The Use of FORTH in the Instruction of Introductory Robotics

Alan J. Cotterman *Daniel M. Willeford* *James E. Brandeberry*

Department of Computer Science
Wright State University
Dayton, OH

Abstract

At the Digital Control and Robotics Laboratory at Wright State University, students use an implementation of fig-FORTH to design a teach mode control program for modified toy robot arms. This program allows the arm to be taught multiple trajectories of up to 100 points (unique arm positions) and then permits repeated execution of these trajectories. The program successfully realizes real-time position monitoring, motion-execution algorithms, management of the trajectory database and features a menu-driven, user-friendly operation.

The hardware environment consists of the host PDP-11/34 minicomputer operating under RSX-11, and four LSI-11 microprocessor workstations. TASK4TH, a standalone derivative of Fig-FORTH, is down-line loaded from the host into the workstation and is capable of supporting peripheral equipment which includes serial and parallel I/O and also analog-to-digital and digital-to-analog conversion. Toy robot arms with five degrees of freedom are connected to the workstations and have been modified to operate under computer control. Position detection is accomplished through analog-to-digital conversions on the outputs of Hall-effect sensors (mounted on the axes of rotation) and arm motion is accomplished by actuating solenoids via the parallel port.

Introduction

The Wright State University Robotics Laboratory employs FORTH to offer senior level engineering students a unique opportunity to develop control programs for a small robot arm. In addition to being easy for students to learn, FORTH has interfaced easily and effectively with our system's extensive feedback controls and motion controlling mechanisms. This paper discusses the laboratory programming environment, surveys the programming projects done by students, and offers a brief evaluation of the environment and the use of FORTH in this application.

Laboratory Environment

The System

The host system is a PDP 11/34 minicomputer operating under RSX-11. Four workstations, each equipped with its own robot arm, are connected to the host via an RS-232c link. Each workstation employs a LSI-11 microprocessor, 64K bytes of read/write memory, parallel and serial input/output, and a sixteen channel analog-to-digital board. When a user logs onto a workstation, he is prompted with a FORTH menu similar to the

one shown in Figure 1. Using these options, students can quickly "find their way around" the system and become proficient at building and debugging applications programs. Editing of screens takes place on the 11/34. Screens can then be downline loaded into the workstation.

```
B       — Backup files
C f l   — Copy screens to seq files
D s     — Delete screen
E s     — Edit screen
F       — New screen Files
H       — Host forth
I       — Initialize diskette
L f l   — Load screens from seq files
M s d   — Move screen
O       — enter host system commands
P f l   — Print screens
R       — Restore files
S n     — create screen file
T s     — Type screen
U       — Unlock screen files
W       — Workstation forth
X n     — eXtend screen file
```

```
                TASK4TH Version 1 Release 2

Looking for screens 1 through 100
  in screen file LB3:[1,60]FORTH

Looking for screens 101 through 254
  in screen file SY:[1,10]FORTH

Using TASK4TH located in LB3:[1,60]

Logged onto account [1,10]
```

```
?       — redisplay menu
null    — exit back to host system
```

```
Function [ars1  [ars2]]?
```

Figure 1. TASK4TH Main Menu. Various utility functions are available for manipulating screens and files. 'Host forth' is FORTH running on the PDP-11/34. 'Workstation forth' downloads a standalone version of FORTH to the workstation.

Robot Arm

The Armatron toy robot arm features five degrees of freedom and a gripper that opens and closes. The two joysticks which engaged the gearing controlling motion of the arm have been replaced by a set of six levers. Two opposing solenoids are required for each lever for forward and reverse directions. Hall-effect sensors were added to the waist, shoulder, elbow, wrist, and gripper to provide position information to the computer. An optical encoder supplies information regarding wrist-roll positioning.

Motion of the arm is controlled via the parallel output facility and is as simple as writing out a 12-bit word with 1 bit active (low). The example below shows definitions of words that cause the shoulder to be moved down or up. Due to power considerations, only one joint is moved at a time.

```
HEX
: SHLDER-DN    FFFE PIOPUT ;
: SHLDER-UP    FFFD PIOPUT ;
```

The position of a particular joint can be determined by performing an A/D on the output of that joint's Hall-effect transducer. The Hall-effect device operates by measuring the field due to a magnet which is mounted on the rotating joint. The transducer is positioned as

in Figure 2. Using a supply voltage of 10 volts, the transducer output varies sinusoidally from approximately 8v when positioned over the north pole of the magnet to about 2v when positioned over the south pole. Between 90 degrees and 270 degrees, the output voltage is unique. Therefore, *absolute* position information is available if joint range is restricted to 180 degrees and the magnet is positioned correctly. However, if the joint range exceeds the 180 degrees, some values will be repeated. (See Figure 3.)

The output of each Hall-effect sensor is connected to an input channel of a 0v to 10v, 12-bit A/D converter in the workstation. The word WAIST? returns the digitized output from analog input channel seven.

```
: WAIST?  7 A-TO-D ; ( channel -- data )
```

A-TO-D is a hardware dependent routine that provides a high level view of the A/D facilities.

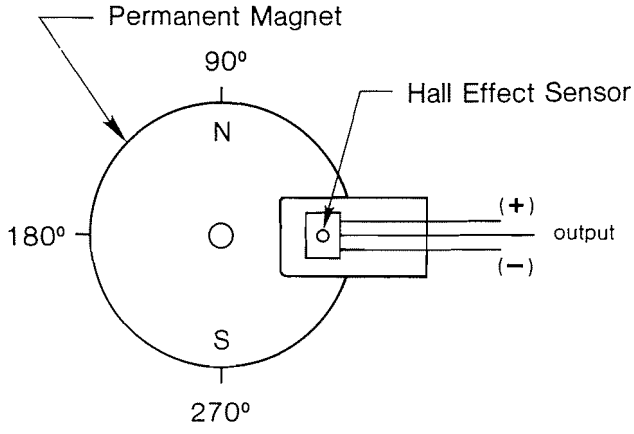

Figure 2. Hall-effect sensor and magnet for a typical joint.

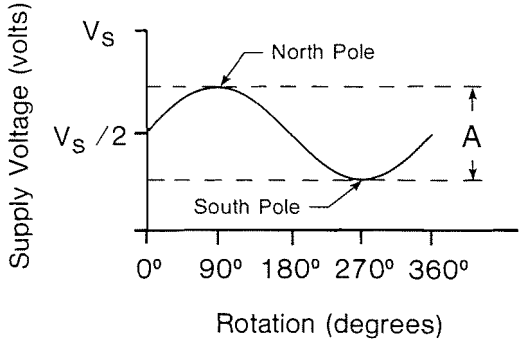

Figure 3. Output of Sensor vs. Rotation of Disk. As the disk (fixed to one link of the joint) rotates, the Hall-effect sensor (Micro Switch, 91SS12-2) produces a sinusoidal output voltage, varying six volts between the north and south poles of the magnet (Permag, SD1701).

Because the wrist roll is not suitable for Hall-effect monitoring, an optical shaft encoding technique is used. A round, paper ring, as shown in Figure 4, is attached to the rotating wrist surface. Two optical sensors, with some additional circuitry, provide a TTL signal that is fed to the A/D and the parallel input port, providing *relative* position information.

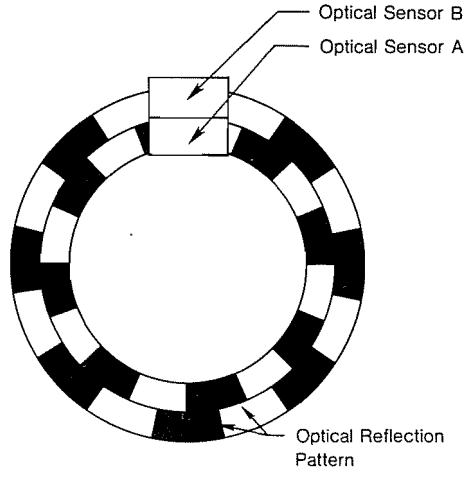

Figure 4. Two phototransistors (TRW Optron, OP125A) mounted over a reflection pattern provide information concerning wrist roll. The offset pattern gives a resolution of 32 state changes (wrist positions) and the direction of rotation.

Photo 1 shows some of the detail of the modifications to the Armatron. The Armatron robot arm is available from Radio Shack for thirty to forty dollars. The solenoids, magnets, Hall-effect sensors and various interface parts are generally available for less than one-hundred dollars total. The most expensive aspect of the modification is the machining and assembly of the levers and base plate, which was performed for us by the Wright State Instrument Shop. A reasonable upper bound to the total modification cost is three-hundred dollars. Because the modification is relatively cheap and straightforward, we have been able to obtain enough arms to have one for each workstation and several backups. Additional information concerning our Armatron modification can be found in a recent *Robotics Age* article [1].

Applications

The Robotics Laboratory

The facilities described above are used primarily in supplemental laboratory instruction for an introductory course in robotics. This course is offered as a four quarter-hour elective for senior level engineering and computer science students. Generally, one or two sections, each with approximately thirty students, are taught during a year. Major topics surveyed in the course include an introduction to FORTH, matrix coordinate transformations, robot arm kinematics, trajectory planning and arm dynamics, feedback control loops for joint motion control, and computer/robot vision research.

The backgrounds of the students who take the course range from that of the engineers, which includes a foundation in both mechanical and electrical systems but minimal program-

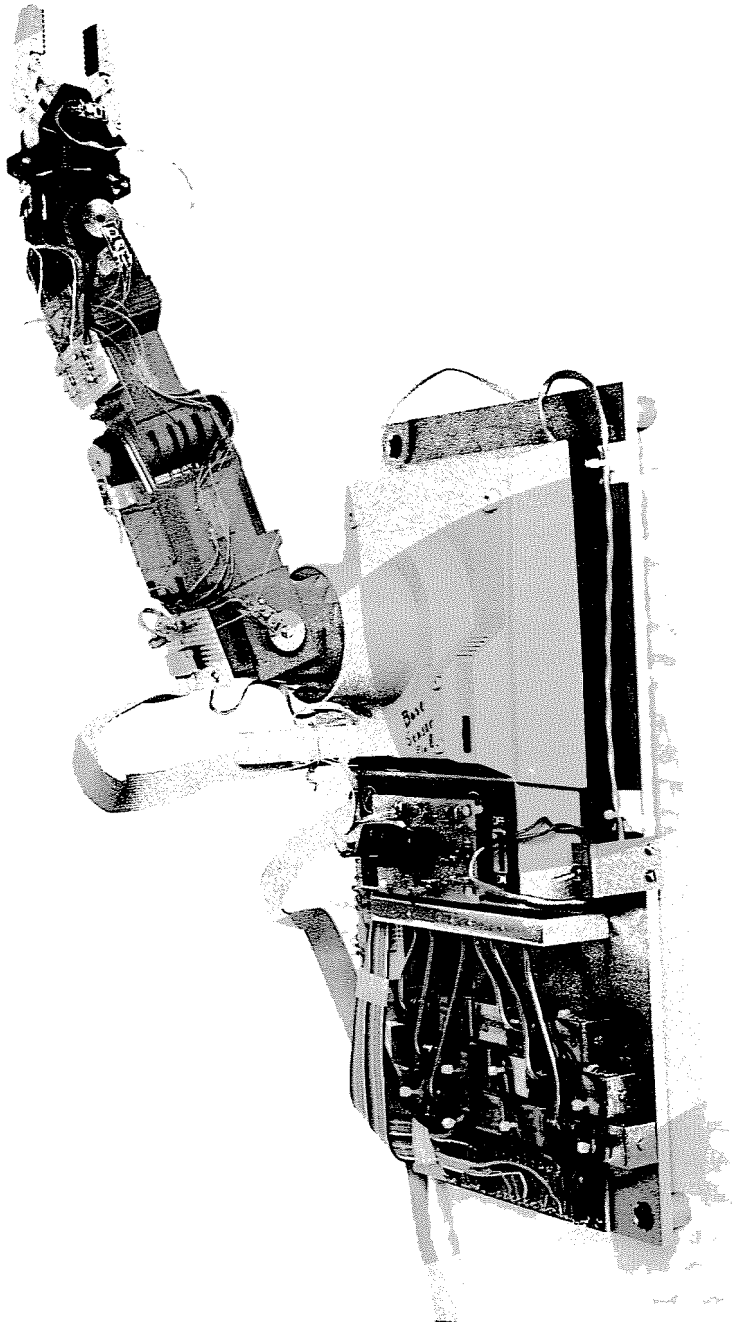

Photo 1. The original Super-Armatron. An improved model is currently used in the Robotics Laboratory, incorporating small PC boards for a heavier power supply and the interface circuits, and a neater arrangement for running sensor leads along the arm. Note the six gearing levers and their solenoids, and also the Hall-effect sensors and magnets mounted on the joint axes.

ming experience, to that of the computer science majors, with extensive software design experience and exposure to digital hardware, but little basic engineering knowledge. These diverse backgrounds have helped to set the survey tone of the course and have necessitated the design of laboratory experiences which do not focus too heavily on either engineering or software design, but instead give students an appreciation of some of the fundamental requirements and difficulties involved in computer control of a robot system.

Generally, students perform four projects during the terms. Each involves developing FORTH programs which control the operation of some machine, and each requires a thorough written report describing the overall approach to the solution and details of the implementation. Projects are done by students working in pairs, and two or three weeks are allotted for each project. In addition to a weekly two-hour lab session, the lab is available to the students on a daily, twenty-four hour basis.

Although given a brief introduction to the important concepts involved in programming in FORTH and to the primary features of the language, the students are expected to learn how to write FORTH programs on their own, with the help of a supplemental text [2]. The first project, which usually involves some basic I/O functions and FORTH control structures, allows the students to become familiar with the programming facilities and with FORTH.

The second project uses a twelve-volt DC motor with an optical shaft encoder attached. This encoder provides a zero pulse sync line and two output lines (four possible states) which make 4096 state transitions per revolution. Students write programs to control the motor and monitor the shaft encoder through the parallel I/O port.

The third project is the most extensive and demanding of the four. In this project students develop a "teach-mode" control program which maintains a database of trajectories which are taught to the modified Armatron through the terminal keyboard, and controls execution of these trajectories from a menu of commands. This project is discussed in detail below.

The final project involves conversion of the values obtained from the Hall-effect transducers into corresponding angular values, allowing positioning of the arm to be accomplished in terms of angles instead of Hall-effect magnitudes.

The Teach-Mode Program

As an illustration of the level of FORTH programming required and of the kinds of problems involved in the lab projects, the following summary of the features of one student solution to the teach-mode project is presented.

The requirements for the project are as follows:

— It must be able to determine the location of all joints of the arm.

— It must be able to control the motion of all joints to orient the arm in any desired position.

— The user should be able to control the arm from the keyboard, and when in a "learn mode" be able to "teach" the arm a sequence of positions in a trajectory.

— Finally, up to ten separate, one-hundred point trajectories should be maintained and the program should be able to execute any trajectory on request.

Figure 5 is a block diagram representing the interaction of the different program modules.

At the lowest level, several I/O primitives were defined to provide convenient control of robot motion and easy input of sensor data and user commands. Using predefined I/O drivers, several words were defined which initiate and stop motion of the different joints. Position data for the waist, shoulder, elbow, and wrist-pitch joints is obtained using the A-TO-D function described above. Wrist-roll data from the optical shaft encoder is brought in through the PIO. Several standard serial I/O functions receive commands and data from the keyboard. A full set of user messages was also defined.

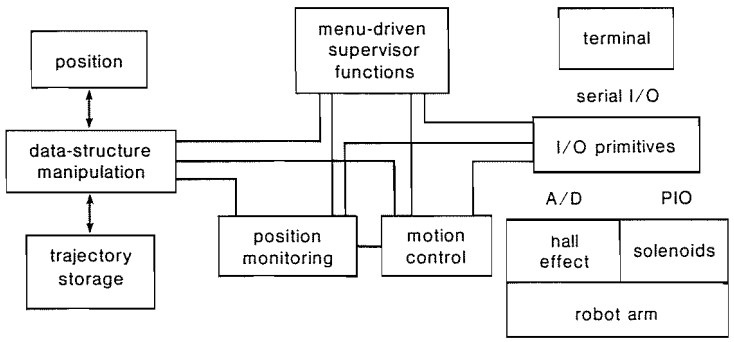

Figure 5. Organization of the teach-mode program. Data-structure primitives provide access to the current position array and the trajectory database. I/O primitives provide access to the terminal and give a higher-level view of robot arm control. The position-monitoring and motion-control subsystems perform most of the work involved in moving from point to point and sensing the current position of the arm. Supervisor functions provide overall control of operations such as manual control of the arm and the learning and executing of trajectories.

The fundamental data structure used by this program is a seven word record containing the position data for all five axes of motion, the gripper, and a time delay factor which determines the time interval between moves. Seven variables corresponding to these values form a "current position group" which is continuously updated with the position of the arm. An arbitrarily large array area holds the blocks forming the learned trajectory database. Two variables pointing to the current one-hundred point trajectory area and the current seven word position block are manipulated by an indexing function to provide an address for data storage and retrieval. A STORE function copies the position values and time delay factor from the current position group to the array when trajectory points are being learned.

The core of this program contains the routines which are responsible for monitoring the position of the robot and controlling its motion. The overall approach used is straightforward. While each joint is being positioned for a new trajectory point, a tracking routine updates that joint's current position variable. When all joints have been moved to the new position, the current position values are transferred to the proper block of the trajectory array area. Execution of a trajectory is accomplished by establishing pointers to the first desired position in the trajectory and then stepping through each position until a trailing null marker is reached. At each position the desired joint position value is examined and compared with the current position. When necessary, the joint is moved and tracked until the values match.

As was previously described, the use of the sinusoidal Hall-effect transducers to establish joint position creates a two-value problem on axes whose range of motion brings a pole past the sensor. In this case, one magnitude represents two unique positions on either side of the pole. This problem is easily solved by adding a sign to the sensed magnitude which corresponds to the slope of the sinusoid at that point. A slope convention was established: increasing magnitudes obtained when moving clockwise/up indicate a position slope; decreasing values indicate a negative slope. The convention is reversed for motion in

the other direction. When motion of a joint is initiated, the direction is stored in a temporary location. Position tracking of joints is accomplished by several functions which compare the newest sampled values with those in the current position group, assign the slope with respect to the direction of motion, and replace the current position group values. Figure 6 gives the general tracking algorithm, and Figure 7 shows a sample of student code implementing the tracking algorithm.

The orientation of the wrist-roll axis is established by counting state transitions of the optical shaft encoder. When initial positions are stored, a value of 16 is assigned to the roll axis. When this joint is moved, transitions in the clockwise direction (0-1-3-2-0) cause the roll value to be increased; counter-clockwise transitions cause the roll value to be decreased. The

```
store direction
repeat
   get old position
   get new position
   determine sign
   store new position
   if wrist_roll transition
     update roll
until motion_stop command (from keyboard)

determine_sign
   if new position = abs (old position) then
     if (new position - abs (old position)   0 then
       new position = new position * (-1)
     if direction = counter-clockwise then
       new position = new position * (-1)
```

Figure 6. Tracking algorithm.

```
: get_w ( -- n) 1 7 a-to-d ;
: store_dir (n --) dir ! ;
: new_pos ( o n -- n') over abs over swap - dup abs 1
                       if 0 < if minus endif dir @
                       if minus endif swap else drop endif drop ;
: wrist_pos ( -- ) roll @ get_r cnvrt dup roll ! swap - dup if dup
                   0 > over 3 = not and over -3 = or if 1 else -1
                   endif wrist +! endif drop ;
: end_mtn ( -- ) off get_key? drop ;
: get_pos (o n -- b) new_pos swap ! wrist_pos is_key? ;
: track_w ( n -- ) store_dir begin waist dup @ get_w get_pos
                    until end_mtn ;
```

Figure 7. A sample of student code implementing the tracking algorithm for the waist joint.

gripper position (open/closed) is sensed by the relative field strength of a mounted magnet whose distance from the sensor is variable. This position is unique and is simply recorded in the current position group. As trajectory points are being learned, the interval time delay between points may be entered. This is also recorded in the current position group.

Two difficulties are encountered using these position monitoring methods. The first is that the resolution given by Hall-effect sensors near the poles of the sinusoid is fairly poor. There is some true position error associated with operation in these areas of motion. The second problem is the nature of the equipment used. Because the arm is a modified toy, there is significant slop and jitter in the joint linkages. Quick starts and stops tend to foul up the slope assignment algorithm. Also, when certain joints are moved, other joints tend to drift. This is a problem for the roll axis, since its position is lost if any state transitions are missed. This necessitates constant monitoring of the roll axis.

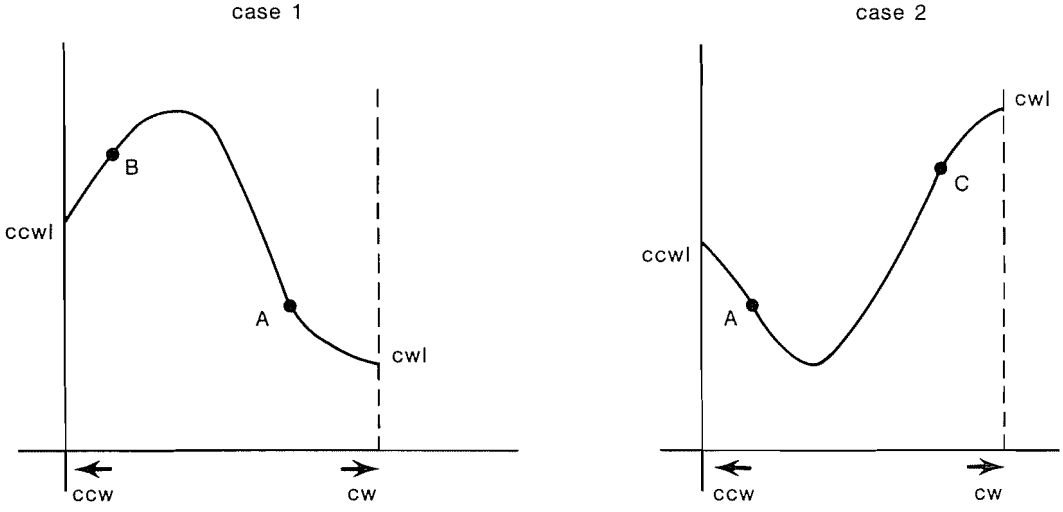

Figure 8. The joint is currently at A. Destinations B and C have the same sign and magnitude. The motion decision algorithm must know something concerning the portion of the sinusoid over which the joint is operating to choose the proper direction — counterclockwise in case 1 and clockwise in case 2.

The motion control algorithm must perform two basic tasks. Given the destination and current position, the algorithm must determine the proper direction of motion. Once in motion, the joint must be monitored until the desired position is reached. Due to the two-value problem, implementaion of an algorithm for choosing a direction of motion proves to be somewhat difficult. Different joints operate over different sections of the sinusoid; in addition, there is significant difference between arms. There is no general way of knowing, for example, whether a desired point can be reached by going further clockwise or all the way back counter-clockwise. (See Figure 8.)

A general decision function was implemented, taking as inputs the current position, desired position, and two limit values representing the clockwise/up and counterclockwise/down limits of motion. These values are initialized by the user at the start of a "learning" session by moving the arm through its ranges of motion and indicate to the decision function

the portion of the sine wave on which the joint is operating. Figure 9 gives the general motion decision algorithm, and Figure 10 shows a sample of student code implementation.

As mentioned earlier, sloppiness in the linkages and feedthrough in the joint gearing require that the wrist-pitch and roll axes be monitored for changes in position when other joints are moved. Because position values for wrist roll and grippers are unique, these motion control functions are standard looping and comparison structures. A general MOVE definition calls each of the six joint control functions in turn, working outward from the waist. After all motion for a trajectory point is completed, the time delay factor establishes a delay of several seconds until the next move.

At the highest level, six functions (available to the user through a main menu) oversee operation of the program. The first function performed at a session, LIMITS, initializes the limit value variables discussed previously. A second definition, POS, displays the current position values at the terminal. Stored position values may be accessed and displayed by a DISPLAY function. The major robot control function is KEYPAD. This routine displays a graphic menu of keypad functions and poll the keyboard for control inputs. These are executed, and tracking of the arm is performed. From this mode, the current position can be displayed; also, the current position values can be stored into the trajectory array.

The LEARN function initializes the trajectory and position values of those requested by the user. Control passes to KEYPAD until the trajectory is finished. LEARN then deposits a trailing null marker.

```
{ c_pos  → current position }
{ d_pos  → desired position }
{ cwl    → clockwise limit  }
{          etc              }

if c_pos < d_pos then
  if (c_pos < cwl and c_pos < ccwl) and
          (d_pos > cwl and d_pos > ccwl) then
    move ccw
  else
    move cw
else
  if (c_pos > cwl and c_pos > ccwl) and
          (d_pos < cwl and d_pos < ccwl) then
    move cw
  else
    move ccw
```

Figure 9. Direction decision algorithm.

```
: move (c d cwl ccwl -- cwl ccwl dir) over over >r >r >r >r
       over over < if dup i > swap r> i swap >r > and swap dup
       r> < swap r> < and and else dup i < swap r> i swap >r
       < and swap dup r> > swap r>> and and not endif r> r> rot ;
```

Figure 10. A sample of student code implementing the decision algorithm.

Finally, TRAJ oversees the execution of motion. Three options are available: move to a unique point; execute a complete trajectory; and execute a trajectory starting at a point other than the beginning. These trajectory functions step through the position blocks, executing the general motion control routines until a null value is reached.

Future Projects

Currently, the teach-mode program is the major lab project of the course, requiring three to four weeks of student effort. Typically, program development requires thirty to forty man-hours, resulting in fifteen or twenty screens of code. Clearly, students receive exposure to problems encountered in computer control of real-time processes. While the use of toy arms does limit the realism and potential of these projects, there is enough flexiblity to develop other, similar projects, possibly at a more sophisticated level. Currently, packaged functions are being developed which convert the Hall-effect sinusoids into angular values. This will provide a basis for projects involving direct kinematics solutions using matrices and inverse kinematics/trajectory planning problems. In addition, our implementation of FORTH allows the use of interrupts and multi-tasking in more sophisticated control programs.

Evaluation and Conclusion

Environment

Generally, we find that the system environment we are using works very well. The PDP 11/34 supports a reasonable development load with good response. The ability to download FORTH to the LSI-11 workstations is particularly convenient, freeing system resources and protecting the system from the inevitable user errors. The workstations have ample I/O capability for most lab work. On the negative side, the system at present lacks the ability to handle rapid data transfer from the workstations back to the host. Thus, data generated at the workstation (such as learned trajectories) is lost at the end of a session.

The modified toy arms have also performed well, providing sufficient realism for a minimal cost. Sufficient numbers are available to operate the lab at full capacity even when some arms are in need of repair. While they suffer from obvious limitations, these arms free resources for more advanced projects and research.

FORTH

In the two quarters that we have used FORTH, it has demonstrated its usefulness as an instructional language for teaching introductory robotics. Because of the requirement for speed and the intensive machine-world interfacing required, FORTH has proved superior to the use of most typical high-level languages. In particular, we have experienced significant improvement over the Pascal implementation used previously. At the same time, FORTH is much more convenient than assembly language for instructional program development. Generally, engineering students with a minimal programming background and computer-engineering students with a more extensive background are both able to gain a command of FORTH sufficient to solve the lab projects. In particular, the functional approach of FORTH seems more easily learned in a short time than the more demanding syntax of a standard high-level language.

However, the advantages of using FORTH do not come without disadvantages. While fine for application in student projects at this level, the need to use complicated data structures or algorithms in more sophisticated projects might overwhelm inexperienced FORTH programmers. The well-known problems of packaging and readability limit the usefulness of FORTH in large projects or group efforts. As more courses at the university begin to use FORTH, and as general packaged functions become available to the students, many of these potential problem areas may be reduced.

References

[1] Schiavone, J. J., M. Dawson, and J. E. Brandeberry, "Super Armatron," *Robotics Age*, Robotics Age Inc., Vol. 6, No. 1, pp. 20–28 (January 1984).

[2] Brodie, Leo, *Starting FORTH*, Prentice-Hall, Inc., New Jersey (1981).

Manuscript received June 1984.

Alan J. Cotterman received his B.S. in Computer Engineering from Wright State University in 1984, and is currently pursuing an M.S. in Computer Engineering. His interests include digital hardware design, software tools, robotics, and computer vision.

Daniel M. Willeford received his B.S. in Computer Engineering from Wright State University in 1982 and is also pursuing an M.S. in Computer Engineering. Dan is currently employed by Digital Technologies, Inc. His interests include CPU architectures, microprogrammed machines, binary vision, and robotics.

Dr. James E. Brandeberry received his BSEE and MSEE degrees from the University of Toledo in 1961 and 1963 respectively, and his Ph.D. from Marquette University in 1969. He is currently a Professor of Computer Science and Engineering at Wright State University. His interests are in electronics, computer hardware design, and robotics.