: 1) refinement of the RCS reference architecture and development of its methodology (the steps and tools used to derive specific implementations from the reference architecture), and 2) experiments to develop advanced vision and world modeling techniques for autonomous robots.

NIST researchers gain several benefits by using Testbed resources and the consistent approach to control architecture provided by the reference architecture. Developers outside of NIST can also gain these benefits by using a reference architecture and a corresponding methodology for system development.

Keywords: reference architecture, robotic land vehicle, hierarchical control.

1. The Mobility Testbed Project

1.1 Goals

The Mobility Testbed project has two technical goals. The first is to develop a formal methodology for systems engineering of real-time intelligent machines that is based on the RCS reference architecture (described in section 2.1). The methodology covers high-level system design, detailed design of system components and interfaces, and subsequent implementation and testing of control systems. The second goal is to contribute to development of sensor and control systems for a sensor-guided autonomous mobile robot.

The rapid pace of technology development is providing engineers and scientists with many new computer-based products that can benefit society. To fully harness the potential of these products, developers must be able to integrate many specialized components into large, intelligent, complex systems. Efficient systems development and integration requires improvements in current engineering methods, models, tools, and the development of systems integration standards.

Further, NIST believes that the development of an open-system architecture reference model for real-time machines, and development of integration standards and computer-aided tools for machine design and implementation, is feasible and necessary for U.S. industry to remain competitive in the world high-technology marketplace. Many closed-system automation and robotic products exist in the marketplace today. Proprietary closed-system solutions hamper efforts to develop large, complex, real-time, intelligent control systems because they restrict access to information necessary to implement interfaces between components developed by different vendors.

1.2 Research

The testbed comprises resources for implementing robotic control and sensor systems. These resources include a mobile robot, cameras and image processing systems, computer and sensor interface hardware, software, and operating system services. The mobile robot, on-board cameras, and video monitors of the vision processing system are shown in Figure 1. The current experiments focus on controlling a vision-guided mobile robot to avoid obstacles, center its motion between obstacles, and map the robot's environment.

NIST's current development practices are documented in an internal document "Testbed Guidelines," which describes the RCS reference model, RCS design and implementation methodology, and integration standards for developing systems on the testbed. The guidelines are derived from experience of developing modules to support applications development, providing systems services such as code configuration control, and participating in testbed demonstrations.

2. The NIST Reference Architecture for Intelligent Systems

2.1 The RCS Reference Architecture

To reduce complexity the RCS hierarchical control architecture modularizes the system by using levels of control and templates for processes within levels. A particular feature of RCS hierarchical control is the

Figure 1. The mobile platform, on-board cameras, and monitors of the vision system

repeated use of a standard template and standard interface characteristics for all levels, regardless of controller functionality. This standardization reduces the complexity of designing, building, and operating RCS systems. Further, the modularization enforced by reference architecture guidelines creates interfaces between system functions and between subsystems that enhance extensibility and provide opportunities for standardization, including open-architecture concepts. Standard level names (and shorthand numbers) in RCS are servo(1), primitive(2), elementary move(3), task(4)[3]. The servo level interfaces to actuators and sensors and is the lowest level.

The coarsest modularization of the processes of a level is described by the **control node** template. A control node comprises four control **elements**: behavior generation (BG), world modeling (WM), sensory processing (SP), and value judgment (VJ), as shown in Figure 2. RCS specifies further decomposition of these elements that is beyond scope of this paper. RCS is described in much more detail in [3] and [8].

Control nodes are the building blocks of RCS control systems. Interconnected nodes are arranged in hierarchical levels to implement control functions. Major system design steps then consist of defining system data, defining control system functions and assigning them to appropriate levels and elements, defining interfaces between elements and between sub elements, and allocating all processes to computers and then providing suitable communications. At the lowest level of an RCS system control nodes send commands to actuators and process signals from sensors that provide feedback about the state of the environment.

Each element of the control node template is described below. Characteristics of elements are then further illustrated by describing the architecture of the Vision-based Intelligent Real-time Robotic Vehicle (VIRRV).

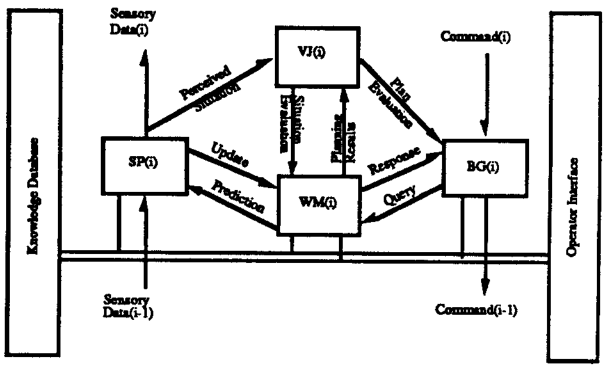

Figure 2. Template for an RCS Control Node. "i" refers to level number.

Behavior generation (BG) processes take a command from a supervisor and decomposes it in real-time into sub-tasks to be performed by subordinate nodes. Any BG may have more than one subordinate, as in a manufacturing cell where the cell controller coordinates a robot, numerically controlled machine tool, and a materials handling device concurrently. Each device has its own controller that may have several internal levels. The BG template specifies three sub-elements that perform spatial decomposition, planning (generation or selection of state graphs for accomplishing goals), and plan execution.

The world modeling (WM) element generates a control node's representation of the external world. This includes dynamic information gathered by processing sensor data as well as static, a priori information such as laws of physics, and robot arm geometry. The world model provides this data to BG processes to be used in decision making and to provide feedback control, and to WM processes of peer and higher level nodes through the knowledge database. Through this sharing, information about the system's environment exists in levels of detail and abstraction that correspond to the levels of task decomposition. Environment data are more detailed at lower levels, more abstract at higher levels. Levels of WM data for vision processing are illustrated by pixels of an image at level 1, the lowest level, recognition of lines at level 2, recognition of surface features at level 3, and recognition of a specific object at level 4.

The world model **knowledge database** (KD) includes static information which is available before action begins, and dynamic knowledge which is gained from sensing the environment and states of the intelligent system itself as action proceeds. Static information includes knowledge about the laws of physics, chemistry, optics, and the rules of logic and mathematics which is represented in the WM functions that generate predictions and simulate results of hypothetical actions.

Sensory processing (SP) modules gather data from sensors and interpret it. The functions of data integration over time and space, and filtering such as Kalman filtering for noise rejection are also performed in SP. Vision processing functions include feature extraction, pattern recognition, and image understanding. As with WM data, SP data is distributed in the control levels with more abstract interpretations of sensory data performed at higher levels.

"**Value judgment** (VJ) elements provide the criteria for making intelligent choices. Value judgments evaluate the costs, risks, and benefits of plans and actions, and the desirability, attractiveness, and uncertainty of objects and events. Value judgments may evaluate risk and compute the level of uncertainty in the recognition of entities and the detection of events. Value judgments can evaluate events as important or trivial. Cost/benefit value judgment algorithms can be used to evaluate plans or steer task execution so as to minimize cost and maximize benefits. " [3].

The Operator Interface (OI) conveys data between a process and a human being who can influence the operation of the process. Graphic user utilities provide ways for the operator to easily generate data and send it to a system, or to access and view the data produced by a system.

RCS systems implement communications between control node elements using standard utilities that provide single writer, multiple reader channels. Details of physical paths and protocols are hidden from the processes.

2.2 Some Current RCS Applications at NIST

The RCS Reference Architecture has been used as a guide in developing many real-time control systems at NIST, some of which are listed here. The Field Material-Handling Robot is a mobile robot that manipulates pallets of ammunition [7]. An Army HMMWV (4 wheel-drive vehicle) is being controlled for teleoperated and autonomous driving [9]. NIST has designed an architecture for automated coal mining [5]. Crane applications such as construction may benefit from the coupling of an RCS controller with a specially stabilized mechanical platform to form Robocrane [4].

3. VIRRV Architecture

The vision-based intelligent real-time robotic vehicle (VIRRV) comprises the following mobility testbed components; the mobile robot, VME single-board computers, a workstation computer used for software development and control, vision systems consisting of video cameras, pan/tilt units, and an image processing system, PIPE [6]. The robot is an electric powered vehicle with an embedded processor for motion control and communications. See Figure 1. VIRRV sensor and control system processors are mounted in a stationary rack. A cable tether to the robot conveys video and RS232 signals.

VIRRV is designed according to formal RCS methodology guidelines to provide an easy-to-use testbed. VIRRV experiments involve developing and testing several image processing algorithms. Experiments test vision system interaction with vehicle control to accomplish collision free, goal-oriented motion and to map objects in the robot space. The current tasks being performed with the VIRRV are robot motion with centering behavior (adjusting motion to equalize visual data "seen" to the left and right of the motion axis), obstacle detection and avoidance, and mapping of objects perceived by the robot sensor systems. Near-future tasks will be to combine sensor guided behavior with generating and executing path plans.

This section describes the VIRRV architecture shown in Figure 3. Since VIRRV was designed using an earlier version of RCS than described in Section 2.1, its architecture does not contain explicit value judgment and knowledge database elements. The functions and information allocated to these elements are distributed primarily among the SP and WM, and partially to BG modules.

3.1 Operator Interface

The operator utilities provide a person an interface to the VIRRV Task level BG and WM processes. The operator interface to BG conveys commands to VIRRV and status from it: logical status of the controller and status of the current task. Through interaction with WM the utilities allow the operator to display data such as current platform location and its recent path, build and display a priori room maps, and build path plans that are executed by the Task level.

There are two modes of operation for RCS processes: operations mode, and development mode. Experiments are run in operations mode with two types of operator interface functions:

- interacting with control nodes. Operator actions include generating commands, selecting operating mode (e.g. single step, remote, local, teleoperation), displaying data buffers.
- reading process data. Operator interface utilities can access the output data of processes for monitoring and troubleshooting by reading global communications channels. The provision of multi reader channels is part of the RCS methodology.

During development testing or error recovery of a process, the operator interface may be used to manipulate all aspects of process operation. These functions are typically performed while automatic processing has been halted: changing modes, generating or changing input data, changing internal data, selectively executing functions, editing startup scripts or data.

3.2 Task Level, level 4

The BG executes an operator command by following either a plan it generated or one generated by the operator, and sending commands to the Emove level. The current VIRRV Emove commands (commands *to* the Emove level) are to go to or through points contained in the Task-level path plan.

The BG module also plans paths from the current robot location to a specified goal in the room. It uses the current map of the room and generates a path so the platform does not collide with known obstacles. It can do the planning before platform motion has begun, and it can replan in the event that the current plan cannot be carried out due to unmapped obstacles.

The WM module maintains a map of the platform's environment in real-time. It combines information from an a priori map, real-time platform location and orientation data, and real-time sensor data describing obstacles. There is currently no task level SP element.

3.3 Elementary Move (Emove) Level, level 3

This level issues commands to the Prim level to traverse path segments assumed to be free of obstacles. The distance range of the commands extends to just inside the range of vision sensors. SP continuously reads data from Prim level and calculates visual "flow" (the transition of intensity information across the camera focal plane due to relative motion between the camera and environmental objects), and then depth (distance to objects) using flow. When depth values indicate an obstacle in the robot path Emove BG sends commands to Prim directing the robot to avoid the obstacle. Obstacle avoidance may cause the platform to depart from a planned path. Emove makes sure that once the obstacle has been avoided the platform goes back to the planned path if possible. If the plan cannot be followed, Emove reports this to Task level which replans a path to the goal.

3.4 Primitive (Prim) Level, level 2

Prim carries out path segment commands from Emove. At the Prim level path segments are assumed to be free of obstacles. Prim BG controls velocity profiles and directions of platform travel. Prim is currently the only VIRRV level where there is spatial task decomposition, i.e., decomposition into tasks to be performed concurrently by two separate servo-level subsystems, the Mobile Platform Controller (MPC) and eye controller.

Prim also controls gaze stabilization necessary for generating valid flow measurements. To stabilize gaze the angle of the forward looking camera is maintained at an absolute heading though the platform may be turning. When a maximum eye angle is reached Prim commands the eye to saccade (move rapidly) to a centered position. The system ignores flow data produced during the saccade.

The SP process is the PIPE Manager, a front-end process that formats data from a high speed parallel vision processor, PIPE[6], and presents it to the rest of the VIRRV system.

3.5 Servo Level, level 1

There are two subsystems at the Servo level, Mobile Platform Controller (MPC) and Eye. They execute commands from Prim concurrently to carry out coordinated actions.

The MPC controls the motions of the platform and processes and publishes the data from dead-reckoning position processing of the platform. In the near future it will interface to vestibular sensors residing on the platform for improving navigation, and gaze and camera stabilization. MPC commands include steer, turn, go at a specified linear speed, and stop. An operator interface allows the operator to take the MPC out of automatic mode, read all external and internal data buffers, and use a joystick to teleoperate the robot.

The Eye is a pan and tilt actuator with attached cameras. Camera data is continuously digitized and processed by PIPE and supplied to Prim level SP. In the future the vestibular sensors may be tightly coupled with the Eye controller for camera stabilization. This would be done by linking the MPC and Eye WM processes.

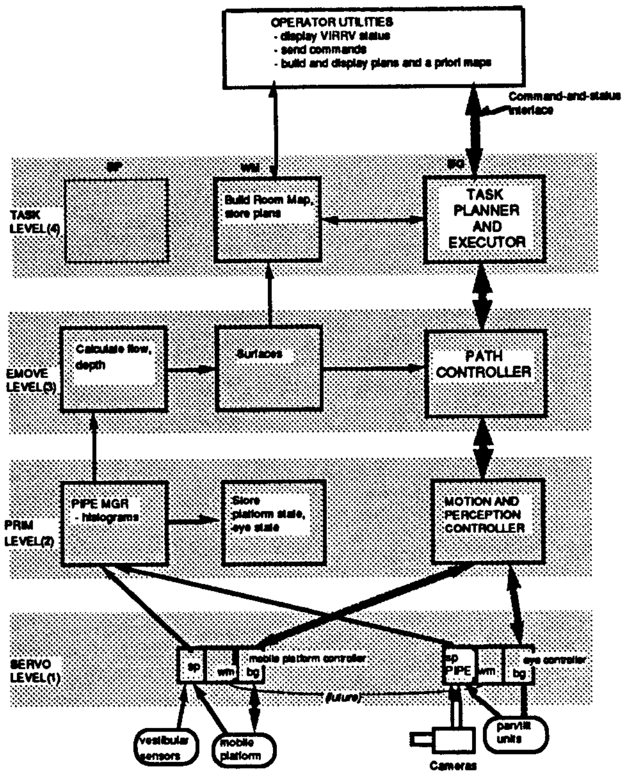

Figure 3 . Logical Control Architecture for VIRRV

4. The Development Methodology for Mobility Testbed Systems

As previously stated the testbed is used to support robotic experiments as well as to define a standard engineering methodology for developing control systems [3] [4]. The methodology describes practices for deriving an application from the reference architecture. The methodology will improve systems development at NIST and will make the RCS reference architecture understandable and usable to developers outside of NIST. Below are some of the methodology elements NIST is addressing.

- The **RCS system development model** is a standard life cycle model that describes chronologically and logically related steps for developers. NIST is emphasizing rapid prototype development in which portions of a system may be implemented before the system design is complete.

- The **RCS control system models** are templates for elements and sub elements used to build up a controller, e.g. SP, WM and BG. Primary issues are what system functions to assign to the elements and how to design the interfaces between them. Standard templates aid consistency and understandability of an architecture, and facilitate use of tools and reuse of software.

- **Standard inter-process communications** includes a standard application process interface for code portability that provides an open architecture, and utilities that implement inter-process and inter platform communications with guaranteed performance.

- **Process scheduling** techniques must ensure real-time execution, with measured performance evaluation so that designers can allocate processes to processors and detect and reallocate processes which violate timing requirements.

- **Software Configuration Management** is necessary when teams of people are working together, to ensure code consistency, and to preserve useful system configurations for tests and demonstrations. People need to be able to easily find source and executable code, and understand code written by others. Testbed practices cover directory structures, policy for maintaining the working code directories, and coding practices (for the C language).

- **Testing procedures** promote stepwise system testing to promote efficient and productive testing. Guidelines include test design, execution, and documentation.

- **Formal design reviews** will be conducted in the near future to ensure compliance with the reference architecture, to foster standard practices, to detect design errors, and to educate staff on useful engineering techniques.

- Definition of a standard **vocabulary** addresses the main problem of subtle differences in understanding of terms. When meaning is imprecise people do not communicate, or they spend a great deal of effort explaining terms and concepts. Concise, precise terminology is essential for groups of people to work on complex systems.

IEEE standards for software engineering and system development are being incorporated into the testbed guidelines where applicable.

5. Summary

NIST researchers gain several benefits by using Testbed resources and the consistent approach to control architecture provided by the reference architecture. Developers outside of NIST can also gain these benefits by using a reference architecture and a corresponding methodology for system development.

The most important benefits and results are:

• NIST staff can develop systems faster, more efficiently, and produce better quality systems by using tools, methods, and documentation available in the testbed.

• researchers can "reuse" hardware and software components developed in other projects.

• subsystem specialists need not become system architecture experts to build their own systems or to do experiments

• designs are easier for others to understand and work on since architecture principles and practices are standard. Researchers gain from a pool of lessons learned among several projects.

Mobility testbed efforts in the near future will include the following:

• Continue to define details of the methodology and of the RCS reference architecture and document them so that NIST researchers can build conforming applications and others can easily understand and use RCS.

• Develop computer-assisted and automated tools for system design, code generation, verification and validation, and simulation. These are needed to free developers of details of implementation, to enforce the design principles, and to test designs for correctness and reliability before they are implemented.

• Continue to experiment with process reallocation between processors and process scheduling to explore issues of communications, code portability, open architecture, and measurement of real-time performance

• Demonstrate multilevel replanning. The VIRRV Task level does path planning (and replanning in real-time), and the Emove level plans for obstacle avoidance. This would occur when an unmapped obstacle is detected at a planned way point. Because the robot cannot traverse the way point, Task level stops executing the current path plan and generates a new one based on the updated room map.

• Work on vision processes and algorithms for mobile robot behavior including combined vision processing (obstacle avoidance plus centering), and mapping using multiple sensors. A vestibular sensor system, including a gyroscope, compass and accelerometers, will be investigated for robot motion control and camera and gaze stabilization. To achieve total robot autonomy the physical tether will be eliminated by putting batteries on-board to power sensors and processors.

References

[1] Albus, James S.; McCain, Harry G.; Lumia, Ron *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM);* NIST Technical Note 1235, 1989 Edition; April 1989. 76 pp.

[2] Albus, James S. *Outline for a Theory of Intelligence*; IEEE Trans. on Systems, Man, and Cybernetics; Vol. 21, No. 3; May/June 1991.

[3] Albus, James S.; *RCS: A Reference Model Architecture for Intelligent Control*; IEEE Journal on Computer Architectures for Intelligent Machines; May 1992.

[4] Bostelman, Roger V.; Dagalakis, N.; Albus, J.S. *A Robotic Crane System Utilizing the Stewart Platform Configuration*; proceedings of the ISRAM '92 Conference; Santa Fe, NM; November 10-12, 1992.

[5] Huang, Hui-min; Quintero, Richard; Albus, James S. *A Reference Model, Design Approach, and Development Illustration toward Hierarchical Real-Time Control System for Coal Mining Operations.* Control and Dynamic Systems: Advances in Theory and Applications, Volume 46: Manufacturing and Automation Systems: Techniques and Technologies, Part 2 of 5; C.T. Leondes ed; Academic Press; 1991.

[6] Kent, Ernie. W.; Schneir, M.O.; Lumia, Ron *Pipe*; Journal of Parallel and Distributed Computing, 1985.

[7] McCain, Harry G.; Kilmer, Roger D.; Szabo, Sandor; Abrishamian, A. *A Hierarchically Controlled Autonomous Robot for Heavy Payload Military Field Applications*; Proceedings of the International Conference on Intelligent Autonomous Systems; Amsterdam, the Netherlands; December 8-11, 1986.

[8] Quintero, Richard, Barbera, Anthony J. *A Real-Time Control System Methodology for Developing Intelligent Control Systems*; NISTIR 4936; October 1992. 72 pp.

[9] Szabo, Sandor; Scott, Harry A.; Murphy, Karl N.; Legowik, Steven A.; *Control System Architecture for a Remotely Operated Land Vehicle*; proceedings of 5th IEEE International Symposium on Intelligent Control; Philadelphia, PA; September 1990; 8 pp.