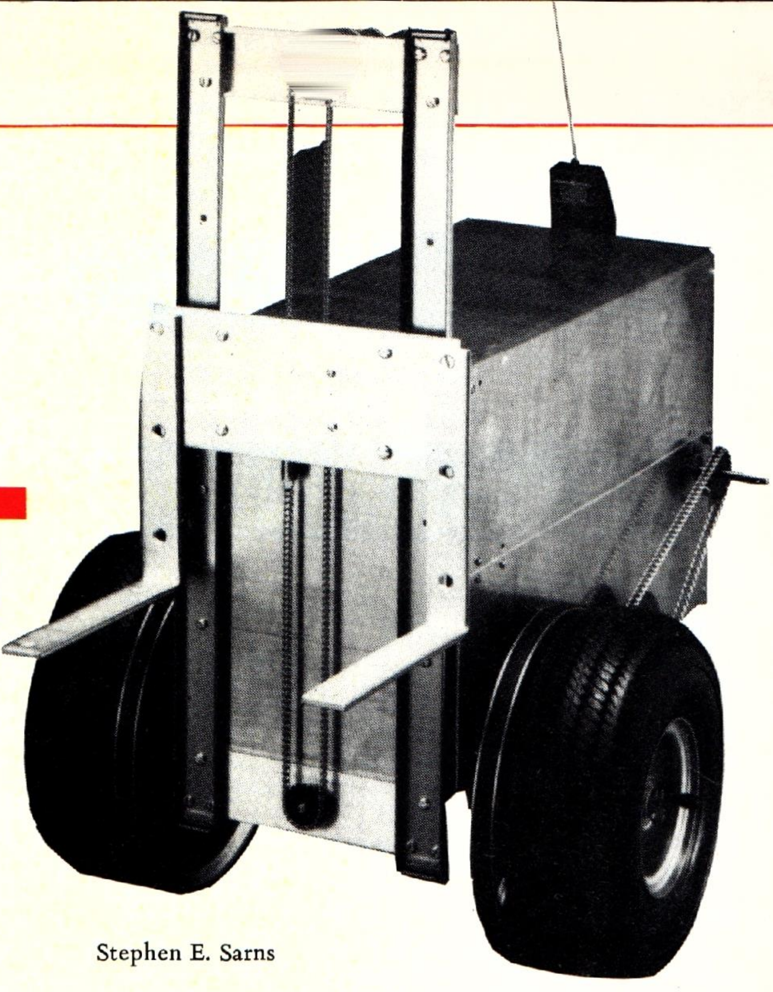
R-E ROBOT

Here's an application that can give a robot builder great "joy."



Stephen E. Sarns

art 10 Now that we've finished building the *R*obotic *P*ersonal *C*omputer (RPC), the mechanical drive system, and the interface and controller board, we have a robot that is ready to run. But what are we going to do with it? In this article we are going to show you a simple application program that will demonstrate how to:

- Develop programs on the RPC.
- Read the A/D converter.
- Read the switch status.
- Control motor direction.
- Control motor speed.

In addition, our first application, a joystick controller for the robot, illustrates the use of the underlying Forth-83 language. You will want to become familiar with Forth-83, even though you have the power of the *R*obotic *C*ommand *L*anguage (RCL) at your disposal, because Forth-83 allows you to extend the RCL to suit your needs. Remember, the RCL is simply an application written in Forth-83. Your application can use the resources of RCL or Forth-83 or both. That is shown graphically in Fig. 1.

The application

A joystick is attached to the discrete user bus (PL3 on the control board). Analog channel 0 is used to monitor the joystick's forward/backward position. Analog channel 1 monitors its left/right position. For safety, a pushbutton *deadman* switch is connected to the D7 (data bit 7) input. That switch must be depressed for the robot to operate; if it is released, all power to the motors is cut. The schematic for the joystick hookup is shown in Fig. 2.

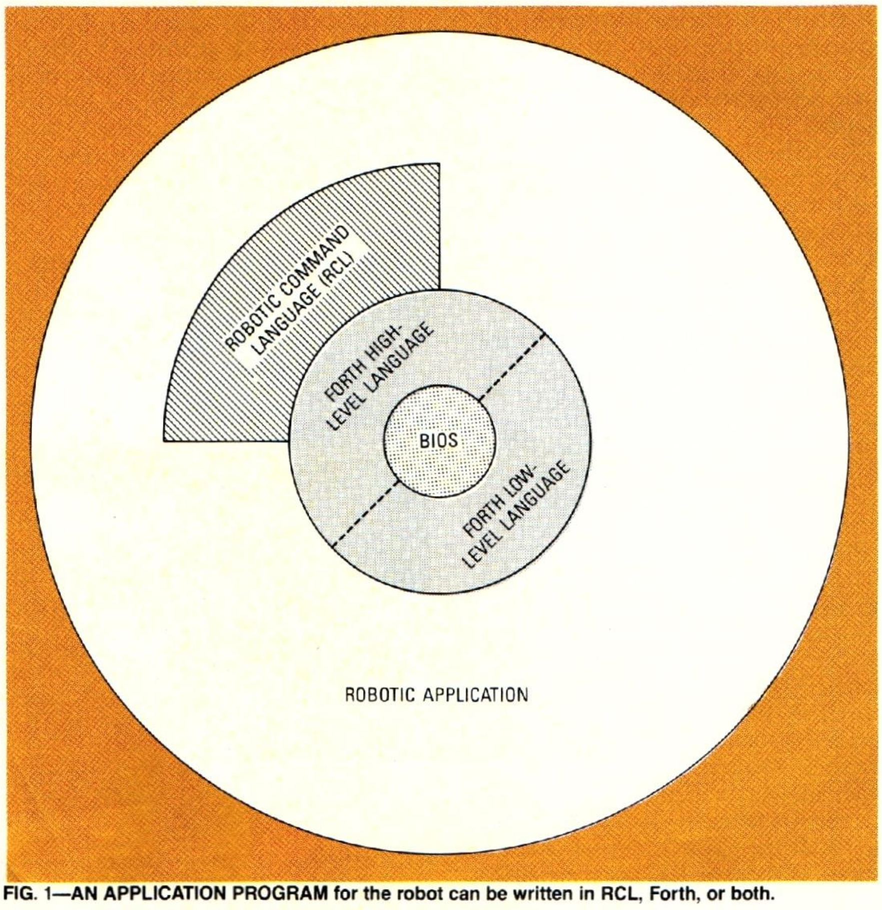


FIG. 1—AN APPLICATION PROGRAM for the robot can be written in RCL, Forth, or both.

All of the code we are going to write will fit into six Forth screens. Therefore, you do not need a disk drive. You can use the RAM word to store all of your source code in virtual memory above 32K. Alternatively, you can attach a disk drive and keep all of your code on disk. Last, you could type in all of the colon definitions from the keyboard, but doing so is handy for testing, not developing an application.

If you are going to use RAM as the development media, power down and install a RAM IC in the IC5 socket on the RPC, execute the RAM word, and remember that you now have virtual memory screens 32 to 39. If you have a disk drive attached, simply insert a disk formatted under MS-DOS version 2.x or the BIOS formatter. Decide where you want to start (never screen 0) among the 360 available screens on the disk. For simplicity and clarity, we'll assume that you'll start your first screen at 32 just like the RAM-based example. Incidentally, the example was developed on a disk, but there is no functional difference after executing the RAM or DISK word.

Install a blank 2764 EPROM at IC31, the EPROM programming socket on the RPC; once again, power should be off when doing that. It's handy to keep an EPROM installed there, especially if you are developing code in RAM and do not want to risk losing it during a power down.

Invoke the editor by typing 32 EDIT. The 32nd screen is read in from disk (or RAM), the terminal's display is cleared, and then the screen of text is listed. Alternatively, you can type EDITOR 32 LIST. Your first command will probably be WIPE; that clears the entire screen. Now begin entering the source code with the editor. When you want to test a word, make sure you FLUSH your changes to disk before you LOAD the screen; otherwise all of your changes will be lost if your system crashes.

LISTING 1

```
Scr # 32
/ joystick documentation

/ analog channel 0 is used by forward/reverse joystick
/ analog channel 1 is used by left/right joystick
/ digital input D7 is a normally open "deadman" switch
/ variables LSPEED and RSPEED contain the joystick position
/    in the range of -127 to +128








-->
Scr #33
/ joystick - i/o
/ all low level drivers for reading and controlling here

hex
: atod ( channel number --- byte )
    dup 150 pc! 150 pc! 100 0 do loop 150 pc@ ;
: switch? ( ---f ) 120 pc@ 80 and 0= ;
: lmtr-period 36 103 pc! 100 /mod swap 100 pc! 100 pc! ;
: rmtr-period 36 113 pc! 100 /mod swap 110 pc! 110 pc! ;
: stop-left      0 124 pc! ; : stop-right      0 125 pc! ;
: enable-left   1 124 pc! ; : enable-right    1 125 pc! ;
: lfwd 0 121 pc! 1 120 pc! ; : rfwd 0 122 pc!  1 123 pc! ;
: lrev 0 120 pc! 1 121 pc! ; : rrev 0 123 pc!  1 122 pc! ;


-->
Scr #34
/ joystick - joystick interrogation
/ INTERROGATE reads both channels of a/d. Results of fore/back
/ are stored in LSPEED and RSPEED. The left/right position is read
/ and the contents of LSPEED and RSPEED are modified according
/ to DIFF-CONV
hex
300 constant maxspeed     variable lspeed      variable rspeed

: speed-conv ABS 5 * maxspeed swap - ;
: diff-conv 5 / ;

: interrogate
    0 atod 80 - dup lspeed ! respeed !
    1 atod 80 - diff-conv dup negate swap
    lspeed +! rspeed +! ;
-->
```

The program

One of the most common criticisms of Forth is that it is unreadable and therefore unmaintainable. We think you will find Forth quite the opposite—if you start at the end and work backwards. Remember, in Forth, you can design your program from the top down, meaning that you can figure out how it should work, write that part of the code, and later add the support words to make it work that way. By starting your examination of the application from the end, you will understand how the program works, leaving the details for later.

Therefore, for now we'll ignore the low-level words, which can be rather esoteric, and begin with screen 37, which contains a single word called JOY. That is the joystick program. While it is executing, pushing the stick forward causes the robot to go forward; pulling the stick back causes it to reverse. Turns are made by moving the joystick in the desired direc-

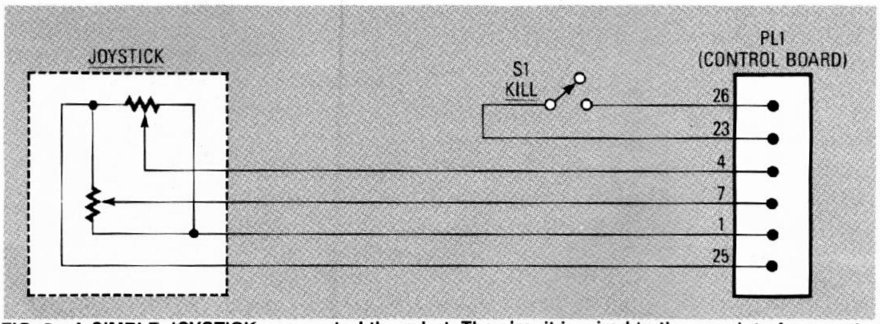


FIG. 2—A SIMPLE JOYSTICK can control the robot. The circuit is wired to the user interface on the control board. For safety, a "deadman" or kill switch must be provided.

LISTING 1 (continued)

```
Scr #35
   Ø / joystick - left motor control
   1 decimal
   2 : leftmotor
   3         lspeed @ 1Ø > switch? and
   4          if lfwd lspeed @ speed-conv lmtr-period enable-left then
   5         lspeed @ -1Ø < switch? and
   6          if lrev lspeed @ speed-conv lmtr-period enable-left
   7          then
   8         lspeed @ -1Ø 1Ø between switch? Ø= or
   9          if stop-left then ;
  1Ø
  11 : testleft
  12       begin interrogate leftmotor lspeed @ . key ? until ;
  13 -->
  14
  15
Scr #36
   Ø / joystick - right motor control
   1
   2 : rightmotor
   3         rspeed @ 1Ø > switch? and
   4           if rfwd rspeed @
   5             speed-conv rmtr-period enable-right
   6           then
   7         rspeed @ -1Ø < switch? and
   8           if rrev rspeed @
   9             speed-conv rmtr-period enable-right
  1Ø           then
  11         rspeed @ -1Ø 1Ø between switch? Ø= or
  12           if stop-right then ;
  13
  14
  15 -->
Scr #37
   Ø / joy
   1 hex
   2
   3 : joy
   4         begin
   5           interrogate rightmotor leftmotor
   6           lspeed @ 1Ø .r rspeed @ 1Ø .r
   7           cr key?
   8           if key Ød =
   9           else Ø
  1Ø        then
  11       until ;
  12
  13 decimal
  14
  15
```

tion. Speed is controlled by the displacement of the joystick.

Note the BEGIN/UNTIL structure in screen 37. The code executed within the loop is indented for easier understanding. The loop repeats or ends depending on the value of an argument left by the IF/ELSE/THEN construct. The loop checks to see if a key has been pressed. If so, the KEY statement retrieves the character and it is compared to a ØDH (carriage return). If the result of the comparison is true, a *true* flag (−1) is left on the stack. If the KEY statement is false, ELSE is executed, which leaves a *false* flag on the stack (Ø). UNTIL causes the loop to continue until a *true* flag is found on the top of the stack. Thus, JOY will execute until the carriage-return key is pressed.

Why not terminate on any key? Sometimes after you disconnect your terminal, the floating RS-232 input can be subject to noise. Generally, after your code is in ROM and you are running without the terminal, install a shorting bar between RxD and ground on the RS-232 connector.

INTERROGATE (in line 5 of screen 37) is a word that updates the two variables LSPEED (left motor speed) and RSPEED (right motor speed). The numbers in LSPEED and RSPEED range from −128 to +127. Positive means forward, negative means reverse.

The next two words, RIGHTMOTOR and LEFTMOTOR, examine the contents of the words of LSPEED and RSPEED respectively and set the appropriate motor's speed accordingly.

Finally, the contents of variables LSPEED and RSPEED are fetched and displayed on the screen followed by a carriage return. The .R word is used to right justify the displayed values.

Testing

Each word should be tested after it is written and compiled into the dictionary (LOADED from disk). A common test/debug method is to load up the stack with a few easily identified values, such as 1, 2, and 3. Then type in each word of the definition being tested (not in a colon definition, but interactively). After each word, execute .S, which prints out the contents of the stack without modifying it. Do so to make sure that nothing has been removed from or left on the stack.

It can be useful to define small test. For example, having defined the word ATOD, which takes a channel number off the stack and returns the value of the corresponding A/D converter, you can make sure that it works with all 8 channels by using the following word:

```
: TEST-ATOD
BEGIN 8 0
DO I ATOD . LOOP CR KEY?
UNTIL ;
```

That word prints out the results of all 8 A/D conversions. Your joystick should show up at the first two positions. Terminate the test by pressing any key.

SOFTWARE SOURCES

Micro K Systems (15874 East Hamilton Place, Aurora, CO 80013, 303-693-3413) will provide the following: Commented source code in RE-robot disk format, $2.00. Printed source-code listing, $15.00. Two 27128 EPROM's with source screens (and without comments) for the R-E Robot, $39.00. With EPROM's you won't need a disk drive, but you should also obtain the printed listing to read the comments. The Laxen and Perry F83 Model disk with full source code and metacompiler for customizing F83, in MS-DOS 360K format, for a PC compatible computer, $25.00. (Very useful for learning FORTH if you already have a PC.) All orders must be prepaid. NO COD's. Include $3.00 for shipping with each order. Additional source code and applications will be available from Micro K Systems. Contact them for more information.

Speed and direction

The word RIGHTMOTOR, shown in screen 36, examines three possibilities:

- Is the value of RSPEED greater than +10 and is the switch pressed?
- Are the contents of RSPEED less than −10 and is the switch pressed?

continued on page 63

continued from page 60

- Are the contents of RSPEED between −10 and +10 or is the switch released?

You will see the three IF/THEN statements that test those cases.

Assuming motion is going to take place, RFWD (right-motor forward) or RREV (right-motor reverse) is executed. Those words enable the appropriate relay and disable the other so that motion takes place in the desired direction. It is important to notice that the relays are set before the motor is enabled. That prevents inrush currents from damaging the relay.

Next, the value of RSPEED is fetched and used as an argument to SPEED-CONV. SPEED-CONV takes an argument (from the stack) that is proportional to the speed and returns an argument (on the stack) that is the period of the motor control-frequency generator. Speed may vary from 0 to 128. When reverse motion has been requested, the speed argument is negative. We could NEGATE it before passing it to SPEED-CONV; however, it is more convenient to have SPEED-CONV use the ABS operator to make sure all arguments are positive.

As shown in lines 5 and 9 of screen 36, the output of SPEED-CONV is passed directly to RMTR-PERIOD (right-motor period), which sets the frequency of the 8253 control element in the right motor's phase-locked loop circuit on the control board.

Last, ENABLE-RIGHT (again shown in lines 5 and 9 of screen 36) electronically enables the high-power output stage of the motor-control circuit. If no motion is going to take place, STOP-RIGHT is executed. The code is identical for the left motor; it is shown in screen 35.

Moving on to screen 34, INTERROGATE reads channel Ø of the analog-to-digital converter, converts it to the −128 to +127 range and stores the result in both LSPEED and RSPEED. Next channel 1 (sideways displacement) is read. That result is converted to the ±128 range and passed as an argument to DIFF-CONVERT on the stack. The result is duplicated (DUP), one copy negated (NEGATE) and used to alter the contents of LSPEED and RSPEED.

Low-level control

Now we start getting into the low-level words shown in screen 33. Those words could be rewritten many different ways and the overall program would still function the same, but the response would change. For example, SPEED-CONV could be rewritten to provide a greater range of motion, limit the top speed, or provide a constant speed.

continued on page 86