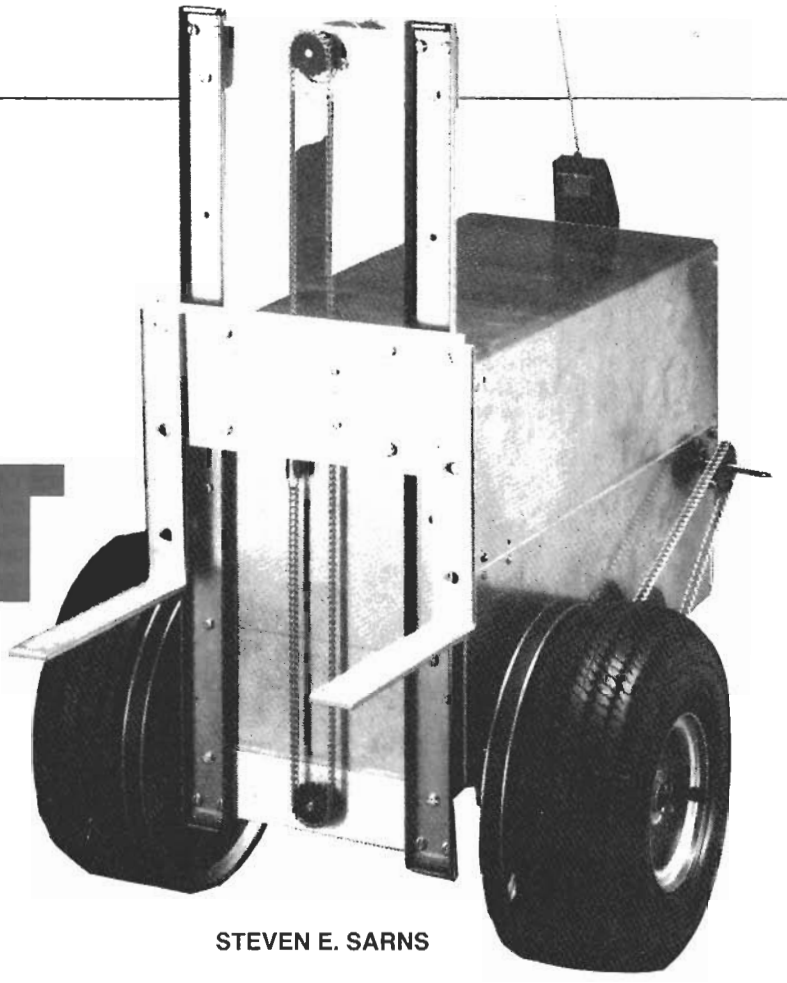
BUILD THIS

R-E ROBOT



More on the electronics that makes our robot go.

STEVEN E. SARNS

Part 7 THIS MONTH WE'LL continue our look at the robot's control electronics. You'll need to refer to the schematic that appeared last time ("R-E Robot, Part 6," **Radio-Electronics**, May 1987), so have it handy as we proceed.

Analog-to-digital converter

An *A*nalog-to-*D*igital *C*onverter (ADC) is required to convert information about the torque output of the drive motors and the state of the batteries to a digital signal that can be processed by the RPC (*R*obotic *P*ersonal *C*omputer). Additionally, an ADC is needed to convert the outputs of environmental sensors to digital form; almost all such sensors have analog outputs.

Because it is easily interfaced with our microprocessor bus, the ADC we selected is the ADC0804 (National). That IC is self-clocking, uses an internal ratiometric reference, and needs only +5-volts DC to operate. The input-voltage range of that IC is adjustable; we've set it for 0 to +5-volts DC.

The ADC's input channel is selected using two 4051 8-channel multiplexer IC's. The input channels are allocated as follows: Two channels are connected to amplifiers monitoring each drive motor's current. Two channels are connected to the RERBUS (*R*adio-*E*lectronics *R*obot *BUS*) connector; we'll discuss the structure of the RERBUS in a moment. Two channels are dedicated to internal heat-sensing. Eight channels are available at the user connector for external sensors or other peripherals you may add later.

The analog-to-digital conversion process is started by writing to port 15ØH. The desired input channel is selected by placing the appropriate data on the data bus and then writing it to IC4, the 74LS377 parallel latch discussed last time. About 200 μs later, the requested data is placed on the data bus by reading port 15ØH.

For example, let's assume that we wanted to check on the battery. Battery conditon is monitored on channel 3. By selecting that channel we can learn of the state of the battery's charge; if it has dropped too low the appropriate action can be taken. The following line of computer code, written in FORTH, returns the state of the battery:

```
: BATT? 2 150 PC! 2 DELAY 150 PC@ ;
```

Speaking of FORTH, we realize, of course, that we have not discussed the language in any detail yet. Undertaking such a task would require devoting several complete *issues* of **Radio-Electronics**, and the job would still be incomplete, at best. Therefore, for space reasons, we must assume that the reader is familiar with the language. On that premise, when we present FORTH routines in future installments of this series, we will explain what the routine does in general terms, but not the function of each term or line.

RERBUS user interface

If circuit designers always knew in the beginning what would be needed in the end, the concept of a bus would never have been developed. Since we lack such foresight, however, a simple interface bus is provided to allow you to expand the robot easily and at low cost. That interface bus has been named the RERBUS; it uses 26-conductor ribbon cable that carries 8 data lines, 4 address lines, 2 control lines, and power.

The RERBUS interface is derived from IC7, a 74LS374 output latch. That latch was chosen over the 74LS377 used earlier because it has greater output drive capacity. The price of that added drive capacity is the need to add an additional OR gate (IC15-b, ¼74LS32) to the clock line. A second 74LS374 (IC8) is used to buffer the address and control lines for the interface. Data is read from the interface using IC6, a 74LS541 octal buffer/line driver.

The advantage of that implementation is that we have complete control over the access time of any circuits connected to the bus. The read or write lines can be enabled as long as the external circuits require. That is particularly important when a flexible cable is used instead of a backplane because a flexible-cable bus requires slow access times because of its high inter-conductor capacitance.

Motor controllers

Before describing our motor-control

circuit, remember that we are dealing with DC brush-type torque motors. Those motors are characterized by poor speed regulation, lack of an integral tachometer, and, consequently, low-cost; the last characteristic explains why they were selected. Stepper motors with the torque capability we need would have been difficult to locate and would have cost much more. Brushless motors, though less expensive, require so much support circuitry that they wind up costing as much to use as steppers. Servo motors with an integral analog tachometer could have been used, but they too are much more expensive than lowly torque motors. Remember, any automobile starter motor will deliver around ½ horsepower, just about what we need.

Motor controllers fall into two general classifications; linear and switching. In a linear controller, the terminal voltage of the motor is varied to keep the motor speed constant. A switching controller keeps motor speed constant by regulating the duty cycle of the power applied to the motor. Bear in mind that our motor controllers may have to handle up to 500 watts. Examining the power-dissipation requirements of typical series-pass transistors we see that there will be a power-dissipation problem if linear controllers are used. The worst case is at high-torque loads and low motor speeds. Then the series-pass transistors must conduct maximum current while withstanding nearly full battery voltage. The result is over 1 kW of power dissipation in the series-pass transistors.

A switching controller, on the other hand, is either fully conducting or fully off. In either case, power dissipated by the series-pass transistors is zero. Actually, it'a very near zero. During transitions between off-to-on and on-to-off, power is lost, but the total dissipation is many orders of magnitude lower than that of a linear controller.

Some form of feedback is required in all motor-control circuits. That can be derived from an analog generator attached to the motor shaft, as is done with servo motors. Sometimes the motor itself is used as an analog tachometer by measuring the back EMF generated. In our case, optical encoders will be attached to the motor shafts to obtain motor-speed information. The information from the encoders is brought to the controller via connectors PL2 (right motor) and PL6 (left motor); it is decoded by a pair of 74LS74 dual flip-flops (IC28 and IC29).

The final stage of each motor controller is the output driver. That stage must deliver the power to the motor. It must also be capable of reversing and stopping the motor. One method of doing that requires two power supplies of equal voltage and opposite polarity. The motor is connected to the positive supply to run in one direction and to the negative supply to operate in the opposite direction. That solution does not use batteries efficiently, so it is not used.

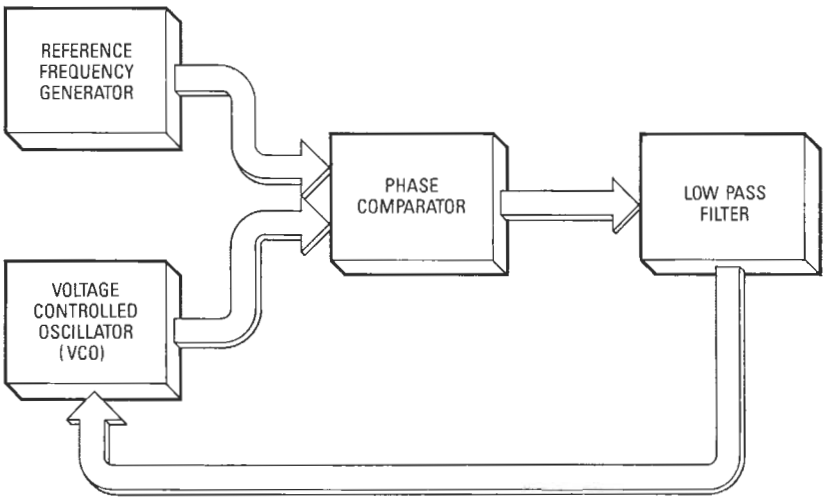


FIG. 1—THE PLL IS THE KEY CIRCUIT comprising the motor controllers. Here, the operation of a PLL is shown in block diagram form.

Another alternative would be to use an H bridge. In that scheme each motor is surrounded by four switches (transistors) that direct the current through the motor in the desired direction. Although efficient in terms of power usage, the circuit requires four high-power transistors and associated circuitry, all of which increases the cost.

Finally, the classic approach is to connect the motor to the center arm of two SPDT relays and to use the relays to establish the direction of the current in the motor. That approach is simple and inexpensive; it is the approach used in our robot.

Design considerations

It is important that the motor controller be able to operate as independently as possible. We could design a motor-control system in which a microprocessor controls the power delivered to the drive transistors and counts the incremental revolutions of the drive wheels. That is an ideal application for a slave processor, and, in fact, that is how the *HERO 2000* works. Unfortunately, however, developing a dedicated slave-microprocessor system for our robot was a luxury that we could not afford.

On the other hand, it is important that we do not load the RPC down with too many tasks. Therefore, our motor controller will have to be smart enough to operate without constant attention from the microprocessor.

We also need an accurate system that can keep track of how far the robot has moved. The drive wheels will be used to steer the robot, so, not only must the motors be capable of accurate differential actions, they must be well-behaved at low (maneuvering) speeds. The robot's arm has just one degree of freedom. The other two degrees of freedom are obtained from the base unit. Therefore, the motor controls must be able to move the unit in small forward, backward, or rotational steps to enable the arm to grasp its target.

The wheel-derived distance will be used as our first-approximation navigational system. However, no matter how accurate our wheel-based navigation is, there is no solution to the "bump in the road" problem. For example, if our robot were traveling in a straight line, maintained by equal distance traveled by each wheel, and one wheel went over a bump, the robot would turn towards the bump. That is because the wheel that went over the bump went farther to achieve the same linear distance that the wheel that missed the bump. In spite of that problem, it would simplify later problems if the robot were able to go where it is told reliably.

The motor controller designed for the robot meets those specifications. We have designed a totally digital, *P*hase-*L*ocked *L*oop (PLL) motor-control system that uses a single pulse-width modulated tran-

sistor. The controller board has two such circuits, one for each drive motor. Directional control is achieved with two relays, a big cost saving compared to the H-bridge approach at high current levels. Feedback is fully quadrature-decoded assuring accurate positional information. Torque information is available to the computer so that stall or high-load conditions can be detected. And it is all done economically.

Now, let's examine the major building blocks of the system.

The PLL

The key to the design of the motor controller is the use of a phased-locked loop. A block diagram of a PLL is shown in Fig. 1. Essentially, a PLL consists of a reference-frequency source, a frequency comparator and a variable-frequency signal that is produced using a *V*oltage *C*ontrolled *O*scillator (VCO). In operation, the frequency comparator constantly compares the variable-frequency signal with the reference-frequency signal and adjusts the variable-frequency signal so that the two match.

By now you may be getting an idea about how our controller works. A reference frequency that is proportional to the desired speed is generated. That frequency is compared with the the output of an optical encoder that is linked to the motor; the encoder plays the part of the VCO in this PLL. If the controlled voltage is lower than the reference voltage, the PLL circuit will turn the drive transistor on more; if it is higher, the PLL will turn the drive transistor on less.

A block diagram of the system is shown in Fig. 2. As mentioned, two such systems are required by the robot, one for each drive motor. To keep things simpler in the following discussion, we will examine the circuit for the left motor only; the right-motor controller works identically.

Reference frequency

The reference frequency is derived from the 80188 microprocessor clock (on the RPC) and one of the three independent timers of IC13, an 8253 programmable 16-bit counter. The clock signal is divided by 16 to produce a 500-kHz clock for the 8253. The 8253 then divides the input clock by a 16-bit number to produce an output frequency between 2 Hz and 250 kHz, depending on the number selected. That output is used as the reference frequency by the PLL. The reference frequency can be turned on and off via pin 11, GØ, by writing to port 124H. When the reference frequency is off, the motor must stop completely.

Frequency comparator

The frequency comparator used by our PLL is the internal type-II comparator in the 4046 PLL, IC23. A sophisticated digital-memory circuit outputs a 1 (logic high) if the phase of the reference signal leads that of the controlled signal. A Ø (logic low) is output if the phase of the reference signal lags that of the controlled signal. As you can see, the circuit is actually a phase comparator. However, in our description we will continue to refer to the circuit as a frequency comparator because in our system it is used to control the frequency of the optical-encoder feedback signal rather than the phase.

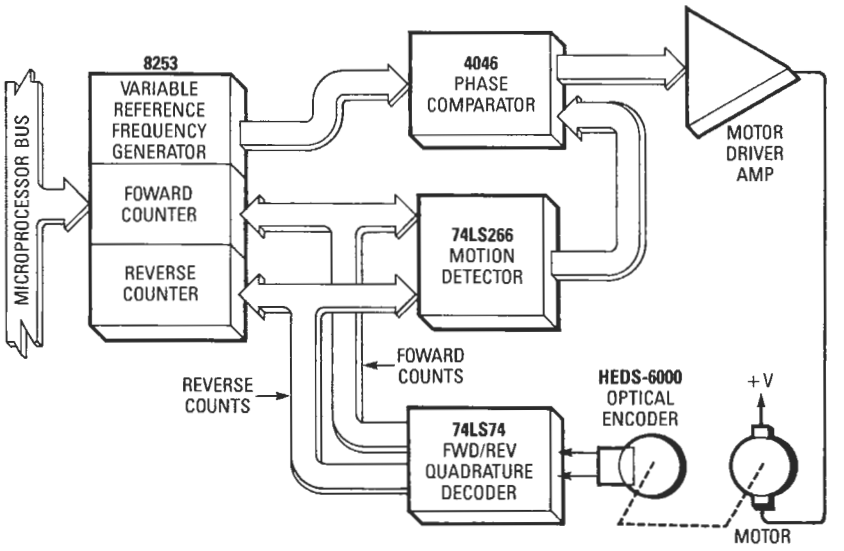


FIG. 2—IN OUR MOTOR CONTROLLER, the optical encoder takes the place of a VCO in the PLL circuit. As the voltage to the motor increases, the output frequency of the encoder increases.

The output of the frequency comparator is a pulse train whose duty cycle will vary from 0 to 100%, depending upon the difference in phase between the reference frequency and the optical-encoder outputs. The frequency of that pulse train will be equal to the reference frequency. If you examine the output of the 4046 with an oscilloscope while applying a load to the motor you will see the duty cycle of the comparator's output increase, but its frequency will remain the same.

Motor driver

Like the reference frequency, the motor driver is gated on and off via port 124H. When the driver is gated on, the output of NAND gate IC24-a drives transistor Q1 into saturation. That 2N3906 small-signal PNP transistor drives a Darlington pair. Note that rather than a single unit, in our circuit we use a Darlington pair fashioned from discrete transistors (Q2 and Q3). That approach is used because it offers you much flexibility in matching drive transistors and motors. The 2N3772 transistor used for Q3 in our circuit is rated at 30 amperes; that means it can deliver up to 1 kilowatt to the motor at the maximum recommended battery voltage (36 volts × 30 amps).

The motor is connected to the center position of two SPDT power relays. A single flyback diode, D2, protects the driver transistor from damage due to motor inductance. The use of two relays instead of an H-bridge output-driver circuit allows us to reverse motor directions, yet saves three expensive power transistors, several level-matching components, and three flyback diodes. Maximum dynamic braking of the motors is achieved when both relays are in the de-energized position, an important consideration when the robot is parked.

It is possible to perform regenerative braking with the circuit, wherein the kinetic energy of the robot in motion is used to generate electricity in the motor and charge the batteries. However, due to the complexity of the control algorithms and the small amount of energy recovered, we did not implement that feature.

Motor current is sensed at the 0.01-ohm resistor, R21, amplified and filtered by the LM358 operational amplifier, IC20, and presented to the ADC. In a given motor, the relationship between load and current is linear irrespective of motor speed. Once that relationship has been determined, the information can be used by the RPC to sense the grade or surface that the robot is operating on.

Shaft encoders.

We used a Hewlett Packard HEDS 6000 optical encoder for our robot. That unit has a quadrature output of 500 counts-per-revolution.

Note that the HEDS 6000 is one of the most expensive components in the system. You have two choices if you do not want to follow our lead in using that encoder. You can make your own encoder, either with or without quadrature outputs,

continued on page 90

continued from page 60

or you can investigate the use of incremental shaft encoders.

With the quadrature system, two pulse trains, 90° out of phase, are produced. Motor-direction information can be obtained by examining the relationship of the two waveforms; that is, to see which one is leading and which one is lagging. With proper decoding, spurious counts that may occur when the robot stalls or the motor shaft vibrates will be rejected.

A non-quadrature optical-feedback system can be implemented simply by attaching a disk with holes drilled around its circumference to the motor's backshaft. A simple optical encoder can then be used to keep track of shaft rotation in the conventional way. Be sure that you drill as many holes as possible, for improved low-speed performance. You will have to condition the encoder's output signal to match TTL levels; that can be done with an LM393 comparator.

A quadrature encoder can be made from two optical sensors positioned so that the output waveforms are 90° out of phase. The sensors are properly positioned when the distance between the optical centers is such that when one hole is centered on one encoder, another hole is covering 50% of the other encoder. With a typical system using using 0.25-inch sensor spacing, the hole pattern consists of ⅜-inch holes on ¾-inch centers. That results in 12 holes on a 1.43-inch radius. That hole pattern is sufficient to give satisfactory feedback for operation at all but the lowest speeds.

Once again, the outputs of the two optical encoders should be conditioned with LM393 comparators. The circuit is then interfaced with the controller board and the output signals are decoded.

SOURCES

The following are available from Vesta Technology, 7100 W. 44th St., Wheatridge, CO 80033 (303-422-8088): Bare RE-Robot controller board, $41; assembled and tested RE-Robot controller board, $200; bare RPC board, $41; assembled and tested RPC, fully populated for the robot function, $294. Add $8.00 shipping per board ordered. Colorado residents add appropriate sales tax. Mastercard and Visa accepted.

Optical endocers (100 counts/revolution, quadrature output) are available from EMC Corp., 373 Hillsboro Way, Goleta, CA 93117 (805-968-3060) for $40 each. California residents must add appropriate sales tax.

Terminal count

The progress of the motor is monitored using two counters of the 8253. One counter is clocked by the forward progress of the robot; the other counter is clocked by the reverse progress.

As mentioned, one of the features of the 8253 is that it is programmable. By loading the appropriate control word into the IC, the counters can be set to operate in one of six modes. For example, to generate the reference frequency we set up one of the 8253's counters to divide by a 16-bit number. That is the IC's mode 2. Here, we need to set up the remaining counters to operate in one of the IC's other modes, interrupt on terminal count (mode 0). In mode 0, the output goes high when the accumulated count has reached a value programmed into the counter. That output may be connected to a digital input for polled operation, but is connected to an interrupt input on the RPC for interrupt-driven operation. The counters can be interrogated by the RPC at any time during motor operations to ensure that the motor has not stalled or is not vibrating.

That's all for now. Next time we'll show you how to build and mount the board. We'll also look at some software considerations. **R-E**