A Robot Control System Based on FORTH

John L. Michaloski
and
Barry A. Warsaw
National Bureau of Standards
Gaithersburg, MD 20899

For over 10 years, the National Bureau of Standards has conducted research in robotics. NBS has developed a Real-Time Control System (RCS) that uses the sensory-interactive hierarchical control model. RCS has been implemented by a small team of programmers and can control a variety of robots (Photo 1). Suggestions and theories detailing the needs of a fourth-generation robot controller have evolved from the implementation of RCS (1,2).

The goal of the robotics research at NBS has been to use the hierarchical model to establish standardized interfaces between successive levels of the task decomposition and sensory-interactive knowledge building. These interfaces would allow communication among levels to consist of a list of agreed-upon commands and status handshakes. With the goal of standardized interfaces realized, control levels developed by different interests could communicate via this protocol.

In robot hierarchical control, tasks are decomposed into simpler and simpler tasks. Eventually, the tasks are decomposed into low-level control commands for the robot. As the tasks for each level are decomposed, sensory-interactive knowledge building is occurring simultaneously. Low-level sensor information is gathered and combined into higher and higher levels of knowledge about the current state of the robot's environment. This feedback is used by the hierarchical control system to compare the actual state vs the expected state to provide control commands for the next command cycle.

Much of the initial work involved integrating a structured light vision system to work with the control system (3) (Photo 2). Current work has emphasized the goal of integrating robot functions as a part of

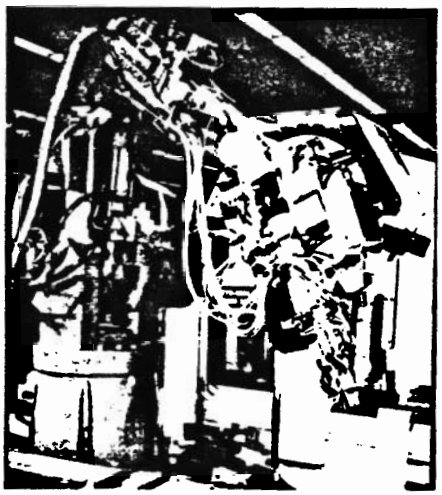


Photo 1. Among the robots in the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards is a Cincinnati Milacron T3, used in the horizontal workstation.

an Automated Manufacturing Research Facility (AMRF) (4,5). The AMRF is a demonstration facility developed to serve as a research tool for studying measurements and standards in the automated production of machined parts. The control architecture developed for the AMRF is an extension of the hierarchical robot control system theory. The AMRF has proved that the concepts established by RCS are not limited to robot control systems, but pertain also to a broad body of programming endeavors.

A REAL-TIME CONTROL SYSTEM

The RCS software has been implemented with FORTH as its base coordinating and development system. Any number of programming languages can be used to solve the wide variety of problems associated with robotics, but some are better suited than others for certain tasks. Assemblers are best for low-level, high-speed hardware interfaces; block-structured compiled programming languages such as C or Pascal are good for mathematically intensive computation; and a high-level symbolic programming language such as LISP or PROLOG is better for logical reasoning and planning. The choice of FORTH was a compromise between a desire to satisfy all robot programming requirements and the integration problems of implementing a control system.

FORTH is both a computer programming language and a philosophy of software development. It opens up all levels to the user with complete and explicit access. It includes an interpreter, a compiler, an assembler, and a multitasking capability, all existing within a highly interactive atmosphere. FORTH breaks away from traditional batch computer programming; instead of a cycle of compiling, linking, loading, executing, editing, compiling, linking, etc., FORTH software development uses small, verifiably correct modules that are incrementally compiled. This iterative style of software development improves programming productivity.

With the increased speed of development, FORTH makes the fine tuning of design easier (6). Because it is so easy to develop a working implementation of a program, results from the prototype can be compared to the design expectations. Depending on the analysis, the design will be reevaluated or further refined. In traditional software projects, the initial prototype is usually the finished product. With robotics applications, the need to fine tune code is imperative.

FORTH has a symbol table, known as a dictionary, that keeps all the variables

and routines, called words, in the system. Newly compiled words are added to the dictionary as combinations of existing words. Because FORTH is written in itself, system modifications to meet a functional requirement, common in a robot control system, are not difficult. FORTH's extensibility allows new control and data structures to be developed to match a particular programming need. This creativity has led to the development of new robot-specific programming constructs, which simplified the software. Although much of the RCS is not strictly FORTH, many of the language's programming philosophies have been maintained or enhanced. RCS adapted the FORTH model to meet the needs of a robot control system. A study of the impact of FORTH on some of the functional specifications that guided the development of RCS will further explain why we selected it.

Real-Time Requirement. An original RCS implementation goal was real-time performance within a cost-effective control system. The hardware consists of a multimicroprocessor system with the different levels of control and support processes residing on multiple boards. The boards operate in parallel and communicate through defined interfaces, including the commands and status that are defined in a common memory map. Actual communication is controlled by an independent process that synchronizes the data exchange between hierarchical levels.

A major functional requirement of RCS is the output of new control signals for the robot in real time. RCS updates to the individual commercial robots vary, but range from a new position every 25 ms (or 40 new poses/second) to a new position every 100 ms. In order to meet the demands of real-time control, the control system was partitioned among these processor boards.

The constraints of real-time performance and multiboard communication had a profound effect on the software requirements. Typically, interprocess communication is carried out through shared data, a process that presents both hardware and software problems. Where to put the shared data, and how a process must perform the read/write handshake must be resolved to ensure proper communication. Exploiting exact knowledge of the absolute addresses for the communication channels simplifies the task.

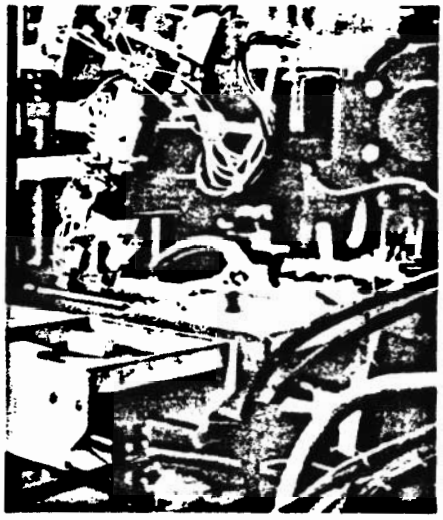


Photo 2. The robotic vision system and the interchangeable grippers are AMRF's additions to the T3 robot.

Understanding system operation is not difficult in FORTH. There is a simple layer of a virtual machine that resides on top of the operating hardware, and the mapping between a logical and a physical address is straightforward. Mixing high-level code with assembly code is also simple. The programming environment provides an on-line assembler for quick and easy machine or hardware access. Interactively testing assembly code further simplifies a hardware-intensive project. Finally, FORTH provides the feel of assembly-level closeness to the machine, yet provides the flexibility of a high-level language.

RCS Reliability. The RCS system was designed to avoid performance degradation due to hardware troubles; failures that require troubleshooting can undermine any unprepared operating system environment. For example, a typical interface to a hardware device has a process awaiting a response. If this polling is embedded in a lower level process, the system performance can come to a standstill.

Because of its dependence on sensor hardware, a robot programming system must be prepared for these problems. RCS incorporates a software design methodology that forbids embedded loops and other types of control structure that would allow a hardware failure to cause the system to hang. Instead, an entire pass through the system is performed each cycle. However, no matter how hard the programmer attempts to anticipate a hardware malfunction, an error status will propagate up the levels of control and must be handled. Taylor (7) has pointed out that in a robot application such as assembly, only 10 percent of the robot commands are for motion; the majority are for handling I/O and the "anticipated but unpredictable" problems. During robotic software development, there is a strong need for testing and debugging aids that diagnose hardware failure symptoms.

As it turned out, the software requirements of RCS were eased by having the control system on a dedicated, not a general-purpose, machine. RCS is thus not required to provide user protection mechanisms associated with most operating systems. The layers of protection provided by such systems may help with multiuser timesharing, but hinder development under a robot control system in that each layer of protection hides more and more of the operation from the programmer.

RCS Support Tools. Although anyone can "program" a fifth-generation robot, most people don't realize the amount of processing required with even a simple task. Programming a robot requires accounting for so many details of operation that understanding the system supercedes how to program the application. The RCS approach has been that the programming environment is as important as the actual control algorithms. The RCS approach is to provide a system to the user that incorporates as much knowledge as possible about the current state of the robot, and to that end has established a tighter relationship between the system support and the control alogrithms.

Understanding complex robot applications demands that a relationship between the system support, including the source code, and the control algorithms be constantly maintained on line. The FORTH on-line global dictionary of the variables and words present in the system provides direct and immediate access to all parts of the system. RCS extends this idea and assigns a variety of modes that depend on defining type with each word in the system. For example, FORTH provides a LOCATE feature that ties a dictionary word to a source disk location. When the user wishes to edit a word, referencing the word with editing intentions fetches the location on the disk, as opposed to remembering an exact location of the word in what file on disk. Further, RCS adds modes of seman-

tic meaning categorized by data type to simplify programming.

Program Correctness and Troubleshooting. In a robotics application, simply proving program correctness is not enough: program correctness deals only with sequential programming, and robotics contains numerous nondeterministic activities. RCS uses a synchronized clock in communications in an attempt to regulate the nondeterministic activities, but even so, intermittent errors caused by unforeseen timing side effects and hardware failures regularly occur in real-time applications and make software testing a tough proposition.

The interactive testing atmosphere of FORTH and RCS simplifies dealing with problematic hardware testing and debugging. The dictionary entry to program a simulated fifth-generation robot to sweep a floor would be as follows:

```
:SWEEP-FLOOR door GOTO
  door OPEN broom GET
  door CLOSE floor SWEEP;
```

But in robot control the anticipated behavior does not always match the actual behavior. If the robot runs into the door, the cause of the failure must be pinned down. FORTH and RCS provide single-step capabilities that help pin down an error sequence. For example, each command can be executed independently to monitor robot behavior. Trying again, *door* is pushed onto the stack and then *GOTO* is interpreted separately. The discovery that this works leads to the test of *door OPEN*. This execution causes the robot to crash into the door. Upon study, the determination was made that a push-through type of door was chosen, not a door with a handle. Loading a new door prepares the robot for another try.

This time *SWEEP-FLOOR* causes the robot to slam the door, requiring a fine tune of the system. A simple extension to FORTH, highly used in RCS, allows any lower level word embedded within the code to be reassigned a new execution sequence because of the availability of indirection. Thus, all higher words defined after *CLOSE* will use the original *CLOSE*, but the indirection will provide the location of the new code.

With some tinkering, a new *CLOSE* will cause the robot to close the door quietly. The fine tuning has been achieved with a minimum of overhead not available in a compiled or downloaded system.

REDUCING COMPLEXITY

Given the complicated nature of robot programming, simply helping with program debugging is akin to handing a pail to the captain of the Titanic. Reducing the software complexity is also required, and one way to achieve this is by reducing the amount of the software through modularization and information hiding. Several programming practices can also help, such as factoring, extensive data and control structure design, and abstract data typing in a class structure architecture.

Factoring. FORTH and RCS promote compact and modular design through expression factoring in which common portions of several different expressions are factored out into a separate expression. Imagine a FORTRAN program containing lengthy formulas. Within three of the formulas, a common expression is calculated each time. Software elegance mandates that the common expression be factored out and represented as a separate expression. The program gains execution speed and software verbosity is reduced. Threaded interpretive languages thrive under this mechanism; not only are mathematical expressions factored, but so are entire algorithms, both small and large.

In FORTH and RCS, a new word can be used to factor recurring larger programming constructs. After a new word extracted from several sources is defined it can act as a building block, and the higher level words can replace the existing code with this word. The FORTH system, as well as application programs, are very compact, and these small modules are easier to verify.

Extensibility. Extensibility allowed the development of many enhancements to accommodate the functional needs of a robot control system. One of the initial extensions was SMACRO, a structured language based on superassembler macros that translates source code directly into assembly. SMACRO was developed to bridge the gap for robot operations that are time-critical, while offering a structured programming control application environment. The SMACRO extension provides FORTH with block-oriented structured language.

SMACRO allows a user to write structured code within a FORTH programming environment. The programs resemble those written in high-level languages like Pascal, C, or PL/I. These macros generate machine instructions directly executed by the host processor. In general, this object code produced by SMACRO is faster than that for an equivalent structured language program. Further, SMACRO performs direct translation that maintains a simple logical-to-physical addressing layer that is normally lost when executing code of a compiler code generator. Finally, SMACRO maintains FORTH's interactive atmosphere plus allowing FORTH words to mix with the SMACRO machine-generated code.

Most of the control algorithms that are mathematically intensive use SMACRO and its floating-point enhancement, FSMAC. FSMAC provides basic floating-point operators, again executing directly on the math coprocessor chip. In addition, operators important to robot kinematics were added, such as vector operations including vector normal, cross multiply, dot product, and matrix operators.

An example of FSMAC will illustrate FORTH's extension to accommodate new arithmetic operators. The following is a fragment of robot kinematic code:

routine CALC-xyz

(Calculate distance from xyz to x'y'z' using directional unit vector to give scaling direction back down the arm.)

pose {*dxdydz*} .*V**S. *yr* .*V*=>. dx'

pose {*xyz*} .*V*−.dx' .V=>. x'

end-routine

The first line contains the .V*S., or vector times scalar binary operator. The vector pose {dxdydz} represents the orientation unit vector in the x direction of the robot pose, which is a three-element, contiguous floating-point array. This orientation vector is multiplied by the scalar length, yr, which is a constant representing the distance from the yaw point to the roll point on the robot. The second line contains .V−. or vector subtraction. The wrist point vector pose {xyz} is subtracted by the amount of dx to derive the amount of distance down the arm to the wrist plate.

Abstract Data Typing. Sophisticated data structures relating to robotics can be developed in FORTH. Within the realm of robotics, the requirement of an object-oriented perspective cannot be denied. The work of a pick-and-place robot, for example, is to transfer an object from here to there. Categorizing the data and the functions associated with it is one way to simplify the problem. This type of methodology is known as an object-oriented approach, as opposed to the procedure-oriented approach of languages such as Pascal and FORTRAN.

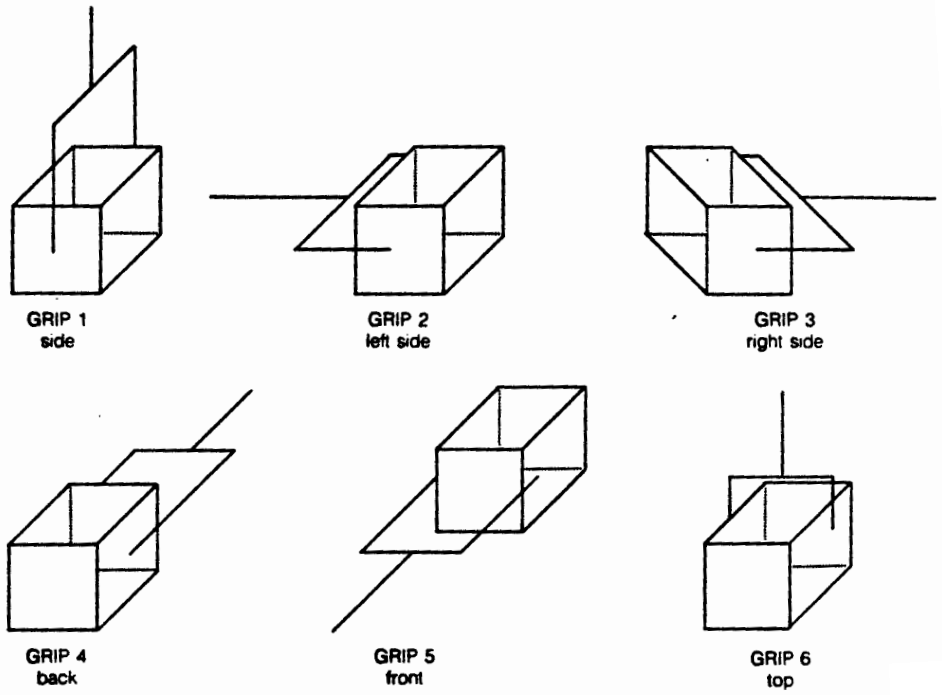


Figure 1. Robotic manipulation of a solild block requires the use of six "gripper rules."

Using CREATE/DOES>, FORTH can achieve object-oriented structures. For example, assume a robot is involved in a reasoning task about children's spelling blocks. The generic block can be considered an abstract data type with six faces to contain letters. The operations available would be based on the rules in Figure 1.

Reasoning would direct the robot to change the orientation of certain blocks so the letters on their forward-facing sides spelled a word.

In object-oriented structures, the way in which these operations are implemented is less important than the ability to declare an instance of the object and then be able to use the operators included with the object. Each declared instance of a spelling block would contain a different letter on each of its faces. For example, to declare two different blocks:

"N" "F" "E" "L" "T" "A"
SPELLING-BLOCK BLK1

"Q" "B" "S" "P" "K" "R"
SPELLING-BLOCK BLK2

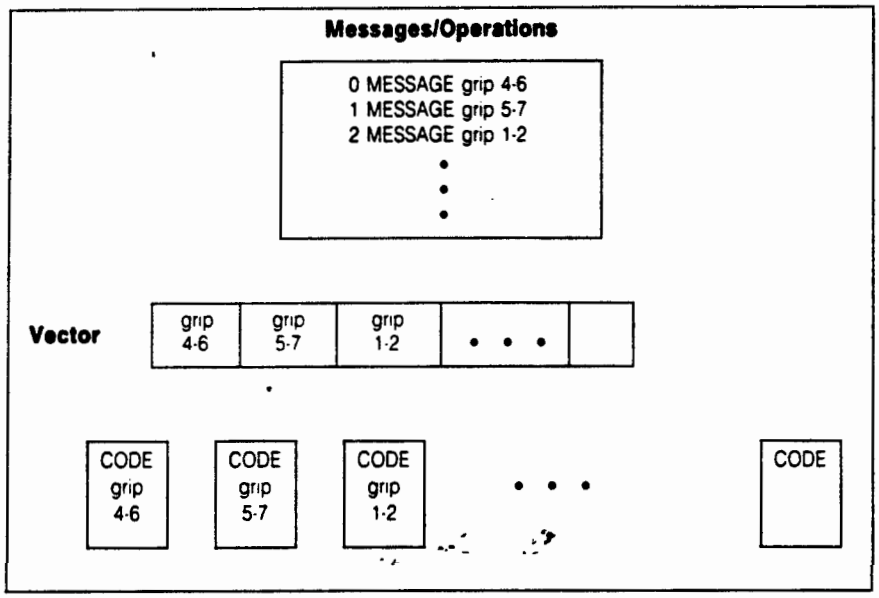
Since both BLK1 and BLK2 share the common operators available with SPELLING-BLOCK, the type of architecture in Diagram 1 (next page) is possible within FORTH.

CONCLUSIONS

The amount of information required for robot control is staggering; thousands, if not hundreds of thousands, of programs, variables, and decision processes, and megabytes of source code, are present in the system. The burden of keeping track of so much data should be transferred from the human brain to the computer. Only with the help of the proper programming environment can sophisticated robot applications be realistically developed.

Robot software development can be easy with FORTH, due to three important features inherent in the language. First and foremost is the interactivity that allows the user to test out routines, do data dumps, or just plain experiment with the system and get immediate responses. Another feature is that routines are usually small and easily verifiable, and tackle only a very specific part of the problem. A routine does not try to handle too much and programs are written hierarchically. FORTH is well suited to top-down design, bottom-up development. The third feature is that

Diagram 1
Class of Spelling Block



FORTH encourages the fine tuning of design through a process of iteration and factoring.

Further information on FORTH is available from the FORTH Interest Group, P.O. Box 8231, San Jose, CA 95155.

REFERENCES

1. Barbera, A.J., M.L. Fitzgerald, J.S. Albus, and L.S. Haynes. "A Language Independent Superstructure for Implementing Real-Time Control Systems," *International Workshop on High-Level Computer Architecture*, May 1984.
2. Fitzgerald, M.L., and A.J. Barbera. "A Low-Level Control Interface for Robot Manipulators," *NBS-Navy NAV/SIM Workshop on Robots Standards*, June 6–7 1985.
3. Albus, J., A. Barbera, J. Evans, and G. VanderBrug. "Control Concepts for Industrial Robots in an Automatic Factory," *Society of Manufacturing Engineers Technical Paper MS77-745.*
4. Furlani, C.M., and E.W. Kent. "The Automated Manufacturing Research Facility of the National Bureau of Standards," *Summer Simulation Conference*, Vancouver, B.C., July 11–13, 1983.
5. Simpson, J.A., R.J. Hocken, and J.S. Albus. "The Automated Manufacturing Research Facility of the National Bureau of Standards," *Journal of Manufacturing Systems*, vol. 1, no. 3, 1982.
6. Harris, K. "The FORTH Philosophy," *Dr. Dobb's Journal*, no. 59, September 1981, pp. 6–11.
7. Taylor, R.J., P.D. Summers, and J.M. Meyers. "AML: A Manufacturing Language," *International Journal of Robotics Research*, vol. 1, no. 3, 1982.

John L. Michaloski and Barry A. Warsaw are computer scientists in the Real Time Control Group of the Robot Systems Division at the National Bureau of Standards.

Reader Feedback

To rate this article, circle the appropriate number on the Reader Service card.

4	14	24
Excellent	Good	Fair