Mobile Robots
Inspiration to Implementation

Second Edition

Joseph L. Jones
Bruce A. Seiger
Anita M. Flynn

A K Peters
Natick, Massachusetts
What happens if your circuit does not behave properly? The best way to proceed is to go back to square one and find something that does work. See if power is getting to all your chips. The batteries could need recharging or possibly an IC's pin was bent and missed mating with a hole in a socket. Check the power supply on an oscilloscope to make sure it is not corrupted with noise. It is good practice to add capacitors across the supply and across the power and ground pins of digital ICs.

If you are debugging a MC68HC11 circuit, start by checking the clock. Pin 5, called E, should be a square wave at a frequency of one-fourth the crystal frequency. Next, check the reset pulse on pin 17 as you depress the reset button. It should rise cleanly without glitches. Check the interrupt pins to make sure that they are normally high. If they are left unconnected, they may float and initiate random interrupts. Check that the processor is set up in the correct mode by observing the signals on pins 2 and 3, MODA and MODB, while you press the reset button. These signals are valid and read by the processor for just a few cycles after reset.

If other parts of your circuit are misbehaving, try the technique of “divide and conquer.” Remove any load from the pin and check again. Be systematic and thorough; always start by finding a point in your circuit that is behaving as designed and gradually debug subsequent portions of the circuit.

Finally, think about connectors. The more subsystems you add to your robot, the more interconnecting is required. Complexity can increase quickly if many sensors and actuators are geographically scattered around the perimeter of the robot, as can be seen in Figure 4.13. (This is a photograph of work in progress on a vacuum cleaner robot built by Masaki Yamamoto at the MIT Mobile Robot Lab.) Make your first robot simple, and key your connectors so they cannot accidentally be inserted backward. One of the most common sources of problems are loose or flaky connections. Be neat, and build reliable connectors!

5

Sensors

5.1 Achieving Perception

As humans, we often take for granted our amazing perceptual systems. We see a cup sitting on a table, automatically reach out to pick it up and think nothing of it. At least, we are not aware of thinking much of it. In fact, accomplishing the simple task of drinking from a cup requires a complex interplay of sensing, interpretation, cognition, and coordination, which we understand only minimally.

Thus, instilling human-level performance in a robot has turned out to be tremendously difficult. A computer program has now beat the reigning world champion at chess but a program that reliably recognizes, say, a chair in an arbitrary scene still does not exist. The parallel computer inside each of our heads devotes large chunks of grey matter to the problems of perception and manipulation.

5.1.1 Transducing versus Understanding

While we would like our robot to understand and be aware of its environment, in actuality, a robot is limited by the sensors we give it and the software we write for it. Sensing is not perceiving. Sensors
are merely transducers that convert some physical phenomena into electrical signals that the microprocessor can read. This might be done by using an analog-to-digital (A/D) converter onboard the microprocessor, by loading a value from an input/output (I/O) port, or by using an external interrupt. Typically, there needs to be some interface electronics between the sensor and the microprocessor to condition or amplify the signal.

5.1.2 Levels of Abstraction

With software, we can create different levels of abstraction, or abstraction barriers, to help us as programmers think about sensor data in different ways. At the highest level, the intelligence system, in order to seem clever, needs to have some variables to juggle: Is it dark in this room? Did a person just walk in? Is there a wall to the left?

However, the only questions the robot is able to ask are ones such as: Did the resistance fall in the photosensor? Did the voltage from the pyroelectric sensor connected to the fourth A/D channel go above threshold? Did the output of the near-infrared proximity detector change from low to high?

Nevertheless, it is possible to install many capabilities in a mobile robot. Figure 5.1 shows a five-foot-tall mobile sentry robot called Robart II, built at the Naval Ocean Systems Center. Robart II serves as a mobile sentry robot (patrolling a building, avoiding obstacles, watching for intruders) and is able to find its recharging station and plug itself in. This robot contains a very large number of sensors, such as near-infrared proximity detectors for obstacle avoidance, sonar rangefinders for localization, microwave sensors for motion sensing, pyroelectric sensors for detecting intruders and temperature, and earthquake and flood sensors for disaster identification.

Another mobile robot covered with sensors is Attila, shown in Figure 5.2. Attila is a shoebox-sized, six-legged robot designed as a rough-terrain explorer. Sensors on the legs are used for detecting obstacles and stepping over them. There are strain gauge force sensors along the shins for detecting collisions, potentiometers on the joint motors for position calibration, and contact sensors on the feet for ascertaining stable footholds. A number of sensors are also mounted...
on the chassis. Whiskers protrude from the front for collision detection, a long-range, near-infrared sensor measures clear space, and a small camera gathers images.

### 5.2 Interfacing Sensors

In this chapter, we will focus on many types of simple sensors and how to interface them to a microprocessor. Threaded throughout the chapter are various examples of sensor-interface electronics and sensor-driver routines. A variety of sensors (such as photosensors, bump switches, microphones, pyroelectric people sensors, near-infrared proximity sensors, sonar rangefinders, bend sensors, gyroscopes, accelerometers, force sensors, compasses, and cameras) can be inexpensively acquired and interfaced to a small mobile robot.

By the end of this chapter, you will be able to understand most of the second half of Rug Warrior's "brain," which is illustrated in Figure 5.3. This brain constitutes the sensors and their interface electronics that fit (along with the computer described in Chapter 3) onto Rug Warrior's 3.4” x 4.5” board. Part of Figure 5.3, the motor-driver circuitry, will be discussed later in the chapter on motors.

Throughout this chapter, as each type of sensor is explained, partial schematics are given that assume the basic MC68HC11A0 circuit is already built. The interface electronics are shown connected to a specific MC68HC11A0 I/O pin, analog-to-digital port, or counter/timer pin, and software fragments illustrate how to convert sensor readings into internal variables. If you would like to see the entire Rug Warrior schematic all in one place, refer to Appendix A.

Most of Rug Warrior's sensors are mounted directly onto the circuit board, which is left exposed. This is to circumvent the need to make connectors and wiring harnesses to any outer cover of the robot. Many of the sensors can be seen in Figure 5.4. The pyroelectric sensor, with a cone-shaped holder for its plastic fresnel lens, points upward in the center of the board. The square aluminum package just in front of it is a Sharp near-infrared detector. Two near-infrared LED emitters are mounted on either side of the Sharp detector. Just to the outside of both LEDs are cadmium sulfide photoresistors for light detection.

![Figure 5.3. In this chapter we will discuss the sensors illustrated on this schematic of Rug Warrior's sensors and actuators: the near-infrared proximity sensors at top left, the three bumper sensors at top right and the shaft encoders, microphone, photoresistors and pyroelectric sensor shown in the center.](image-url)
5.2 Interfacing Sensors

A few of Rug Warrior's sensors can be seen more clearly in Figure 5.5. The microphone on the left is available at Radio Shack. The microswitches in the center are of the type used on a bump skirt to detect collisions. Just to the right of the microswitches is a mercury tilt switch, which is not actually used on Rug Warrior. If the bulb is tilted, the mercury flows to cover two contacts, thus acting as a switch. Such a sensor is useful for detecting if your robot is climbing a ramp. At the far right is a Radio Shack cadmium sulfide photoresistor.

5.2.1 Software Drivers

Once a set of sensors has been selected and the proper interface circuitry has been designed to connect your sensors to a microprocessor, the microprocessor has to be programmed to read the sensors. These pieces of code are often written in assembly language and are known as software drivers.

Software drivers are pieces of code that provide a well-defined interface between a hardware device and a program that needs to use the device. We will describe here several examples of driver code that make the hardware simple to use. Where it is instructive to do so, we will implement our examples of software drivers in both assembly language and the C language. The syntax we use for our sample assembly language programs closely follows Motorola's 6811 assembly language. One notable exception is that, in our syntax, unless set off by spaces, we use "~" as a normal character rather than the subtraction operator.

Software drivers deal with the hardware-software interface. These routines might constantly poll an A/D pin, waiting for the trigger from a pyroelectric sensor, or they might be implemented as interrupt handlers that are only called when the return signal from, say, a near-infrared proximity sensor goes high. Sensor-driver code might take this data and store it in a memory location. Used in this way, the output from the sensor can be thought of as the value of a variable or as a flag. This data then becomes fodder for a higher-level abstraction. For instance, another part of the intelligence system might use such a flag or variable to trigger a behavior or perhaps combine it first with other information into a type of virtual sensor.
5.2 Interfacing Sensors

5.2.2 Sensitivity and Range

Two important concepts to understand when analyzing any sensor are sensitivity and range. Sensitivity is a measure of the degree to which the output signal changes as the measured quantity changes. Let's call the sensor output \( r \) and the measured physical quantity \( x \). For example, a photodetector might output a voltage of say, 0.87 volts \( (v) \) when it is struck by \( 2.3 \times 10^{10} \) photons per second \( (x) \). The sensitivity of the sensor is defined by:

\[
\frac{\Delta r}{r} = S \frac{\Delta x}{x}
\]

Here, a small change in the measured quantity, \( \Delta x \), is related to a small change in the sensor response, \( \Delta r \), by the sensitivity, \( S \).

A sensing device reacts to varying levels of some physical stimulus by outputting a characteristic voltage (or current, frequency, etc.). Typically, the circuitry associated with the sensor then amplifies or otherwise transforms this voltage and feeds it into an analog-to-digital converter connected to a microprocessor. The A/D converter is sensitive only to a limited range of voltages, often 0 to 5 V. In the case of the 8-bit A/D converter built into the MC68HC11, the voltage is then converted into one of 256 discrete levels. This, then, is the microprocessor's window on the world. No matter how complex and subtle, all phenomena are collapsed into a number, or set of numbers, with values between 0 and 255.

It is, therefore, important to consider carefully how a physical quantity is transformed into a digital value accessible to the microprocessor. Figures 5.6 and 5.7 illustrate two options—both linear and logarithmic mappings of voltages to numbers.

Suppose the motion of a robot arm is restricted to a well-defined range, 0 to 90 degrees. We wish to know the position of the arm with equal sensitivity over all portions of its range. Under these circumstances, a linear mapping of joint angles to A/D readings is provided by the simple potentiometer circuit shown in Figure 5.6. Figure 5.7(a) shows the mapping.

Figure 5.6. The potentiometer in (a) is connected to the joint of a robot arm. The voltage across the network between point \( A \) and ground has a linear relationship to the angle to which the joint is set. The photodiode in (b) produces a linear response to a very wide range of illumination levels. After the signal from the diode has been amplified by the logarithmic amplifier however, the voltage at \( B \) is proportional to the logarithm of the illumination.

Figure 5.7. It is always necessary to consider how the quantity measured by a sensor will be mapped into the range of digital values available to the microprocessor. (a) The linear mapping illustrated here would map an arm joint angle of 0° from the vertical to the number 0 and an angle of 90° to 255. (b) A linear mapping of illumination units to numbers would map 250 illumination units to the number 64 and 1,000 illumination units to 255. (c) A logarithmic mapping over a larger dynamic range, from 0.1 illumination units to 1,000 illumination units for an 8-bit (0 to 255) A/D converter.
5.3 Light Sensors

Visible light sensors and infrared sensors span a broad spectrum of complexity. Photocells are among the easiest of all sensors to interface to a microprocessor, and the interpretation of a photocell output is straightforward. Video cameras, on the other hand, require a good deal of specialized circuitry to make their outputs compatible with a microprocessor, and the complex images cameras record are anything but easy to interpret.

5.3.1 Photoresistors

Light sensors can enable robot behaviors such as hiding in the dark, playing tag with a flashlight, and moving toward a beacon. Simple light sensors can be purchased as photoresistors, photodiodes, or phototransistors. A photoresistor (or photocell) is easy to interface to a microprocessor. As shown in Figure 5.8, only one external component is needed. Photoresistors are simply variable resistors in many ways similar to potentiometers, except that the resistance change is caused by a change in light level rather than by turning a knob.

Phototransistors provide greater sensitivity to light than do photoresistors. A phototransistor is almost as easy to interface to a microprocessor as a photoresistor. Figure 5.9 illustrates a simple configuration using a phototransistor.

Photodiodes possess great sensitivity, produce a linear signal over a very wide range of light levels, and respond rapidly to changes in illumination. This makes them useful in communication systems for detecting modulated light; the remote control receiver in almost every TV, stereo, and compact disk (CD) player on the market makes...
use of a photodiode. The output of a photodiode does, however, require amplification before it can be used by a microprocessor.

Because the photoresistor is so useful and easy to incorporate, we will further analyze a practical circuit for connecting one to a microcontroller. Consider the circuit for the left photoresistor in Figure 5.8. Here, two resistances form what is called a voltage divider. The total resistance in this circuit, $R_T$, is the sum of the individual resistances: $R_T = R + R_L$. According to Ohm's law, the current, $I$, through the circuit is $I = V/R_T$. In order for the A/D converter in the microcontroller to measure a voltage, some current must flow into pin PE1. However, because the MC68HC11 has high-impedance inputs, the amount of current required is negligible compared to the currents in the rest of the circuit. In this case, the connection to PE1 can be ignored while analyzing the voltage divider. Thus, the voltage present on PE1 is:

$$V_{PE1} = IR_L$$

The resistance of the photoresistor falls as the light level increases. This means that the voltage at PE1 decreases. Substituting for $I$, we get:

$$V_{PE1} = \frac{R_L}{R+R_L}V$$

The 8-bit A/D converter in the MC68HC11 maps the variable voltage, $V_{PE1}$, into the range 0 to 255. Although the mapping provided by the simple voltage-divider circuit is not logarithmic, as was recommended for light sensors in Subsection 5.2.2, a useful output can nevertheless be extracted. A good compromise between sensitivity and range will be achieved if the resistance, $R_L$, is set to the same value as the resistance exhibited by the photoresistor when exposed to the light level in the middle of the range of light levels in which the robot must operate.

Typically, photoresistors are made from cadmium sulfide (CdS). Hamamatsu, Clairex, and EG&G manufacture CdS photoresistors; often, photoresistors can be purchased at electronic stores. In addition, most of the semiconductor manufacturers have optoelectronic divisions that fabricate silicon photodiodes and phototransistors. Try Hewlett-Packard, Motorola, Texas Instruments, National Semiconductor, NEC, Siemens and Sharp. Ask for the optoelectronics data book for each company. Texas Instruments sells a TSL250 photodiode with integrated on-chip amplifier. Assemblies of LEDs and photodetectors for encoders or optical switches can be obtained from Omron, Optek, HEI, and Digi-Key. Some companies, such as Hamamatsu and Centronic, also sell photosensor array chips and imagers, although these can be somewhat more expensive. The Texas Instruments TSL214 is a low-cost, 64-element photodiode array.

A Software Driver for Photoresistors

Here, we take a moment to explain in some detail how to configure the analog-to-digital converter and program a software driver for photoresistors. These tasks encompass both the capabilities of the hardware and the responsibilities of the programmer.

As was mentioned in Chapter 3, port E of the MC68HC11 can be configured as either an 8-bit input port or an 8-channel analog-to-digital converter. Internally, there is only one A/D circuit for the entire port and only four registers to store results from the eight channels. Thus, to achieve the full potential of the A/D port, a certain software protocol must be enforced.

First, the voltage reference pins on the MC68HC11 (VRH and VRL) must be set to calibrate the hardware. If these pins are set to +5 V and GND, respectively, then A/D result values of 255 and 0 will correspond to those limits, respectively. Voltages between the limits are proportionately scaled. Two control registers, ADCTL and OPTION, are used to configure the mode of conversion. Reference
5.3 Light Sensors

check_result
   brclr adctl #10000000 check_result; Wait in tight loop
   ldaa adrl ; Get value from rt photocell
   staaa ph-right ; Save right value
   ldaa adrl ; Get value from lf photocell
   staaa ph-left ; Save left value
   rts ; Return to calling code

The C version of the photocell code is somewhat simpler:

int ph_right = 0; /* Variable for right photocell data */
int ph_left = 0; /* Variable for left photocell data */

void update_photo()
{
   poke(option,0b100000000); /* Enable A/D system */
   poke(adctl,0b00010000); /* Begin conversion */
   while( (peek(adctl) & 0b10000000) == 0 )
   {
      /* Wait until conversion finished */
      ph_left=peek(adrl); /* Get and store A/D channel 1 */
      ph_right=peek(adrl2); /* Get and store A/D channel 2 */
   }
}

In both versions, we first designate locations where the results of the A/D conversions will be stored: ph-right, ph-left for the assembly version and ph_right, ph_left for the C version. We enable the A/D system by writing the proper value to the OPTION register; then we begin a conversion by writing to the ADCTL register. The next part of both programs polls the conversion-complete bit of the ADCTL register, remaining in a tight loop until the conversion flag is set by the internal hardware of the A/D. Finally, the results of the conversion are moved from the result registers, ADRI and ADRL, to the designated locations.

To learn the details of which registers and which bits control the various functions of the A/D converter and the microprocessor's other systems, you should really consult the documentation for the MC68HC11A0.

5.3.2 Near-Infrared Proximity Detectors

Following behaviors are easy to implement on a mobile robot. Using a sonar rangefinder to measure range to a person and then staying
5. Sensors

5.3 Light Sensors

Figure 5.10. The robot can be made to follow a wall using two detect/no-detect infrared sensors, A and B. When neither sensor detects an obstacle, the robot arcs to the right, searching for a wall. When only sensor B detects something, the robot moves forward. When sensor A detects an obstacle, either alone or with sensor B, the robot turns left.

An infrared emitter and detector pair are illustrated in Figure 5.11. The emitter (top) is an LED made from gallium arsenide, which emits near-infrared energy at 880 nm. Both emitters and near-infrared detectors (photodiodes and phototransistors) can be purchased from nearly any semiconductor company that has an optoelectronics division (Siemens, Motorola, Hewlett-Packard, etc.). Radio Shack also carries both near-infrared LEDs and near-infrared phototransistors. More conveniently, Sharp sells two detector packages the GP1U52X and the newer IS1U60, that contain integrated amplifiers, filters, and a limiter. The GP1U52X unit is distributed by Radio Shack, Sterling Electronics and a number of others.

The Sharp detector responds to a modulated carrier put out by the near-infrared LED. This means that the programmer is responsible for blinking the LED in a certain pattern such that the detector will respond. This modulated carrier protocol increases the signal-to-noise ratio. A minimalist circuit (only one IC is needed, a 74HC04 inverter), which achieves an interface of such a proximity sensor to a MC68HC11, is shown in Figure 5.12.

The Sharp detector responds to a carrier frequency of 40 kHz. A 40 kHz frequency means the LED is blinked on and off with a period of 25 microseconds (μs). According to the device specification, this
and PD3 of port D are asserted. Thus, the programmer is responsible for turning these on and off for 600μs each. The Sharp detector outputs a low signal when it detects reflected energy and a high signal when it detects nothing. Figure 5.13 shows the low signal asserted by the Sharp detector when an object reflects energy from the emitter back to the detector. The output of the Sharp detector is a digital signal, either 0 or 5 V. Consequently, pin PE4 of the MC68HC11 can be used in the normal digital input mode. The A/D converter capability is not necessary here.

The circuit that controls the emitters is a rather odd one. It is uncommon to have the outputs of inverters connected together. Normally, an AND gate would be used to allow signals PD2 and PD3 to modulate the oscillator output. (An AND gate outputs a high signal only when both inputs are high.) We chose instead the circuit shown here for practical reasons: It provides the same functionality as an AND gate, and it does not require adding another chip to the circuit.

The geometrical layout of the sensors has the detector mounted at the center-front of the robot and pointed straight ahead. The emitters are set one to each side and aimed slightly outward to the left and right. This saves having two detectors. Rug Warrior can get by with just one and yet still see to both left and right.

An obstacle-detection program can be written very easily in C using the sleep function, as the following code fragment shows. PD2 is asserted and a sleep period begins. After 600μs, PE4 is polled and its state is saved in the variable val_on. Then PD2 is deasserted and the program waits another 600μs. Next, we poll PE4 again and store its value in val_off. An obstacle is detected if the detector output is low when the emitter is on and high when the emitter is off. The function ir.detect() should be called as often as necessary to keep the variable ir.status updated. A similar loop is repeated for the other LED.

```c
int ir.status = 0; /* Global var for IR detection status */

void ir.detect()
{( int val_off, val_on; /* Intermediate vars for IR detection */
  bit_set(port.D, 0b000000100); /* Turn on one emitter */
  sleep(0.000600); /* Wait for 600μs */
  bit_clear(port.D, 0b000000100); /* Turn off one emitter */
  val_on = bit_get(port.E, 0); /* Read status */
  bit_set(port.D, 0b000000101); /* Turn on next emitter */
  sleep(0.000600); /* Wait for 600μs */
  val_off = bit_get(port.E, 0); /* Read status */
  bit_clear(port.D, 0b000000101); /* Turn off next emitter */
  ir.status = (val_on && !val_off) || (!val_on && val_off); /* Check for obstacle */
}
```

Figure 5.12. A Sharp G1U52X near-infrared proximity detector (Radio Shack 276-137) detects reflected power emitted from near-infrared LEDs, such as a Siemens SFH 484 LED.

Figure 5.13. The obstacle-detecting infrared beam has a 40 kilohertz (kHz) carrier modulated at 1667 Hertz (Hz). Note that the transmitted signal must be broadcast for several cycles before being acknowledged by the detector. Likewise, when transmission ceases, a few microseconds pass before the detector changes state. Both these delay times can depend on the signal strength.

signal should then be modulated at a lower frequency. The blinking should be on for 600μs and then off for 600μs. Figure 5.13 gives the timing diagram and protocol for this emitter-detector pair.

The 40 kHz oscillator portion of the infrared emitter circuit in Figure 5.12 is implemented using two inverters, a capacitor, a resistor, and a potentiometer. This 40 kHz oscillator runs constantly while Rug Warrior is on, but the LEDs blink only when pins PD2
val.on = peek(port_E); /* Get value of detector */
bit.clear(port_D,0b000000100); /* Turn emitter off */
sleep(0.0000600); /* Wait for 60u */
val.off = peek(port_E); /* Get value of detector */
if ((val.off & ~val.on & 0b000000100) == 0b000000100 )
    ir.status = 1; /* Obstacle detected */
else
    ir.status = 0; /* No obstacle detected */
}

Common fluorescent lights put out a great deal of noise, to which the IR detector is sensitive. Using the turn-on, test, turn-off, test strategy just outlined will help to eliminate spurious obstacle detections due to noise.

Hamamatsu makes some very convenient-to-use optical sensors, ranging from photocells and near-infrared emitters and detectors to position-sensitive devices, photodiode arrays, and triangulation-based near-infrared rangefinders. One very simple implementation of a near-infrared proximity detector uses the Hamamatsu S3599 light-modulation photo IC. This detector contains an on-chip oscillator to drive an accompanying LED and also an integrated correlating receiver. This means the entire system can be built in a very small package. (The discrete-component 40kHz oscillator of the previous example is extraneous here.) Figure 5.14 illustrates a sample circuit.

There is a trick you can play to squeeze a little more information out of an IR proximity sensor. The detector responds to the IR power it receives by activating if the incoming power is high enough and not activating if the power is too low. If the power output by the emitters can be varied then it is possible to determine whether a detected reflection comes from a nearer or more distant object. To estimate range, start by setting the output power at some high level, then check for a reflection. If a reflection is detected, reduce the power and check again. Continue in this way until no reflection is detected. The output power level at which the reflection becomes undetectable is related to the distance of the object.

The effective power seen by the detector can be varied in several ways. The brute force method is to build a digitally-controlled analog circuit where the output power is set by some number of input bits. A second method is to tune the oscillator frequency away from the nominal 40 kHz preferred by the detector. The more the frequency differs from 40 kHz, the shorter the range at which the detector will respond to obstacles. A third method is generally most convenient for a fully-digital implementation; simply change the duty factor of the oscillator. The duty factor of the oscillator period when the signal is high, is approximately 50%. (This is the duty factor of the circuit in Figure 5.12.) Building an oscillator circuit with a digitally-controllable duty factor allows estimation of the obstacle's range. This is the scheme used by some commercial research robots.

5.3 Light Sensors

5.3.3 Near-Infrared Range Sensor

The GP1U52X IR detector discussed in the previous section is a popular, inexpensive, and easy to use proximity sensor. However, a sophisticated new sensor, able to accurately determine the range to a nearby object, has recently been made available by Sharp. The GP2D02 consists of an IR emitter and position sensitive detector, PSD, in a single package (see Figure 5.15). Unlike IR proximity detectors, the GP2D02 computes an actual range to an object based on triangulation. This means that (also unlike proximity detectors)
the GP2D02 is relatively insensitive to the color and texture of the object at which it is pointed.

Figure 5.16 shows how the detector works. The emitter, the lower element in the rectangular package, illuminates a small spot on an obstacle with modulated IR light. A lens forms an image of the spot on the active element at the back of the detector. The output of the detector element is a function of the position on which the image falls. In Figure 5.16(b), the image forms at the center of the active element. When the device is farther from the obstacle as in a the image is closer to the bottom. And in Figure 5.16(c), with the device close to the obstacle, the image of the projected spot forms near the top of the active portion of the detector element.

As is suggested by the drawing, when the distance between the detector and the obstacle reaches some minimum, about three inches, the image misses the active portion of the detector element entirely. Thus, the GP2D02 cannot detect obstacles that are too close. Also, at some large distance, the reflected energy is too weak to activate the detector. The maximum distance at which the GP2D02 can detect an obstacle depends on the color and surface properties of the obstacle. From about three to about 15 inches, the output is almost linear with distance, and objects of almost any color can be detected.

Unfortunately, the interface of the GP2D02 is not nearly as friendly as that of the IR proximity detectors described earlier. The GP2D02 mates with a miniature 4-position connector that, outside of Japan, is very difficult to acquire. No matter, you can solder wires directly to the pins of the connector.

The next problem is getting data out of the device. The GP2D02 returns range as an eight bit value. Since there is only one output pin, you must actively clock an input pin to get the GP2D02 to output data. Start with the $V_{in}$ high then hold it low for 70 ms or longer. The detector makes its measurement during this period. Then send 8 pulses whose positive half lasts for 200 $\mu$s or less.
the positive going part of each pulse, read \( V_{\text{out}} \). The most significant bit of the range measurement comes first, the least significant last. Higher numbers in the result correspond to shorter ranges. Consult the manufacturer's literature for more details of the procedure.

There is one last confusing part to the interface. If you simply connect a digital output line from your microprocessor to the \( V_{\text{in}} \) input line of the GP2D02, it won't work! The GP2D02 uses a curious "open drain" input circuit. Your output line can pull this circuit low, but must never pull it high. That means you should connect a diode between your output line and \( V_{\text{in}} \), the anode of the diode goes to \( V_{\text{in}} \).

If you can put up with its idiosyncrasies, the GP2D02 makes an excellent short-distance range sensor. It even performs well in bright light.

### 5.3.4 Pyroelectric Sensors

One of the most useful sensors for endowing your robot with a means of interacting with humans is a pyroelectric sensor. A *pyroelectric sensor* is the essential component in certain types of motion-detecting burglar alarms. The output of a pyroelectric sensor changes when small changes in the temperature of the sensor occur over time. The active element in such a sensor is typically a lithium tantalate crystal. Charge is induced as the crystal is heated. Inexpensive pyroelectric sensors are optimized to detect radiation in the 8–10 \( \mu \text{m} \) range (the range of infrared energy emitted by humans) and require no cooling to produce a useful signal. This makes them suitable for use in motion sensors and security alarms.

Pyroelectric sensors are sold by a number of companies. Figure 5.17 depicts a dual-element sensor with integrated amplifier, the 4423, sold by Eltec. The package is shown in the can with the window at the left. To the right is a construction-paper cone for holding a plastic fresnel lens (made by Fresnel Technologies) at the focal distance from the window.

Other companies (Watlow, Mikron Instrument, Detection Systems, Microwatt Applications, Hunter Products, Linear, Spiricon, etc.) make pyroelectric sensors. Nippon Ceramic makes a low-cost version of the pyroelectric sensor shown in Figure 5.17 but without the integrated amplifier. Acroname, Inc. offers a pyro sensor complete with Fresnel lens designed for compatibility with Rug Warrior.

Figure 5.18 illustrates the interface between the MC68HC11 and a pyroelectric sensor. The Eltec 442-3 sensor shown incorporates two lithium tantalate crystals. The amplified difference of the voltage across the crystals is the output of the sensor. In the case that both crystals are at the same temperature, the sensor produces an output signal that remains steady at about 2.5 V (assuming a 5 V power supply). If a person walks in front of the sensor moving from left to right, the signal will rise above 2.5 V by about one volt and then fall below it, finally returning to the steady-state value. Should a person walk in front of the sensor moving from right to left, the reverse will happen. The signal will first fall, then rise, and then settle at 2.5 V. Figure 5.19 illustrates the time-varying output signal of the Eltec sensor.

By taking advantage of the MC68HC11's A/D port, we can implement the interface with a minimum of components. The same "flavor" software driver as used in the photocell routines can gather pyroelectric data. A program to notice when the readings go above or below a preset threshold can trigger some robot behavior. More sophisticated software could look for trends and try to determine...
5.3.5 Ultraviolet Sensors

On the opposite end of the spectrum from pyroelectric sensors, Hamamatsu offers a line of *ultraviolet sensors* called the UVtron series. These devices are sensitive to radiation in the 185 to 260 nanometer (nm) range but are very insensitive to light in the visible range. In most environments the only source of UV light is a flame. Hamamatsu UVtron sensors have been used with good results by contestants in the Robot Home Firefighting Contest (see Appendix F) held each Spring at Trinity College in Hartford, Connecticut. For information on the UVtron visit Hamamatsu’s website at: optics.org/hamamatsu/hps_home.html

5.3.6 Cameras

Video camera technology continues to become more compact and more inexpensive everyday. Small cameras from security systems are a good buy, as illustrated by the Chinon camera in Figure 5.20. Sony also sells small Watchcam cameras.

While onboard vision computations with a MC68HC11 probably are not feasible (especially given all the other sensors connected to Rug Warrior’s processor), transmitting to an onboard workstation can be viable. A cable may be used for this application, although a television transmitter is preferable. Some inexpensive and amazingly small (postage stamp sized!) video transmitters are now available. These transmitters operate on the experimental TV (ham radio) frequencies and require a license from the Federal Communications Commission. Contact Supercircuits for information.

5.4 Force Sensors

In general, *force sensors* have proven the most reliable, exhibit the lowest noise, and produce the most easily interpreted signal of all
sensors. Force sensors can be used to determine when the robot is in contact with another object and where that object is in relation to the robot. Such information allows the robot to maneuver away from collisions.

### 5.4.1 Microswitches

Microswitches, such as the two shown in Figure 5.5 (page 116), are small, momentary switches that can be attached to bumpers to signal when the robot has run into an obstacle. Such switches can be purchased from a number of suppliers, such as Gerber or Digi-Key.

Figure 5.21 illustrates one method for using microswitches to detect collisions between the robot and various obstacles. The switches are mounted in such a way that, when the robot contacts an object, one or two switches will close, thus revealing the relative positions of robot and object.

Figure 5.22 show two ways to interface the bump switches to the microprocessor. The circuit in Figure 5.22(a) is straightforward. One pin of port E is used for each switch. When the robot collides with an obstacle, one or two switches close, changing the state of the corresponding bit(s) from 0 to 1. This approach has the advantage of being easy to understand and implement, but it uses up three of the MC68HC11's input lines.

There is another way to achieve the same functionality that uses only one MC68HC11 input pin. This second approach is shown in Figure 5.22(b), where a network of resistors is used to create different voltages at the MC68HC11 input pin, depending on which switch is closed. (The A/D mode for port E must be used.) The bump switch software driver must read pin PE3, do a conversion, and then set one of eight flags. The correct flag signifies which switch or set of switches is closed; this is determined by in which of eight ranges the measured voltage falls.

A careful analysis will show that the circuit in Figure 5.22(b) is essentially a voltage adder. As long as the current flowing from the +5 V supply through the single 2.2 kilohm (KΩ) resistor and two 1.2K resistors to ground is large compared to the current flowing through any other part of the circuit, this approximation will hold. If, as shown, the powering voltage divider has taps at 1/4, 1/2, and 1 times the supply voltage, then the voltage sum will be 1/3 × (A + B + C) (where each of points A, B, and C is connected either to its corresponding tap or to ground). Since the A/D converter produces
5.4 Force Sensors

A factor of about 3 to 5 as the bend sensor goes from straight to maximum bend.

5.4.3 Force-Sensing Resistors

Interlink Electronics manufactures a line of force-sensing resistors that, like bend sensors, are based on conductive ink technology. The resistance of a force-sensing resistor can change by several orders of magnitude as force is applied. (This is a much greater change than the bend sensor exhibits.)

Force-sensing products come in a variety of shapes and sizes, from 0.2 inch diameter circles to strips 24 inches long. A linear potentiometer is also available, which can determine the position of a contact anywhere along its length.

5.4.4 Rubbery Ruler

An interesting, yet quite simple sensor, was recently developed at the University of Melbourne. Called the Rubbery Ruler, this sensor can accurately measure changes in its own length (see Figure 5.24). Such a device might be used in the bumper of a robot to detect deformations or displacement or could be used to determine
5.5 Sound Sensors

Sensors for sound in the audible range can allow the robot to interact with its operator. Ultrasonic transducers help the robot detect and avoid obstacles.

5.5.1 Microphones

A microphone can easily be interfaced to a microprocessor. Typical behaviors instigated on a robot are: moving toward or away from noise, listening for a specific pattern of sounds, and localizing a sound’s position within a room. The microphone shown in Figure 5.5, (page 116) came from Radio Shack, but Digi-Key also sells microphones, as do a number of surplus stores.

The signal from the microphone typically must be amplified before being read by the microprocessor. Figure 5.25 shows one approach, using an LM386 op-amp. Again, the output of the amplifier is connected to an A/D pin of port E and software driver routines similar to the previous examples can be used to read the data.

One significant problem with using a microphone is the need to sample the signal very frequently. Figure 5.26 illustrates the type of signal output from a microphone. If the robot is trying to detect a hand clap or a whistle, for example, it must sample the signal from the microphone often enough so that it does not miss the event (Instantaneously, the reading from the microphone is just a voltage between 0 V and 5 V.) The signal produced by a hand clap may last only a millisecond or so. This means that the microprocessor must check the output of the microphone at least that often. Thus, looking for very brief or high frequency signals can require all of the microprocessor’s time. It may be necessary to dedicate a microprocessor or other custom hardware solely to the task of monitoring the microphone.

Another important problem is that a microphone mounted on a robot is most likely to detect the sound made by the robot's own motors. It will usually be necessary to shield the microphone in some way to guard against this.

More sophisticated acoustic sensors are available that can digitize and record voices for later playback. Other systems do rudimentary (usually speaker-dependent) voice recognition. Still, these systems see continuous improvement and lower prices as time goes on. Speech-synthesis boards are also available from suppliers such as RC Systems. Writing data strings to various registers signals the device to output an assortment of phonemes. The programmer can then create a number of sentences to give the robot simple language facilities.

5.5.2 Piezoelectric Film Sensors

Piezoelectric film is a remarkably versatile and inexpensive sensor material. Properly configured, the same material can be used to detect vibrations, changes in applied force, changes in temperature,
and even far-infrared radiation. In each case, the sensing operation consists of measuring the voltage imposed on a pair of electrodes on opposite sides of a polyvinylidene fluoride film. Piezoelectric film sensors produce a voltage only when subjected to changes in the sensed quantity. For example, when used as a collision detector, the piezoelectric sensor will generate a voltage spike at the moment the robot bumps into an object but will produce no signal while the robot is pressed against the object. Piezoelectric film allows the robot builder to construct highly customized sensors. Piezoelectric film, evaluation kits, and sensor components are available from AMP (www.amp.com/sensors/sensors.html).

5.5.3 Sonar

While the most common near-infrared detectors deliver only proximity information (something is or is not there), a sonar transducer can actually provide distance information because it is possible to measure the time of flight between the initiation of a ping and the return of its echo. By measuring the time of flight and knowing the speed of sound in air, it is possible to calculate distance covered by the round-trip of the ping.

Figure 5.27 shows the Polaroid sonar rangefinding system, which is one of the most commonly used sensors on mobile robots. These rangefinders were developed as autofocus mechanisms for cameras, but the units can be purchased separately. The driver board has a very simple protocol for interfacing to a microprocessor. Figure 5.28 illustrates the necessary interface electronics.

To measure the distance to an object, the ranging board begins by sending a brief 400 volt signal to the transducer. This creates an ultrasonic chirp. After transmitting the chirp, the ranging board monitors the transducer for a returning echo. The board automatically increases its gain with time to better detect the fainter echoes returning from more distant objects. When an echo is detected, the ranging board asserts (sets to high) its output line. By measuring the time between initiation of the chirp and return of the echo, the robot's microprocessor can determine the distance to the object responsible for the echo.

In the example circuit we use two input capture registers to record the time when the sonar ping begins its flight and the time when the echo is detected. The input lines associated with the timers are designated PA1 and PA2. These lines are unassigned and thus available to the sonar module. However, in Rug Warrior's standard configuration, all of the MC68HC11's outputs are dedicated to built-
When the sonar pulse occurs, the ranging board draws 2 amperes of current for a fraction of a millisecond. Such a large current can be a challenge for the robot's power supply; you will get better results if you install a capacitor of about 500 μF from power to ground near the ranging board. It is also useful to wire a second capacitor of around 1 μF directly to the backside of the ranging module between the V+ and Gnd pins. The ranging board is a sensitive, high-gain device, and without this second capacitor, noise on the V+ line can cause the Echo line to go high as soon as the sonar pulse terminates.

Figure 5.28 shows the Bink and Binh signals connected to ground and no connection to the Osc line. Advanced features can be enabled by using these lines in different ways. Consult the manufacturer's documentation for more information. A technical manual and application notes are supplied with Polaroid's Developers Kit, Designers Kit, and OEM Kit.

A software driver of surprising simplicity can calculate the range, given the circuit in Figure 5.28. The following three functions are all that is needed.

```c
/* Enable input capture on rising edge on lines PA1 and PA2 */
void init_sonar()
{
    bit_set(tct12, 0b010100); /* Use bit_set and bit_clear rather */
    bit_clear(tct12, 0b100100); /* than poke to avoid changing */
    /* other tct12 bits */
}

/* Initiate a sonar ping */
void ping()
{
    poke(tflgl, 0b10); /* Writing a 1 bit clears echo received flag */
    bit_set(portg, 0b0001100); /* Turn on PD2 and PD3 => Start ping */
    sleep(0.030); /* Wait 30 milliseconds for an echo */
    bit_clear(portg, 0b0001100); /* Clear echo line */
}

/* Determine if an echo was received, if so compute the range */
float range()
{
    if ((peek(tflgl) & 0b10) == 0)
        return -1.0; /* IC2 didn’t capture echo */
    else
        return 0.000569; /* Echo detected, compute time and convert to feet */
    } /* <<< Time to the nearest millisecond (ms) */
```
It is possible to determine the time-of-flight using only one input capture line. To do this, the robot must store to a variable the time when it commanded the ping. After the echo is received this variable is subtracted from TIC1. The only problem with this approach is latency. Because the microprocessor can be interrupted at any instant, there is no top-level way to be sure that the sonar ping was sent immediately after the time was recorded.

As described above, the INIT and ECHO lines of the ranging board are wired to the MC68HC11's PA1 and PA2 inputs. PA1 and PA2 are associated with internal input capture systems IC2 and IC1, respectively, sic. The input capture system allows the current time to be captured (written to an associated register) the moment the signal on an input capture line goes from low to high. The time-stamp registers associated with IC1 and IC2 are TIC1 and TIC2, respectively. Time is measured in units of ticks; a tick is 0.5 microseconds long.

Whenever the signal on an input capture line does go high, another internal register (called TFLG1) records the fact by automatically setting a bit corresponding to the particular input capture line that went high. We can use this feature to determine if the sonar board received an echo.

The IC1 and IC2 timers must be initialized so that they behave as described. This is accomplished by the init_sonar() function. From the MC68HC11 documentation we know that the register TCTRL2 controls IC1 and IC2. Rising edge capture is enabled by writing binary %00010110 to TCTRL2. The command init_sonar() must be executed before sonar ranging is attempted.

All we need do to initiate a sonar chirp is turn PD2 and PD3 on together. This is accomplished by ping(). After starting the sonar pulse, ping() waits 30 ms, then turns off PD2 and PD3, turning off the Init line. We need not wait much longer than 30 ms, for an echo. Echoes taking longer than 30 ms to return are generally so faint that they will not be detected by the ranging board. Another reason for the 30 ms cutoff is that the registers that count and capture the time are only 16-bits long—after about 32 ms, the registers overflow and would give meaningless results.

Finally, range() is called after ping() to compute the time of flight and convert to units of length. First range() checks the bit in the TFLG1 register that tells whether IC2 has actually captured a time. If it has not, then no echo was received; range() returns -1.0 to indicate the absence of good range data. If an echo has been received range() subtracts the time captured by IC1, the ping initiation time, from the time captured by IC2, the time at which the echo returned. This difference is the time of flight in units of 1/2 microseconds. range() multiplies this number by a constant that converts to units of feet. The speed of sound at normal temperature and pressure is 1138 feet per second, one tick is 0.5 x 10^-6 seconds, so the ratio is: 1138 / 0.5 x 10^-6 = 0.000569 feet per tick. Multiplying this number by the difference computed above would give the total distance the echo travels. However, since the echo travels both out and back, the distance from robot to obstacle is actually half this. We thus divide 0.000569 by 2.0 to get 0.0002845. Multiplying ticks by this number gives the distance to the obstacle in feet.

A careful examination of the range() function in the listing above reveals that this is not exactly what is done. As is often happens in robotics, there is an additional complication. The IC programming language stores signed integers using only 16 bits. This means that integers are restricted to the range -32768 to 32767. Integers larger than 32767 are interpreted as negative numbers. Subtracting the contents of TIC1 from that of TIC2 may result in a number larger than 32767. This is the reason for the shift right bit operation (>>1) in the range() function. The number produced by this shift (effectively dividing by 2) cannot be larger than 32767. This solves the sign problem but we must correct for dividing by 2 by multiplying the final constant by 2.0, thus 0.0002845 becomes 0.000569 again.

One caveat should be mentioned. A single transducer is used here to both transmit and receive the sonar ping. After transmitting a ping, the transducer continues to oscillate for a brief time. While these oscillations decay, the ranging board must not attempt to detect an echo because it has no way to distinguish a legitimate echo from residual ringing of the transducer. The sonar unit automatically handles this by blanking detection of an echo until 2.38 ms after the ping begins. The effect of this is that in normal operation the sonar unit cannot detect objects closer than about 16 inches.
Sonar ranging is useful for obstacle detection, corridor following, localization, and map building. However, sophisticated the final behavior, this underlying primitive operation of calculating the range of a ping is the same in all cases.

## 5.6 Position and Orientation

For a robot to find its way about in the world, it often needs to make certain measurements. For example, it may be helpful for the robot to know the direction of gravity, the local compass heading, or how far it has moved or turned since it was in some known position. In this section, we review sensors that can provide such information.

### 5.6.1 Shaft Encoders

A **shaft encoder** is a sensor that measures the position or rotation rate of a shaft. Typically, a shaft encoder is mounted on the output shaft of a drive motor or on an axle. The signal delivered by this sensor can be either a code that corresponds to a particular orientation of the shaft (such shaft encoders are called *absolute encoders*) or it may be a pulse train. Shaft encoders that produce a pulse train are called *incremental encoders*. Each time the shaft turns by a small amount, the state of its output changes from high to low or vice versa. Thus, the rate at which pulses are produced corresponds to the rate at which the shaft turns.

A potentiometer can be used as an absolute position encoder. Each position of the shaft produces a unique resistance. Absolute encoders are commonly used for determining the positions of robot arms.

One way for the robot to get feedback on how far its wheels have turned or on synchronizing two wheels' velocity is to connect an encoder to each motor shaft. Shaft encoders can be purchased as enclosed units or built in as an integral part of a motor. Some incremental shaft encoders contain a spinning disk that has slots cut in it. The disk attaches to the motor shaft and spins with it. A near-infrared LED is placed on one side of the disk’s slots and a phototransistor on the other. As the disk spins, the light passing through the disk is interrupted by the moving slots, and a signal in the form of a pulse train is produced at the output of the phototransistor. By using a microprocessor to count these pulses, the robot can tell how far its wheels have rotated. The combination of such an infrared LED emitter and a photodetector, packaged for the purpose of being mounted on either side of a shaft encoder’s disk, is called a *photointerrupter*.

Another implementation of a shaft encoder is a *photoreflector*, which shines light from an infrared LED onto a striped wheel, which then reflects the light back to a phototransistor. A palette of radially alternating black and white stripes will alternately reflect or not reflect light to the phototransistor, yielding a similar pulse-train output. The photoreflector used by Rug Warrior is packaged with the two devices next to each other in a very compact unit. Figure 5.29 illustrates one of these small devices, attached to the side of a servo motor in such a way as to be within a few millimeters of...
and facing the striped pattern on the wheel. A closer view of the mounting scheme is shown in Figure 5.30.

Because the near-infrared energy emitted by the LED can penetrate thin, white paper, it is important to take into consideration what is behind the striped paper pattern. Two pieces of plain, white paper discs backing the striped wheel should be enough to make the white segments adequately opaque so that the beam will be reflected back to the detector. Figure 5.31 illustrates 32-, 48-, and 64-count encoder patterns. You can photocopy these patterns and use them to construct your own reflective shaft encoders.

The photoreflectors we have chosen for Rug Warrior are the Hamamatsu P5587s (these devices replace an earlier P306201 part). We have chosen these devices because they have circuitry integrated in the package to amplify and condition the output of the phototransistor. The only interface components required for connecting to the MC68HC11A0 are two resistors: one for pulling up the phototransistor’s open-collector output and one for limiting the current through the LED. For reading the shaft-encoder data into Rug Warrior’s control system, we have chosen to take advantage of the timer-counter hardware connected to the MC68HC11A0’s port A. Port A’s 8 pins have various input capture and output compare registers associated with them, which are able either to mark the time that events happen on those pins or to initiate events at preprogrammed times. We use PA7 and PA0 as the port A pins to accept the input from the left and right shaft encoders, respectively, as shown in Figure 5.32.

A pulse accumulator function is associated with PA7 making it easy to count the pulses produced by the left shaft encoder in software. It would have been convenient if the MC68HC11A0 designers had included two of these features on their chip (more advanced versions of the MC68HC11 do have more features for reading shaft encoders and for pulse width modulating motors), but since we do not have that luxury, we connect the right shaft encoder to the PA0 pin and use its input capture function to count the pulses.

Figure 5.33 illustrates a simple open-loop control scheme, where a motor is given a speed command and the shaft encoders are used simply to monitor its velocity. Later, in Chapter 7, we will use other portions of port A’s timer system, output pins PA5 and PA6, to drive the motors, and we will also discuss how to use shaft-encoder feedback data to implement in software a velocity controller. In this section, however, we concentrate on describing how to get the shaft-encoder sensor data into the microprocessor.
5.6 Position and Orientation

Reading Shaft Encoders

In order to use the shaft-encoder sensors in some sort of velocity control scheme for Rug Warrior, we must first interface the photoreflectors to the microprocessor and store the ensuing counts for each wheel in two variables. One shaft encoder is fed into the pulse accumulator on port A pin PA7, and the other shaft encoder is fed into PA0 with its associated input capture three (IC3) register. Reference should be made to the Motorola MC68HC11 data books for more complete descriptions of the timer-counter system than those undertaken here.

The Pulse Accumulator

The pulse accumulator is an 8-bit counter register, PACNT, associated with port A pin PA7, that makes it very easy to count the number of rising or falling edges input to that pin. This register will overflow after $2^8$ or 256, counts:

\[
\begin{array}{cccccccc}
\text{PACNT} & \text{Bit 7} & \cdots & \cdots & \cdots & \cdots & \cdots & \text{Bit 0} \\
\hline
\end{array}
\]

In order to configure the system for our needs, we first have to assign pin PA7 as an input. This can be done by setting the data-direction bit for pin PA7 (which is in the pulse-accumulator control register, PACTL) to 0 for configuration as input pin. Three other bits in the PACTL register must also be assigned. The pulse-accumulator enable bit, PAEN, must be set to 1 to enable the pulse accumulator; the mode-select bit, PAMOD, must be set to 0 for event counting; and the edge-select bit, PEDGE, must be set to 1 or 0, depending on whether it is desired to choose rising (PEDGE = 1) or falling (PEDGE = 0) edges of the shaft encoder's output. We will arbitrarily select to count rising edges and so set the PEDGE bit to 1:

\[
\begin{array}{cccccccc}
\text{PACTL} & \text{Bit 7} & \cdots & \cdots & \cdots & \cdots & \cdots & \text{Bit 0} \\
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\hline
\text{DRA7} & \text{PAEN} & \text{PAMOD} & \text{PEDGE} & 0 & 0 & \text{RTRI} & \text{RTR0} \\
\hline
0 & 1 & 0 & 1 & x & x & x & x \\
\end{array}
\]

Once the PACTL register has been configured, the pulse accumulator will start counting the number of stripes passing in front of
the photoreflector. The main program running on Rug Warrior then simply needs to poll the PACNT register at certain intervals to see how fast the wheel is turning.

**Shaft-Encoder Pulse-Accumulator Software Driver**

Following is C code that initializes the pulse-accumulator system and returns the number of pulses since the last reading. To activate the pulse-counter system, call `init_velocity()` during system initialization. Velocity of the left wheel can be found by calling `get_left_vel()` at regular intervals. Velocity is in units of encoder clicks per time interval (where the time interval is the time between two successive calls to `get_left_vel()`).

```c
int PACTL = 0x1026; /* Pulse accumulator control, 8-bit reg */
int PACNT = 0x1027; /* Pulse accumulator counter, 8-bit reg */

void init_velocity() /* Initialize hardware for vel monitoring */
{
  poke(PACTL, 0b01010000); /* PA7 input, enable pulse acc, */
  poke(PACNT, 0); /* Start with 0 measured velocity */
  /* rising edge */

float get_left_vel() /* Left vel from PA7 using pulse counter */
{
  float vel;
  vel = (float) peek(PACNT);
  poke(PACNT, 0); /* Reset for next call */
  return(vel);
}
```

Once the pulse accumulator hardware has been initialized, it will run in the background, automatically incrementing the count every time a stripe on the encoder wheel moves past the photoreflector. The robot's main program does not have to keep track of this activity but is free to attend to other sensors and actuators. When it needs to know the encoder count, the main program calls the function `get_left_vel()`.

Although an assembly language routine to start the pulse accumulator would also be very simple, we use an example of C code here for a particular reason: Namely, later, in Chapter 7, we will describe how to use shaft-encoder data as the feedback in a velocity controller for Rug Warrior's two motors. As that algorithm will require some multiplication and a fair amount of bookkeeping, it is easier to describe control algorithms by sticking solely to C.

**Input Capture Registers**

For the encoder wheel connected to port A pin PA0, more software complexity is in store. Because the MC68HC11A0 has only one pulse accumulator, we must use an interrupt to count encoder clicks from the right wheel. We will use the IC3 register associated with PA0 to generate an interrupt on every rising edge. The interrupt-handler routine, which automatically runs whenever a rising edge is detected, must increment a counter, clear the interrupt flag, and return from the interrupt.

To configure IC3 for this operation, a few associated registers must be initialized in a way similar to setting up the pulse accumulator. In this case, we will be generating interrupts and writing an assembly language interrupt-handler routine that keeps track of the count.

The TMSK1 register contains the bits that must be set to enable interrupts associated with events on any input capture pin. We will set the bit associated with IC3I, enabling interrupts:

```
TMSK1 Bit 7 Bit 0
$1022 OC1I | OC2I | OC3I | OC4I | OC5I | IC1I | IC2I | IC3I
x | x | x | x | x | x | x | 1
```

The TFLG1 register contains a flag bit, IC3F, which is set whenever the interrupt condition is met. If IC3F is set while global interrupts are enabled (the I bit of the condition code register is clear), then the hardware will automatically initiate an interrupt—the user's interrupt-service routine is called. Code in the interrupt-service routine must clear the IC3F flag; otherwise, when an attempt is made to return from the interrupt, the hardware will think the IC3 interrupt is pending and immediately service it again. Clearing the interrupt flag is accomplished by writing 1 to the bit in the TFLG1 register that corresponds to that interrupt's flag. We will write the binary number %00000001 to TFLG1 to clear the IC3F flag.

```
TFLG1 Bit 7 Bit 0
$1023 OC1F | OC2F | OC3F | OC4F | OC5F | IC1F | IC2F | IC3F
x | x | x | x | x | x | x | 1
```
Figure 5.34. The four actions possible by any input capture pin are to never capture, to capture on rising edges, to capture on falling edges, or to capture on any edges. Two bits in the TCTL2 register (the most significant bit, EDGXB, and the least significant bit, EDGXA) set the desired response for any successful input-event detection.

Another matter to take care of is assigning on which type of edge the input capture interrupt will trigger. Figure 5.34 gives the possibilities and the associated 2-bit code for assigning the desired trigger. We will trigger on rising edges, since that was the arbitrary choice made for the encoder connected to PA7.

These bits must be written to the TCTL2 register to configure it for rising edge-triggered interrupts. Storing %00000001 to TCTL2 will assign this properly:

```
TCTL2 Bit 7 Bit 0
$1021 0 0 EDG1B EDG1A EDG2B EDG2A EDG3B EDG3A
```

After these interrupts are configured, the main program loop must enable interrupts globally with the CLI instruction. Until this instruction is executed no interrupts can occur. Once this is done, any rising edge arriving on pin PA0 will trigger an interrupt. The vector address for the IC3 interrupt is $FFE8. The two byte address stored at this location is the address at which the user's interrupt handler code must begin.

5.6 Position and Orientation

Shaft-Encoder Input Capture Software Driver

The Interactive C compiler used on Rug Warrior, IC, has a means of interfacing to MC68HC11A0 assembly language routines. (It does this by following certain naming conventions for routines and variables and by using certain file-loading protocols.) We use these features here to write an interrupt-handler routine for input capture register IC3, which counts the shaft-encoder pulses and stores the running sum in right_clicks, a global variable accessible by the main C program.

```
TFLG1 EQU $1023
ORG MAIN_START

;Timer Flag 1, 8-bit reg
;Orgin for assembly module

subroutine.initialize_module: ;This module runs on reset
  ldd #IC3.interrupt_handler
  ;16-bit addr of interrupt handler
  std $FFE8
  ;Store in IC3 interrupt vector
  cli
  ;Enable interrupts generally
  rts
  ;Return from subroutine

variable.right_clicks: ;Create a C variable, right_clicks
  fdb 0
  ;Fill double byte, 16 bits, right_clicks = 0

IC3.interrupt_handler:
  ldd variable.right_clicks
  add #1
  ;Add one more encoder count
  std variable.right_clicks
  ldaa #00000001
  ;Clear the IC3 flag by writing a one
  staaa TFLG1
  ;Store in TFLG1 to clear IC3 flag
  rti
  ;Return from interrupt
```

These code fragments accomplish several goals. A code initializer module, subroutine.initialize_module, is created, whose purpose is to store the address of the interrupt handler in the correct location. The IC system calls subroutine.initialize_module each time the reset button is pushed. A variable, variable.right_clicks, for storing the encoder counts from the right shaft encoder is also created. (C routines will reference this variable using the variable name right_clicks.) Finally, IC3.interrupt_handler, an interrupt-handler, is written, which increments the right-encoder counts variable each time the reflective photosensor sees the stripe it is looking at change from black to white.

If we compare this example with the code for the other shaft encoder connected to PA7, the contrast is clear. The pulse accumulator provided us with special purpose hardware to relieve the main program of the duty of incrementing a counter every time an event occurred. Here, the programmer must specifically set up an interrupt-handler routine to attend to this chore.
5. Sensors

Now we add a function, `get_right_vel()`, to our existing C code, which returns the value of `right_clicks` (then resets it) whenever it is called. Our supervising C program must now also include the commands to initialize the appropriate registers for using the IC3 input capture interrupt.

For instance, our C program might look like the following:

```c
int TCTL2 = 0x1021;  // Timer Control 2, 8-bit reg, interrupt edge */
int TMSK1 = 0x1022;  // Timer Interrupt Mask, 8-bit reg */
int TPLG1 = 0x1023;  // Timer Flags, 8-bit reg */
int PACTL = 0x1026;  // Pulse accumulator control, 8-bit reg */
int PACTNT = 0x1027; // Pulse accumulator counter, 8-bit reg */

void init_velocity()
{
    poke(PACTL, 0b010100000);  // Set TA7 in, ena pls acc, rising edge */
    poke(PACTNT, 0);          // Start with 0 measured velocity */
    bit_clear(TCTL2, 0b0000000010); // IC3 interrupts on rising edges */
    bit_set(TCTL2, 0b000000010);  // IC3 interrupts on rising edges */
    bit_set(TMSK1, 0b000000001);  // Enable only IC3 interrupts */
}

float get_left_vel()
{
    float vel;
    vel = (float) peek(PACTNT);
    poke(PACTNT, 0);  // Reset for next call */
    return(vel);}

float get_right_vel()
{
    float vel;
    vel = (float) right_clicks;
    right_clicks = 0;  // Reset for next call */
    return(vel);}
```

The functions `get_left_vel()` and `get_right_vel()` provide a uniform way to acquire each motor's shaft encoder data. This is the essence of an abstraction barrier. Even though the hardware interface to each shaft encoder is implemented differently, the programmer simply relies on the function `get_left_vel()` and the function `get_right_vel()`. The programmer need not worry about how these functions interface to the hardware.

Later, we will use these primitive operators to create a higher-level program, a velocity controller, which will cause the two motors to always go at the same speed, enabling the robot to maintain a constant heading.

5.6 Position and Orientation

Figure 5.35. Futaba makes a small, rate gyro for model airplanes. The input is a pulse-width-modulated signal, and the output is an increased or decreased pulse width, depending on the rate of rotation.

5.6.2 Gyros

Another sensor that is useful in monitoring how the robot moves is a rate gyroscope. Mechanical gyroscopes use the principle of conservation of angular momentum to keep one or more internal axes pointed in the same direction as the exterior of the gyroscope, the gyroscope case, translates and rotates. Thus, a gyroscope attached to a robot makes it possible to determine either how rapidly the robot is rotating or how far it has rotated, relative to a fixed coordinated system.

Humphrey, Columbia, and Murata sell small gyroscopes, as does Futaba. The inexpensive model from Futaba, shown in Figure 5.35, is a single-axis rate gyro made for model helicopters. A rate gyro produces a signal proportional to the rate of rotation about an axis perpendicular to the axis of the gyro, but it does not provide absolute orientation information. The Futaba gyro takes a pulse-width-modulated signal provided by the MC68HC11 and modifies it (increasing or decreasing the pulse width) based on the rate of rotation of the gyroscope case.
5.6 Position and Orientation

5.6.3 Tilt Sensors

Determining whether your robot is level or tilted can mean the difference between negotiating rough terrain smoothly or tumbling over. Many types of sensors can provide information about the relative angle between the robot body and the gravity vector. The simplest and generally least expensive tilt sensor is the mercury switch, such as the one illustrated in Figure 5.36(a). This sensor consists of a small, glass bulb containing two or more contacts and a drop of mercury. Depending on which way the bulb is tilted, the bead of mercury will close or open the circuit.

Such a sensor is easy to interface in a microcontroller. When mounted properly, it provides a digital signal, alerting the microprocessor that the robot has tilted too far in one direction. Several mercury switches fixed at different orientations can provide information about the degree and direction of tilt. Software conditioning of the signal from a mercury switch is almost always required, however. As the robot starts, stops, and bounces about, the bead of mercury frequently makes contact, even when the robot is not dangerously tilted.

The electrolytic-tilt sensor, a type of inclinometer, offers an improvement over the mercury switch in many applications. Figure 5.36(b) diagrams an inclinometer. This sensor has two or more electrodes immersed in a conductive fluid. The conduction between the electrodes is a function of the orientation of the sensor relative to gravity. The electrolytic-tilt sensor produces an analog signal proportional to the degree of tilt. Such sensors are typically much more expensive than mercury switches. Spectron offers a full line of electrolytic-tilt sensors.

An exciting recent development in sensor technology is the micromachined accelerometer. This device is a chip with a tiny suspended mass machined into the silicon. Piezoresistors embedded in the structure are used to sense minute changes in position of the mass as the chip undergoes acceleration. Such devices can also be used to detect the direction of gravity. Micromachined accelerometers offer an accurate, rugged, and reliable means for determining the direction of tilt of a mobile robot. IC Sensors and Lucas Novasensor are good sources for these sensors.

5.6.4 Compasses

A compass provides a way for your robot to acquire absolute information about its orientation. This can be very helpful when writing a navigation algorithm. In open areas, compasses are very reliable, and once calibrated to local magnetic north, they are also accurate. If your robot is to be used indoors, however, the serviceability of a compass becomes more problematic. Magnetic fields from electrical wiring, structural steel in buildings, and even the metal components of the robot itself can all produce large errors in the compass reading. As long as errors of, say, ±45 degrees can be tolerated, the compass is a viable option. Certain electronic compasses intended for use in automobiles can, with sufficient modification, be employed by your robot. ETAK manufactures digital compasses. Precision Navigation, Inc., whose compasses are distributed by Jameco, has several useful models. Some models from Precision Navigation contain automatic tilt compensation. The Fetch robot described in Section 11.3.2 uses such a compass.
5.7 Proprioceptive Sensors

A proprioceptive sensor is any sensor used to measure the internal state of the robot. Monitoring these sorts of sensors can tell the robot when it is time to recharge its batteries, when a motor is overheating, or when a component has malfunctioned.

5.7.1 Battery-Level Sensing

By sensing its battery voltage, a robot can determine when it is time to return to the charging station or curtail power-draining operations. Designing a battery-level indicator is a simple matter when the microprocessor operates from a regulated supply, as in Figure 5.37(a). As shown, only a voltage divider is needed.

In the circuit of Figure 5.37(a), when $V_{RH}$ has been connected to the regulated output voltage from an LM7805, $V_{RL}$ will go to ground. We wish to determine $V_B$, the battery voltage. The voltage supplied by the batteries must always be higher than the regulated voltage in order to achieve good regulation. In this case, suppose that the batteries are effectively exhausted when their voltage reaches 7.0 V.

If we simply connected one of the A/D channels, say, PE0, to the positive battery terminal, it would not be possible to determine the battery voltage. Since the voltage at PE0 would always be greater than that at $V_{RL}$, the A/D converter would always report a value of 255 to the ADR1 result register.

We must engineer a circuit that will deliver a maximum of 5.0 V to PE0 when the batteries are fully charged and a smaller voltage as the batteries discharge. This is the purpose of the voltage divider. We will choose resistors $R_1$ and $R_2$, such that the voltage at PE0 begins at 5.0 V and decreases as the batteries discharge. Suppose that, when fresh, the batteries supply a maximum voltage of $V_{B,\text{max}}$. With the voltage divider connected as shown in Figure 5.37(a), the maximum voltage that can be present at PE0, $V_0 = \frac{R_1}{R_1+R_2}V_{B,\text{max}}$. To compute $R_1$ and $R_2$ we choose $V_0$ to be 5.0 V, since higher voltages cannot be measured. Given that we also know $V_{B,\text{max}}$, we can now solve for $R_1$ and $R_2$ if we arbitrarily choose the sum $R_1 + R_2$. This sum should be high enough so that the drain on the battery due to the voltage divider is insignificant compared to that of the rest of the electronics; at the same time, the sum should be small compared to the internal impedance of the A/D converter.

To complete the example, assume that $R_1 + R_2 = 4700\Omega$ and that power is supplied by eight NiCd cells whose fresh voltage is 9.6 V. Now we have $R_1 = \frac{9.6}{2.5} \times 4700 = 2447\Omega$, $R_2 = 2252\Omega$. By measuring the voltage at PE0, we can determine $V_B$: $V_B = \frac{2447}{2252} \times V_0$.

There is a complication if the microprocessor supply does not include a regulator, as in Figure 5.37(b) and (c). As we have seen, the A/D converter works by comparing the voltage at PE0 with the reference voltages at $V_{RH}$ and $V_{RL}$. If connected as shown in (b), the ratio of these voltages remains constant as battery voltage declines. Thus, the A/D converter always reports that the battery voltage equals $V_{RH}$, and the result of the conversion is always 255.

In Figure 5.37(c), we make use of the diode voltage drop to produce a reference to which we can compare the battery voltage. Whenever current through a diode exceeds a certain minimum, a characteristic voltage (usually about 0.6 V) develops across the diode. In the circuit in Figure 5.37(c), the A/D converter will compare the constant $3 \times 0.6 = 1.8$ V at pin PE0 with the changing voltage at $V_{RH}$. If the battery pack is fully charged, at say, 7.0 V
and depleted at 4.5 V, then the result from the A/D converter will be $255 \times \frac{1}{8} = 66$ and $255 \times \frac{15}{45} = 102$, respectively.

### 5.7.2 Stall Current Sensing

One reliable way to determine if a robot is stuck is to monitor the current being used to drive the motors. If all other sensors fail to detect an imminent collision, the robot will, in short order, come to rest against the obstacle. In this situation, the wheels will stop rotating while current to the motors will go to a maximum. Thus, motor current serves as a collision detector of last resort. One way to detect motor current is to put a small resistance in series with the motor (typically, a fraction of an ohm), amplify the voltage across the resistor, and measure the voltage with one of the A/D channels. Some motor-driver chips have built-in circuitry to simplify this measurement. The L293E and IR2100 motor-driver chips have such features.

The software that monitors motor current in order to detect a collision should not respond too quickly. Each time the robot accelerates from a dead stop, motor current will typically go to a maximum, then decrease as the robot speeds up.

### 5.7.3 Temperature

It is often a good idea to monitor certain temperatures within the robot. If the electronics get too warm, the microprocessor may crash. High temperatures can also shorten the lives of motors, and NiCd batteries may be damaged by heat if high current charging continues after the batteries are already fully charged. Certain motor-driver chips, the IR2100 for example, have built-in, over-temperature sensors. For other applications, many companies manufacture discrete temperature sensors including Murata, EDO Corporation, and RCD Components.

### 5.8 Exercise

To this point, we have seen how to take a large number of simple sensors and interface them to a microprocessor. In Chapter 9, we will see how to arrange higher-level programs, using behavior control, to enable the robot to act in response to its sensor readings to create seemingly intelligent behaviors. As we have seen, sensors merely deliver voltages to the microprocessor. What the robot manages to achieve with these signals depends upon how clever the programmer can be with software.

Many times, however, the programmer just does not have enough variables in her or his environment to juggle. The problem often dictates going back to hardware and inventing a new sensor for the job. For instance, in Figure 5.38, Rug Warrior is about to tumble off the edge of a step. All its sensors point upward and all its code implicitly assumes that it will always travel on level surfaces. Try to invent a sensor that will detect a step. Mount it on your Rug Warrior’s chassis, and interface it using connectors we discussed earlier in some spare prototyping space you left open on your board for expansion features. Try programming a software driver, and see how it works!
5.9 References

Whole volumes have been written about sensors for mobile robots, but here we have had the opportunity to touch only briefly on a few simple sensors that can be incorporated inexpensively into Rug Warrior. For the definitive reference on sensors for mobile robots consult (Everett 95). This book gives in depth coverage of a wide variety of sensors.

More sophisticated robots, such as Robart II, from the Naval Ocean Systems Center shown in Figure 5.1 (Everett, Gilbreath, and Tran 1990), and Attila from the MIT Mobile Robot Lab, shown in Figure 5.2 (Angle and Brooks 1990) take advantage of redundant sensors to endow themselves with increased awareness of their surroundings.

Robart II predated and influenced much of the hardware design later undertaken at the Mobile Robot Lab, especially in the realm of sensors. Everett and Stitz (1992) gives a complete exposition on the workings and wonders of a wide variety of sensors applicable to mobile robots.

Angle (1991) describes how the six-legged Attila was designed to use its legs as sensors as the robot moved through its environment, and how various sensors of increasing reliability were situated to trigger the lowest-level behaviors in a layered control system. Perrell (1992) expands on that theme and discusses the notion of creating virtual sensors from combinations of concrete physical sensors to make Attila more reliable.

For books on sensors and interfacing electronics, Beckwith and Marangoni (1990) detail making mechanical measurements from position sensors, force sensors, accelerometers, and the like, while Jung (1986) presents a “cookbook” of useful op-amp designs for amplifying and conditioning small sensor signals. Seipple (1983) is another useful sensor text.

6.1 Locomotion

From slithering to hopping, there are a great variety of ways to move across a solid surface. Among robots, the three most common systems use wheels, tracks, and legs.

Wheeled vehicles are by far the most popular for several practical reasons. Wheeled robots are mechanically simple and easy to construct. The payload weight-to-mechanism ratio is also favorable. Both legged and tracked systems generally require more complex and heavier hardware than wheeled systems designed for carrying the same payload. Additionally, a wide variety of wheeled devices, such as toys, can be modified for robot use.

The principal disadvantage of wheels is that, on uneven terrain, they may perform poorly. As a rule, a wheeled vehicle has trouble if the height of the object it must surmount approaches the radius of the wheels. One solution is simply to use wheels that are large compared to all likely obstructions. In many instances, however, this is impractical.

For robots that must operate in a natural environment, tracks are an appealing option because tank treads allow the robot to nego-