Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

A GENERIC ROBOT SIMULATOR

D. L. Kimbler, Ph.D., P.E.
Associate Professor
Department of
Industrial and Management Systems Engineering
University of South Florida
Tampa, FL 33620

ABSTRACT

A significant problem in both educational and industrial systems is access to development systems. Universities typically have neither funds nor space for a broad variety of industrial robots. Few manufacturing concerns can afford the luxury of robots dedicated to system development and training.

An alternative to industrial robots is a physical simulation system. This system should include a robot capable of a broad range of motions and a controller capable of emulating robot controllers. The key to this system is its software, which provides a structure which allows the system to be tailored to specific needs.

Such a system is a large undertaking. This report describes a beginning in developing this system. While the resulting system is specific to a single robot, it is expandable and adaptable to other robot families.

INTRODUCTION

Of all kinds of simulation, physical simulation is one of the most versatile. More than just an iconic model, the typical physical simulation uses a control system to provide a realistic decision process and uses a functioning physical system to demonstrate the results of control actions. A physical simulation may allow large complex industrial systems to be studied with less investment in equipment and space. This has led to extensive use of physical simulation in educational institutions and during planning stages of industrial plant expansion and renovation.

The generic robot simulator is an attempt to expand physical simulation in robotics to allow several different robots to be simulated by the same system. This system would allow the user to select a robot from a computer menu. The selection would then enable that robot's command language, letting the user develop programs in a realistic software environment. Program execution would then drive a small-scale robot, simulating operation of an industrial system.

This implementation of a generic robot simulator is based on a generic control language [1] and a scaled down working envelope. The generic language is a robot control language which could be used on its own. High level languages are compiled into the generic language for execution. The scaled work envelope allows portability of programs from industrial robot to simulator, with no modification to point definition.

The system described here has both advantages and disadvantages. The software is written in BASIC, allowing ease of portability and extension. A result, however, is also slow execution and limited program size in a microcomputer installation. The small scale robot selected is the MICROBOT Minimover-5. [2] An obvious disadvantage is that only a half sphere of envelope can be simulated, and the speed range cannot duplicate industrial articulated arms. This is one result of physical simulation. Comprimises are made at the expense of fidelity where the alternative is exact duplication, not simulation. The only serious lack of fidelity in this system is the use of an articulated arm, which reduces the effect of simulated cylindrical systems and specialized arm configurations. The software is also the result of compromise, in that not all commands can be adequately simulated. From these compromises, however, comes a system that can be used effectively to simulate a production system at a fraction of system cost and space.

THE GENERIC LANGUAGE

The generic language is a set of operation codes that can be used to control a robot. This language is an intermediate level language comprised of 23 primitive commands in four groups: motion, point data, modification, and program control. These commands cause direct motion, control hand opening, orient the robot in its cartesian system, control execution, and display data. The result is a language capable of directing motion, branching, integer addition, conditional execution, and communication.

All calculations are performed in a five cell accumulator, allowing point data to be stored for five axes. The program counter keeps track of the command in execution. There is also a stack, to hold program counts and subroutine returns. The program itself is a sequence of decimal operation codes and their parameters. Execution is a series of subroutine calls based on operation code, with the program counter incremented in the execution routines. The result is a very simple but effective execution language.

INSTRUCTION GROUPS

The four instruction groups provide all commands necessary for simple robot control. These commands are stored as numeric codes followed by numeric arguments in an array OBJ. A pointer OC identifies the index of the array containing an executable code. Point definitions are stored in cartesian form in array P, with a corresponding array PNT$ holding point names. User variables such as counters are stored in array VAR, with names in VAR$.

Four motion commands are used for arm control. MOVE directs motion to the point stored in the accumulator. STLINE approximates a straight line by executing MOVE over a series of short distances. HAND code is followed by an argument which specifies the size of the hand opening, and can cause an open or close. OHAND has as an argument a hand opening width, which is tested against the actual opening.

Point data commands manipulate coordinates and orient the robot. LOAD causes a specified point to be put in the accumulator. STORE takes them from the accumulator and puts them in the point array. CHANGE has three arguments, and modifies the x, y, and z components of the accumulator. NDIR is used to find the current wrist direction cosines of the point in the accumulator. NEAR uses these cosines to find the axis nearest to the tool angle. COORD has an offset along the tool angle as an argument, and defines a new point an offset distance along the tool ray. CTOOL and CBASE are used to modify points by applying tool and base transformations to succeeding moves.

Modification commands deal primarily with character strings and variables. VALUE calculates the value of a numeric argument or a variable name. STVAR retrieves a value from the accumulator for storage, PVAR converts it to a string, and TYPES displays it on the screen. CSPEED is used to set a new motion speed.

The primary execution control command is CHIN, which sets the object code array index to the argument of CHIN, thus allowing direct branches. Branching logic is supplied by TEST, with its arguments of relational operator and the index for next command if the condition is true. Subroutines are implemented using STACKA, which increments the stack pointer and pushes the return index, and STACKL, which decrements the pointer. Execution is terminated by QUIT, and short pauses are executed using TIME.

Execution proceeds by inspecting OC and executing a module identified by OBJ(OC). This proceeds sequentially until the code for CHIN, STACKA, or STACKL are encountered. Various commands place or retrieve items in the accumulator. TEST, for example, expects the first two elements to hold values for comparison. These values are put there by LOAD. A message may be output by repeatedly LOADing characters and executing STVAR, PVAR, and TYPES.

SYSTEM TRANSLATION

The translator is presently programmed for VAL [3], the command language for Unimation robots. Selecting the PUMA 600 menu option sets scaling factors and arm limits for the PUMA robot. Points are checked for validity in both the PUMA and MICROBOT arm envelopes. Expansion of this system to other robots would require tables of arm limits for point validation.

Compiling proceeds in two steps. The first pass searches for recognizable source language names and converting them and their arguments to generic code. This pass also includes syntax checking. The second pass searches for control transfers and labels, and adds this information to appropriate generic operation codes. After compiling, error messages are printed and control is passed back to the menu level.

Translation in the first pass consists of successive calls to subroutines which parse high level commands and store operation codes and arguments. The VAL translator consists of 34 modules, allowing almost all VAL programs to be executed. Several VAL commands are not supported due to complexity, infrequent use or both. The VAL translation table is shown in the Appendix. As shown in the table, very useful programs can be constructed. VAL commands with no generic equivalent are accepted but ignored. Illegal commands result in error messages.

One aspect of VAL not simulated is its operating system. The program itself is a sort of menu driven operating system, allowing disk storage and retrieval, program editing, point teaching, and execution. During point teaching the arm is controlled from the keyboard. When the arm is in a desired position, the point is identified by name and its coordinates stored. Points may be recalled by name and altered after being taught. Files are stored as character strings, with all program lines followed by point definitions.

BASIC IMPLEMENTATION

The initial implementation of this system is in Applesoft BASIC in an Apple II +. [4] The Minimover - 5 is interfaced using the MICROBOT ARMBASIC firmware card, which adds robot commands to BASIC. This implementation is fully operational, but it has two disadvantages -- speed and size.

Execution speed is limited by the translation of BASIC code. This is compounded by the motion limit tests that require several calls to transcendental functions to verify legality of each move for both PUMA and Minimover - 5. As a result, there is a discernible pause between moves, while computation for the next move takes place. Compiling is a potential solution. The size of the program, however, prevents compiling it as a unit. Compiling would also make it more difficult to modify and extend the system.

Size is also a problem in extending the system to other robots. At the outset of the project the intention was to include several robots in the system. This is no longer feasible. Instead, extension to other systems can be done by replacing translation routines, leaving the execution system intact. The result would be a collection of programs with a common structure, rather than a single large program.

The speed problem has been reduced by adapting the software to a Zenith Z-100 microcomputer and a MICROBOT Teachmover. The Teachmover is mechanically identical to the Minimover - 5. It has the advantages, however, of serial communications by RS-232, allowing it to be used with a broader range of computers. It also has a teach pendant and can be used without a computer, making it more versatile in a laboratory.

The use of an 8/16 microcomputer, even in interpretive BASIC, is an improvement. A further improvement can be had by compiling, which is possible on the Z-100 with sufficient memory. After compiling, there is no discernible hesitation between moves, and overall system operation is improved.

With minor changes, notably in data communication statements, the Zenith version has been installed on an IBM-PC in BASICA. In addition, the program has been modified to use the MICROBOT Alpha, the larger industrial counterpart of the Teachmover. To add more realism to the Teachmover, an I/O module with four inputs and outputs has been developed. This module can be used to operate small motors and solenoids, and receive discrete inputs such as switch closures. This module is also used with the Minimover-5

The system is presently operating in several configurations. Using an Apple II+ with ARMBASIC and Apple Super Serial interfaces, the software can use a Minimover - 5, Teachmover, or Alpha. The Zenith Z-100 and IBM-PC interface serially to the Teachmover and Alpha. In unrelated research an Alpha was controlled by a Hewlett Packard 9816. It is expected that adaptation to other microcomputers would be simple, assuming that sufficient memory and a serial interface controllable from BASIC were available.

CONCLUSION

This research began as an attempt to develop a useful and flexible way to simulate industrial robots in an academic environment, as a means of exposing students to a variety of systems at low cost. While the only system completed is the VAL System, the expandable structure is there. As a VAL simulator, the project has been successful. The system has been used at the University of South Florida with good results in student robotics laboratory assignments. One valuable result is that our PUMA is now available for more research while increasing course offerings in undergraduate robotics instruction.

The development and expansion of the system has also been valuable for the graduate and undergraduate students involved. The availability of a system that has so many characteristics of industrial robots yet is available and open to experimentation is valuable indeed. At various levels of detail, this system has been used by students from the high school to doctoral levels.

Finally, this system has been valuable to the author for its defects as well as its good points. Several lessons learned have yet to be implemented, but are expected to be highly useful. Many techniques used and techniques learned are readily transferable to industrial systems. As modular, interconnected robot systems proliferate, these techniques will become very useful.

ACKNOWLEDGEMENT

The original program upon which this report is based was done by Thomas A. Lacksonen during his MS work at the University of South Florida. System enhancements and expansion were developed by Dulio Furtado and Cesar Malave, doctoral students at USF. Their contributions to the success of this project are gratefully acknowledged.

APPENDIX

The following commands are grouped according to function, as presented in the body of the report:

Name	Function
MOVE	Move to point in accumulator
STLINE	Interpolated straight move
HAND	Open hand
OHAND	Find hand opening
LOAD	Load point or numeric data
STORE	Store data in table
CHANGE	Modify accumulator
WDIR	Find hand direction
COORD	Find offset along DIR ray
NEAR	Identify nearest axis
CTOOL	Change tool transform
CBASE	Change base orientation
VALUE	Convert string to numeric
STVAR	Store accumulator value
PVAR	Convert numeric to string
TYPES	Put string on screen
CSPEED	Change arm speed
CHIN	Modify code pointer
TEST	Branch on condition
STACKA	Push address and branch
STACKL	Pull return address
QUIT	Stop execution
TIME	Pause

These commands are converted to VAL commands by the following correspondence:

VAL	GENERIC
MOVE	Load, move
MOVES	Load, Stline
DRAW	Load, Change, Move

ALIGN	Load, Wdir, Near, Move
APPRO	Load, Wdir, Coord, Move
APPROS	Load, Wdir, Coord, Stline
DEPART	Load, Wdir, Coord, Move
DEPARTS	Load, Wdir, Coord, Stline
READY	Load, Wdir, Coord, Move
OPENI	Hand
CLOSEI	Hand
GRASP	Hand, Ohand, Test, Chin
SETI	Value, Stvar
TYPEI	Pvar, Types
HERE	Load, Store
SET	Load, Store
SHIFT...BY...	Load, Change, Store
TOOL	Ctool
GOTO	Chin
IF...THEN...	Value, Value, Test, Chin
GOSUB	Stackq
RETURN	Stackl
STOP	Quit
DELAY	Time
SPEED	Cspeed
BASE	Cbase
TYPE	Types
SIGNAL	Commo
WAIT	COMMI

The following commands are allowed but ignored (commands not listed are typed as errors):

```
REMARK         RIGHTY
ABOVE          LEFTY
BELOW          FLIP
       NOFLIP
```

REFERENCES

1. Lacksonen, Thomas A., Generic Robot Simulator with Microbot and Microcomputer, M.S. Thesis, University of South Florida, Tampa, Fl. 1983. 95 pages.

2. Hill, John, "Introducing the Minimover-5," Robotics Age, 2, 2, 18-27, Summer, 1980.

3. Unimate PUMA Robot Manual 398H. Unimation, Inc. Danbury, Ct. 1980.

4. Applesoft II: Basic Programming Reference Manual, Apple Computers, Inc., Cupertino, CA. 1978.