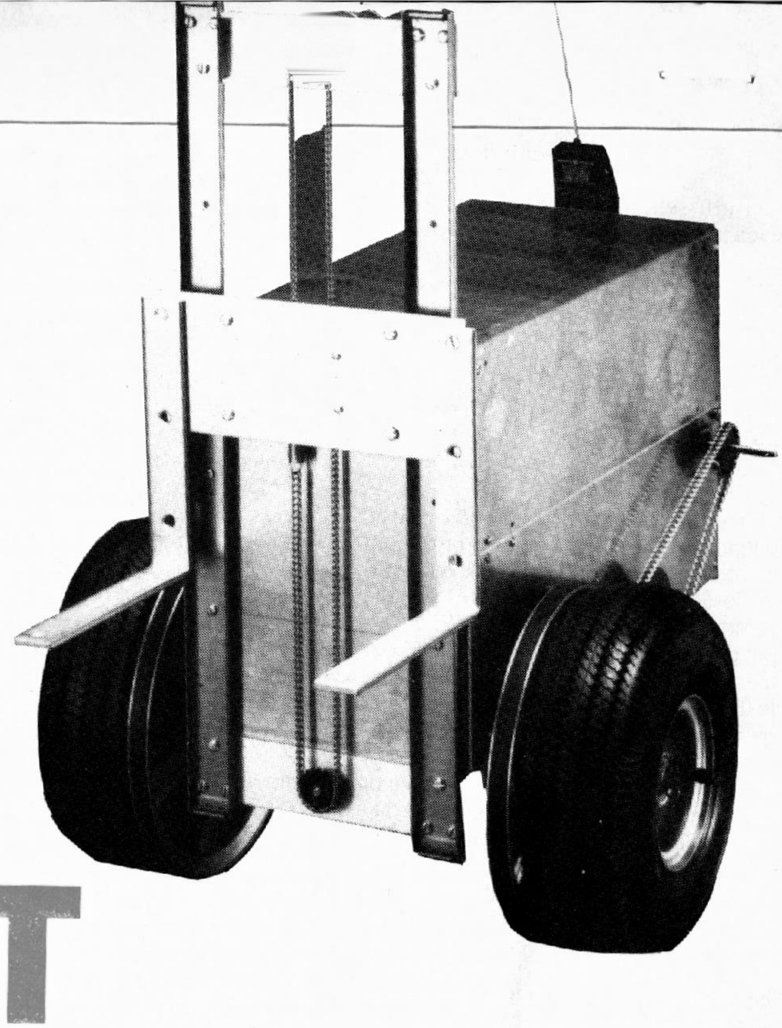

R-E ROBOT

The robot's command language

Part 9 NOW THAT WE'VE ASsembled the robot's hardware, it's time to dig into the software. In this article we'll describe the R-E Robot's command language, RCL (*R*obotic *C*ontrol *L*anguage). It's an easy-to-learn and easy-to-use language written in FORTH. Don't be scared by FORTH; you can use RCL without being an expert at programming in the language. And, as you learn RCL, you'll learn (painlessly) the basics of how FORTH works, so that, if you want to, you can go on and learn the language itself. To give you a chance to see RCL in action, we'll present a robot-based mail-delivery system. You can study our program to learn how RCL works, and you can also use it as the basis of your own program.

How difficult is RCL? Not very. For example, suppose you wanted the robot to move in the forward direction 3.4 feet at a speed of 2 miles per hour. You would simply type in the following code:

```
RERB 2 MPH 3.4 FEET FORWARD
```

RCL includes commands to move the robot forward and backward, to turn left and right, to move its manipulator up and down, and to open and close a gripper. You can combine a sequence of commands and store them for execution at a later time. In addition, commands can also be executed immediately from the keyboard.

Real-time control

The R-E Robot consists of a computer-controlled set of electromechanical devices. The assembly is broadly known as a *motion-control system*.

Real-time motion control requires real-time sensing and processing. One way to ensure proper sensing and processing is to force the computer to execute a control loop at regular intervals. That control loop will be the computer's highest priority. Everything else the computer does will be secondary, and it will have to do those other things as it finds time.

A simple way to implement the control loop is to have a clock IC generate an interrupt at regular intervals. Each time the clock interrupts the microprocessor, it will execute the control loop, and then it will return to whatever it was doing before the interrupt occurred. The amount of time the computer spends executing the control loop must be less than the time interval between interrupts.

RCL basics

The software that controls the robot is built up layer by layer. The most primitive words must be defined first; more-complex words are defined using the previously defined words; at the top level are the RCL words that make motion control easy. As each word is defined it can be tested and debugged. When it is debugged, the next layer may be defined.

Notice that you are defining words, rather than writing a program, as with most computer languages. That's not just a matter of semantics; it's also a way of looking at a programming problem. The problem can be broken down into a series of smaller problems, and then those problems can be broken down further, and so on, until you have a set of problems that can be programmed. Each little problem becomes a FORTH word, which in turn becomes part of another FORTH word, so that eventually all we have to do is say something like

```
TURN-LEFT
```

The real-time control portion of RCL consists of the hardware interface, interrupt control, following-error monitoring, velocity control, and position control.

Low-level words

The most-primitive words deal with the robot's hardware: turning the motors on and off, setting the direction in which each motor rotates, and enabling the speed-control circuits. To control the hardware, values must be written to and read from various registers on the robot's control board. Those registers are read and written using the microprocessor's I/O statements (IN and OUT). In RCL, to write an eight-bit value to an output port, the word PC! is used:

```
PC! (value port ----)
```

That statement specifies that *value* is to be output to I/O port *port*. A word about notation is in order. The stack diagram, enclosed in parentheses, represents the parameters required by the word PC! Input parameters appear to the to the left of the dashes, and output parameters appear to the right. In this case there are no output parameters.

FORTH words in general (and those of RCL in particular) make extensive use of the stack, both for parameters supplied to a word, and those that it may produce. The top of the stack is always the parameter furthest to the right. In the preceding example, the stack diagram shows that the value to be written must be pushed on the stack followed by the port to which it is to be written. The word PC! removes these parameters from the stack, uses them, and leaves nothing on the stack. Other words may leave one or more values on the stack.

Motor-control words

Several words operate the speed-control circuits and the relays. For example, ENABLE and DISABLE write an appropriate value to turn on or off a particular function of the hardware. STOP-LEFT, STOP-RIGHT, and STOP use DISABLE to turn the relays off. FORWARD, REVERSE, CW, and CCW enable the proper relays to allow the motors to turn in the desired direction. CW and CCW allow turns to be made by enabling the wheels to turn opposite to each other. GO and COAST enable and disable the speed-control circuits and the motor-drive current as well.

Speed control

Hardware on the control board is responsible for controlling speed (accelerating and decelerating). The hardware makes the software system much simpler than it would be if the software were required to maintain speed alone. The phase-locked loop on the control board maintains the desired motor speed under varying loads. The software only has to set the speed, and to accelerate and decelerate the base unit.

The speed at which each motor runs is determined by the frequency of a signal that is generated by counter 0 of the 8253 timers. Setting the number of counts in the counter determines the period of a squarewave output. The phase-locked loop circuitry responds to the frequency corresponding to that period.

The frequency of the signal applied to the 8253's on the motor-control board is the 2-MHz system clock divided by 16, or 125 kHz. Therefore a count is generated every 8 microseconds (1/125,000). The 8253 is programmed to generate a squarewave whose period corresponds to the value loaded into the counter. So, if the counter is loaded with the value 125, the total period would be 125 × 8 microseconds, or 1 millisecond, which corresponds to a frequency of 1000 Hz.

With a 500-count-per-revolution encoder, the motor speed would be 1000/500 = 2 revolutions per second, or 120 rpm.

The counter can be loaded with any value between 1 and 65,536 (0 actually), corresponding to frequencies ranging from 125 kHz to just under 2 Hz.

Interrupt control

Motor speed must be updated many times per second to produce smooth acceleration and deceleration. The update rate is set by the interrupt-control routines to 100 times per second (i. e., there are 10 ms between interrupts). The 80188 microprocessor has three built-in timers that can generate interrupts. Timer 0 is used by the BIOS and the DOS to maintain a time-of-day clock. The BIOS is set up to generate interrupt 01Ch every time timer 0 counts down to 0. If we change the count value in timer 0 we can use it to generate the motor-control interrupt. However, the time-of-day clock will count in 10-millisecond periods instead of the usual 55-millisecond periods, so a set of time-of-day words will have to be defined for the new rate. In addition, we'll have to install a new BIOS-level interrupt handler to maintain compatibility with MS-DOS.

First of all, we must define the interrupt routine we want to execute. Then we can install that routine so that it is executed each time the interrupt is generated by the timer.

The word INT-OFF disables interrupt generation by the timer so that we can change the interrupt vector, or disable it. INT-ON turns timer-interrupt generation back on. SET-TIMER takes a count that sets the period for the timer. The input frequency to the timer is 2 MHz/3, yielding a period of 1.5 microseconds per count. If the count is set to 6667, the timer will count down to 0 every 10 milliseconds and generate an interrupt.

GET—CS is a special word that is used to return the code segment in which the FORTH system is executing. SET-INT sets the interrupt vector to the word we want to execute each time the interrupt is generated.

INSTALL performs all the tasks necessary to link a new interrupt handler into the microprocessor's interrupt vector in low RAM. After executing INSTALL the interrupt-control word will be executed every 10 milliseconds, and will continue to do so until the system is turned off, the interrupt is disabled, or a new interrupt routine is installed.

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.

Position-counter words

The hardware position counters must be initialized by the robot's software. In addition, the position counters are only 16 bits wide, so the robot won't move very far before the counters overflow. So it's necessary to extend counter length with software. If we look at the counters often enough, they will not overflow. The software maintains a 32-bit position counter.

Because the counter routines must be executed many times per second, the time required to execute those routines is important. So all counter routines (and several others) have been written as CODE words. To experiment with those words, you'll have to know 80188 assembly-language programming.

The high-level words for reading the counters are ?CNT1 and ?CNT2 to read the positions of motor 1 and motor 2, respectively. The hardware causes the 16-bit counters in the 8253 IC's to decrement for each encoder count that is in the proper direction. The difference between a motor's forward and reverse counts gives the absolute position of the motor.

Following-error words

To detect a problem with the motors, it is necessary to compare actual speed with expected speed. If the two differ by more than a small percentage, an overload condition exists, so the motors could overheat and be destroyed. The following-error words constantly monitor the motors and detect a stalled motor by comparing the current motor position with the expected position. If the difference is too great the motors are turned off immediately. This also means that if you specify a value of acceleration that is too high, a following-error will be detected, and the motors will be shut down.

Numeric input

FORTH normally works with 16-bit signed integers. Such numbers can range in value from −32768 to +32767. In addition, a decimal point may be included anywhere in a number and FORTH will treat it as a signed double-precision integer with a possible range of −2,147,483,648 to +2,147,483,647. The position of the decimal point is kept in a system variable, DPL. If a number without a decimal point is entered, the system sets DPL to −1. If a number is entered with a decimal point, DPL will contain the position of the decimal point relative to the least significant digit entered. A number may have a maximum of four digits to the right of the decimal point. The FORTH system converts the input number to a signed integer representing the integer part and a signed integer representing the fractional part. The pair of single precision numbers each carries a sign bit; the numbers can be used alone or together.

Table 1 illustrates how various numbers are stored. Keep in mind the fact that the decimal-point position stored in DPL is correct only for the last number entered by the user from the keyboard. Numbers compiled into a definition do not affect the value of DPL after compilation. You must be in the decimal base (base 10) when entering numbers with decimals.

The word FIXED converts the last number input to an integer and a fraction. FIXED gets the value from DPL and puts it on the stack, then it calls (FIXED). We defined the separate word (FIXED) to do the actual conversion, because it can be made more general—it can convert any number, even if it was not entered from the keyboard.

EXTRACT strips the fraction digits from the number one by one until all have been removed. That leaves the integer part of the number on the TOS (*T*op *O*f *S*tack) with the digits beneath it. The digits are reassembled into a single number with BUILD. SCALAR produces a value that is used to adjust the fraction to the proper range. If the unscaled fraction is 9, we need to know whether it is 9000/10,000, 9/10,000, or another value.

The word FRACTION takes a fraction, an integer, and a multiplier and creates a double-precision integer. So the value −932.015 converted by FIXED is a fraction and an integer. Taking these two numbers and a multiplier of 1000 would give us the double precision number −932015 as follows:

```
932.015 FIXED 1000 FRACTION.
```

TABLE 1—NUMERIC STORAGE

Input	Value	Size	DPL
725	725	16	−1
−1	−1	32	0
1.2	12	32	1
−9.999	−9999	32	3
38.04	3804	32	2

FRACTION is used by many other words to convert values for internal use.

User-input conversion words

Several words convert user-input values to more basic units the hardware can use for the move commands.

Distances are entered in units of INCHES, FEET, MILES and DEGREES. INCHES takes the value specified and converts it to internal form. The input value and a scale factor are saved for later conversion. FEET takes a distance in feet and MILES takes a distance in miles. The scale factor is set appropriately for each word in terms of the number of inches each word represents. DEGREES calculates how far each motor must move to make the specified turn.

Speed can be entered in miles per hour by using the word MPH, inches per second by IPS, feet per second by FPS, and feet per minute by FPM. Each of those words stores the value and an appropriate scale factor for later conversion.

G converts the input value (in terms of the acceleration due to the earth's gravity, i.e., 32.2 ft/sec/sec) to a count that is used to accelerate or decelerate the motors, if necessary, each time speed is updated by the interrupt routines.

Motion

To move from one point to another, the motors must be accelerated and decelerated. By allowing the user to set a value for acceleration, deceleration, and maximum speed, the behavior of the robot can be controlled precisely.

Before a move is actually made, the software does a series of calculations to determine the top speed that can be attained, and the positions at which acceleration should end and deceleration should begin in order to attain a trapezoidal velocity curve, as shown in Fig. 1.

Calculated speed may be less than desired speed, but that is not a problem for short moves. Maximum speed will be used for moves that are long enough to allow the motors to accelerate to their maximum velocity. For short moves, acceleration is more important than maximum speed.

To perform a move, breakpoints on the trapezoidal velocity curve must be found. The points where acceleration ends and deceleration begins, as well as the end point position, must be calculated.

The robot is a speed-controlled system, so the acceleration and deceleration breakpoints must be used to calculate what speed will be achieved by accelerating at the specified value of acceleration to the breakpoint position. That new speed is saved with the breakpoint position. The same values of speed and distance are used to calculate the breakpoint where deceleration is to begin.

Trapezoidal velocity control

To perform a move, the robot must be accelerated from a speed of zero to top speed, and then decelerated at the appropriate point to arrive at the desired position. The simplest system would just set the speed of the motors, turn the motors on until the end point was reached, and then turn the motors off. That type of approach assumes instantaneous acceleration and deceleration, but in an actual

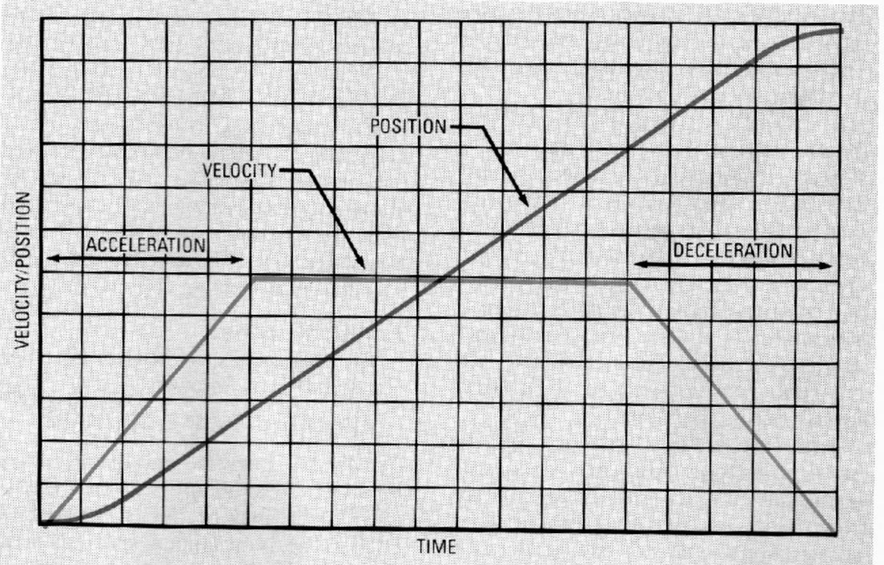

FIG. 1—ACCELERATION AND DECELERATION BREAKPOINTS must be calculated in order to move the robot from one point to another.

system it's not practical. Therefore, we have to take into account the acceleration that actually can be achieved by the system. In practical terms, acceleration might be a fraction of G, or it could be several G's, depending on the size of the motors in relation to the size of the load.

To accelerate and decelerate the robot, velocity actually must be changed many times per second. In general, the robot starts with a velocity of zero and then accelerates at a constant rate to the top speed. Then it must decelerate at a constant rate until it stops at the final position.

The velocity-versus-time profile is also shown in Fig. 1, but superimposed on the velocity trapezoid. Note that the position profile is not simply a straight line. In terms of calculus, position is the integral of speed over time. The basic equations of motion are as follows:

$$V = V_O + AT$$

$$D = V_O T + \tfrac{1}{2}AT^2$$

where V stands for velocity, A for acceleration, T for time, and D for distance. V_O refers to starting velocity.

From the previous equations we can derive an equation that describes the distance required to accelerate from one speed to another:

$$D = (V^2 - V_O^2) / (2A)$$

We can use that equation to compute the distance required to change speeds.

For a short move, the distance required to accelerate to the desired speed and then decelerate to a stop may exceed the distance to move. In such a case, deceleration must begin at some speed less than maximum.

The word DISTANCE takes the original speed and the desired speed (both in rpm) and calculates the distance in inches that will be required to change speeds. The word COUNTS changes the distance from inches to position-encoder counts. The word EXPECTED converts the user-input distance to position-encoder counts. The word SPEED converts the user-input maximum speed into rpm.

The word BREAKPOINTS calculates the positions on the velocity trapezoid to stop accelerating and begin decelerating. The acceleration and deceleration segments can't be more than half the total move distance, so the distance to accelerate from 0 to the input speed is calculated and compared to half the move distance. The minimum of these two values is then used as the acceleration distance. The breakpoint positions are saved in arrays for use during the move.

After the robot starts moving, the breakpoint positions are compared against the current position every time the control loop executes to determine when acceleration should stop and deceleration should begin. If the move distance is long enough, there will be a period during which the motors run at maximum speed. For a short move, acceleration will stop before maximum speed is attained, and deceleration will start immediately after acceleration stops.

Command language

The RCL includes a simple command set to allow movement of both the base unit and the arm.

The base-movement commands allow forward and backward motion, and left and right turns. Maximum speed, acceleration rates, and move distance may all be altered by user input. After each move is complete, a new move command can be executed. By defining FORTH words we can chain several move commands together in a motion sequence. We'll discuss such a sequence shortly.

The arm commands move the arm up and down, and open and close the jaws.

Command syntax

In general, a command consists of a device name, a speed value, a distance value, and the command:

```
[DEVICE] [n SPEED] [n DISTANCE]
COMMAND
```

where bracketed quantities indicate optional values that will be: the value entered with the command; the last value if a new value is not included; or a default value if this is the first time the particular command is issued. The value of *n* depends on the command. DEVICE may be RERB for the base unit or ARM for the arm unit.

Base commands

The general syntax for the base-movement commands is as follows:

```
[RERB] [n SPEED] [n DISTANCE]
COMMAND
```

COMMAND may be one of the following: FORWARD, BACKWARD, LEFT, or RIGHT. SPEED may be one of the following: MPH, IPS, FPS or FPM. DISTANCE may be one of the following: INCHES, FEET, MILES or DEGREES.

The command G is used to set the acceleration constant used to change speed; the constant is expressed in G's of acceleration. Any acceleration may be specified, up to the maximum acceleration the system can achieve. The acceleration may be specified in a separate command.

Arm Commands

The basic syntax for the arm-movement commands is as follows:

```
[ARM] [n DISTANCE] COMMAND
```

COMMAND may be one of the following: UP, DOWN, OPEN, or CLOSE. DISTANCE may be INCHES or FEET. DISTANCE is the amount specified in the COMMAND direction relative to the current position.

The example program shown in Listing 1 illustrates how you can combine several RCL commands to cause the robot to traverse a square. The sequence first sets the acceleration constant to 0.1 G. Then the RERB device (i. e., the base) is selected to move at 25.5 inches per second. Then it moves 3.5 feet forward and makes a left turn. The latter actions are repeated three times so that the robot ends up where it started.

Here's a short routine that moves the arm down and then back up:

```
ARM 3.1 INCHES DOWN 2 INCHES UP
```

By defining FORTH words we can create macros to perform various functions. For example, Listing 2 shows a macro that will cause the robot to traverse a box of any size.

Example program

Now let's show how the robot could be used to collect and deliver office mail. Figure 2 shows the office layout that we will use in the example program. Trays for incoming and outgoing mail are attached to the robot.

The overall sequence of operations goes like this: The robot starts from a "nest," and travels around the corridors, waiting at several locations for people to retrieve and deposit mail and then returns to the mail room.

We defined several low-level FORTH words for the program. To allow the robot to wait for different time periods, we defined several words to execute time delays. See Listing 3. The first is MS, which waits for the specified number of milliseconds. The next is SECONDS, which uses MS to delay the specified number of seconds. The last is MINUTES, which uses SECONDS to delay the specified number of minutes.

LISTING 1

```
.1G    ( 3.22 ft/sec/sec)
RERB  25.5 IPS
3.5 FEET FORWARD 90 DEGREES LEFT
3.5 FEET FORWARD 90 DEGREES LEFT
3.5 FEET FORWARD 90 DEGREES LEFT
3.5 FEET FORWARD 90 DEGREES LEFT
```

Next we define several words for convenience and to improve the readability of the source code. The robot will announce its arrival at each place it stops. That is done by sounding a beep. The word ATTENTION generates the beep.

WARNING, sounds several short beeps. It is used to avoid running over anyone when the robot is ready to move.

Since all the turns in our model office are at right angles, it's convenient to define left and right 90° turn words, TURN-LEFT and TURN-RIGHT. When the robot starts its trip it must back out of the

continued on page 80

R-E ROBOT

continued from page 60

LISTING 2

```
: BOX     ( feet ---- )
       RERB
            25.5 IPS
            2DUP (FEET) FORWARD
       90 DEGREES RIGHT
       2DUP (FEET) FORWARD
       90 DEGREES RIGHT
       2DUP (FEET) FORWARD
       90 DEGREES RIGHT
       2DUP (FEET) FORWARD
       90 DEGREES RIGHT;
```

LISTING 3

```
: MS   ( milliseconds ---- )
  0 ?DO 33 0 DO LOOP   LOOP ;

: SECONDS   ( seconds ---- )
       0 ?DO 1000 MS LOOP ;

: MINUTES   ( minutes ---- )
       0 ?DO 60 SECONDS LOOP ;
```

LISTING 4

```
: ATTENTION ( ---- )
       BEEP BEEP 1 SECONDS ;

: WARNING ( ---- )
       5 0 DO BEEP 1 SECONDS LOOP ;

: TURN-AROUND ( ---- )
       RERB 10 IPS 180 DEGREES
       LEFT ;

: TURN-LEFT ( ---- )
       90 DEGREES LEFT ;

: TURN-RIGHT ( ---- )
       90 DEGREES RIGHT ;

: AHEAD ( feet ----)
       FEET FORWARD ;

: COLLECT ( minutes ---- )
       ATTENTION MINUTES WARNING ;
```

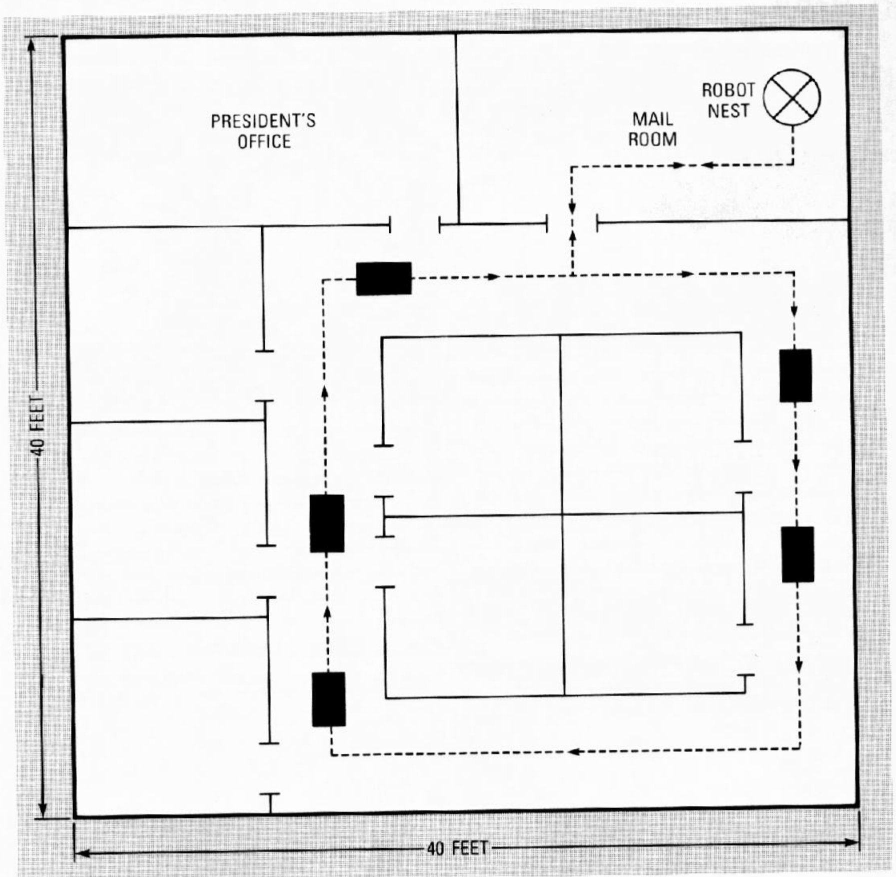

FIG. 2—MODEL OFFICE FOR THE EXPERIMENTAL TRIP program that is shown in Listing 5.

LISTING 5

```
: TRIP    ( ---- )
          WARNING       RERB          20 IPS        3 FEET
          BACKWARD      TURN-AROUND   2.5 AHEAD
          TURN-RIGHT    12 AHEAD      TURN-LEFT
          5 AHEAD       TURN-LEFT     36 IPS
          10 AHEAD      TURN-RIGHT    2 COLLECT
          7 AHEAD       2 COLLECT     8 AHEAD
          TURN-RIGHT    25 AHEAD      TURN-RIGHT
          3 AHEAD       2 COLLECT     11.5 AHEAD
          4 COLLECT     11.5 AHEAD    TURN-RIGHT
          ATTENTION     3 COLLECT     ( President)
          14 AHEAD      TURN-LEFT     4.5 AHEAD
          TURN-RIGHT    12 AHEAD      TURN-LEFT
          10 IPS        5.5 AHEAD
```

recharging area and turn around to go forward. The word TURN-AROUND makes a 180° turn. The word AHEAD is shorthand for a forward move. COLLECT combines the ATTENTION, WAITING, and WARNING functions, because we always use them together. The definitions of those words are shown in Listing 4.

A trip consists of backing out of the recharger and exiting the mail room, making a clockwise trip around the office, stopping at several points (including a long stop at the president's office), and finally returning to the mail room. The word TRIP executes the entire program; it is shown in Listing 5.

TRIP only causes the robot to make one excursion around the office, but we want

LISTING 6

```
: MAILBOT  ( ---- )
        8.5 AM WAIT-UNTIL TRIP
        9.5 AM WAIT-UNTIL TRIP
       10.5 AM WAIT-UNTIL TRIP
       11.5 AM WAIT-UNTIL TRIP
        1.5 PM WAIT-UNTIL TRIP
        2.5 PM WAIT-UNTIL TRIP
        3.5 PM WAIT-UNTIL TRIP
        4.5 PM WAIT-UNTIL TRIP
```

the robot to make several trips during the day, without having to tell it to do so each time. We can schedule the trips when desired using the words AM, PM, and WAIT-UNTIL. WAIT-UNTIL simply waits in a delay loop until the current time is identical to the desired time. AM and PM set the desired time. Time is specified in hours, so minutes must be expressed as fractional hours. For example, 8.5 AM is 8:30 am. The entire MAILBOT program is shown in Listing 6.

You can extend RCL to deal with additional hardware and to provide greater software flexibility. FORTH gives you the freedom to experiment and add to the capabilities of the system. **R-E**