3
CHAPTER

Robot Anatomy

To help you understand how the *TAB Electronics Build Your Own Robot Kit* works, I will go through the robot's anatomy as if it were some kind of living being. Just as you can be described as being bilaterally symmetrical (both sides of you look the same), with paired primary sensors (eyes, ears, and hands), end effectors (hands), a method of locomotion (feet), and a method of fuelling (food), the robot's anatomy can be described using many of the same terms and concepts.

This may seem strange, but it reflects one of the most basic characteristics of any robot; robots are usually designed from models of animals and insects. While the prototypical (and most famous) robots are based on the human form, most robots are based on other living organisms. The state of the art is a long way from being able to reproduce the capabilities of the humanlike "Robbie" (from *Forbidden Planet*)—it is closer to being able to copy just the movements and abilities of an ant or other simple insect.

Soon after the basic design of the *TAB Electronics Build Your Own Robot Kit* was complete, I was asked to describe the robot. The description that I came up with (including its anatomical analogs in italics) was:

Body Type. Differential drive robot, 5" long and 4" wide

Fuel source. Powered by a single 9-volt alkaline (or rechargable) radio battery
- 20 minutes to 30 minutes of life per battery

Locomotion. Full H-bridge for both motors running with 19-kHz PWM frequency
- With fully charged battery, runs a bit faster than walking speed at 100% duty cycle on the PWM
- 4 PWM levels (25, 50, 75, and 100%) along with full stop
- H-bridge limits current draw in stalled motor conditions

Control. I/R TV remote control that provides the following commands:
- Stop/Forward/Reverse/Turn Left/Turn Right
- Random movement behavior (with collision detection)

- "Photovore" behavior (with collision detection)
- "Photophobe" behavior (with collision detection)
- Wall-hugging/maze-solving behavior (keeping a set distance from a wall on the right side)
- PWM speed select (off + 4 levels)
- Button controls to BS2

Sensors. Noncontact collision detection (somewhat of an oxymoron) using I/R LEDs and I/R TV remote control receivers/detectors
- Detection distance set by Pot on the Robot PCB
- Collision detection can be disabled

Sensors. CDS cells for light level detection

Control. Robot peripherals controlled by a PIC16C505

Control. BS2 socket with "AppMod" socket and BS2 programming connector
- BS2 socket driven by 9-volt battery directly
- AppMod socket driven by 9-volt battery directly
- Two-wire interface between the robot's PICmicro MCU and the BS2 for control. This interface is designed to use standard BS2 "SHIFTIN" and "SHIFTOUT" commands.

Along with this list of features built into the robot, the design of the electronic circuits built into the robot (shown in Figures 3-1 and 3-2) help to explain how the robot is designed and works.

This description and circuit diagram will give an engineer familiar with robots a reasonably good understanding of how the robot is designed. To a novice, it may seem somewhat cryptic and difficult to understand. To help you understand how this robot works, I will discuss in this chapter how the different features of the robot listed above work together as a complete package.

The organic model works quite well for describing the different features and characteristics of this robot. However, there is one major capability of organic beings that has not yet been replicated in robots—the ability to spontaneously reproduce. One of the ultimate dreams of roboticists (along with true "artificial intelligence") is to build robots with the ability to create copies of themselves without the need for factories.

Robot Type

Just as there are numerous different types of animals—walking, crawling, slithering, and swimming about the earth—there are quite a few different ways of laying out robots. When I describe robots in this chapter, I will be dealing primarily with robots that move. I will not discuss robotic arms (like those used to assemble cars) except to discuss issues they have in common with moving robot systems.

As mentioned earlier, chances are that when you hear the term "robot," you will think of a robot in human form such as the "False Maria" robot of *Metropolis* (see Figure 3-3). Unfortunately, the human form is probably the most difficult type of robot to create because it is so complex.

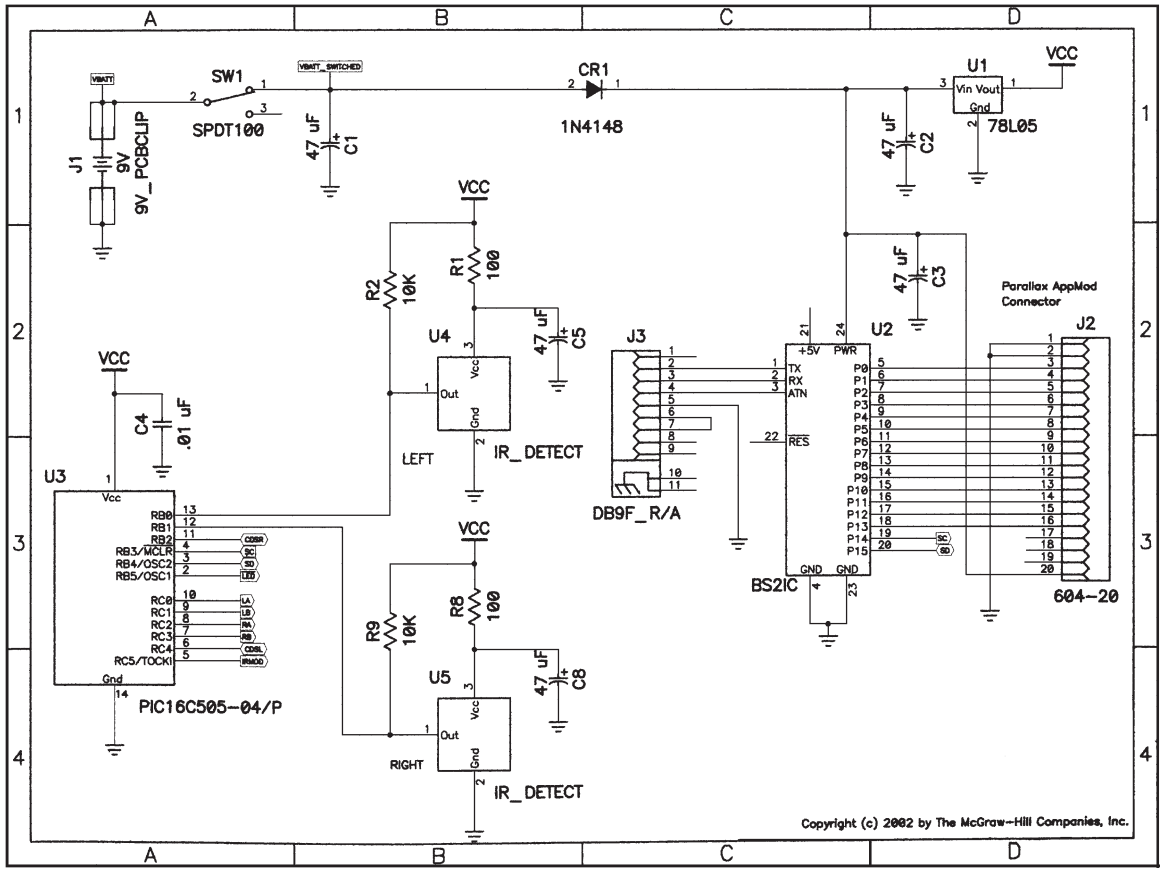

Figure 3-1 Design of the electronic circuits built into the robot (page 1).

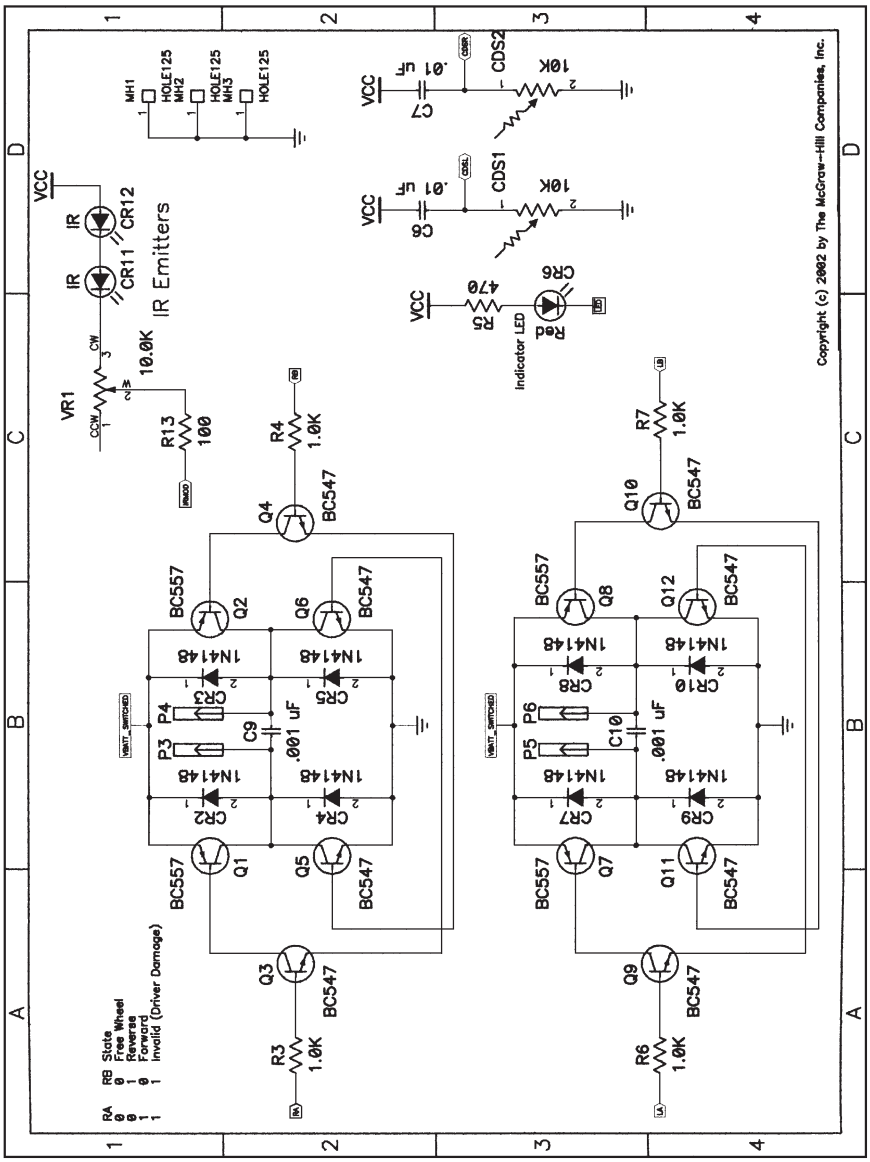

Figure 3-2 Design of the electronic circuits built into the robot (page 2).

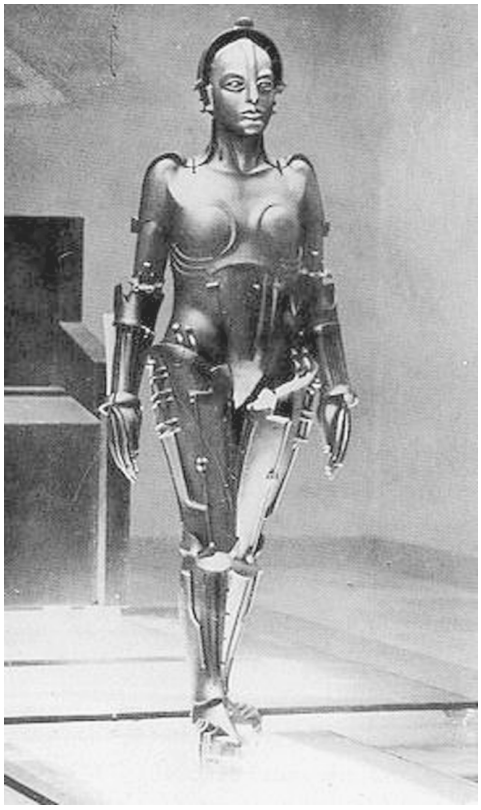

Figure 3-3 The "False Maria" robot of *Metropolis*.

Just standing requires a system of muscles ("actuators") in addition to position feedback and angle "sensors." A complex "control" system interprets the information coming from the sensors to command the actuators to keep the body upright.

An obvious solution to the problem of keeping the robot stable and not expending power to have it stand is to use wheels instead of legs. Legs offer the advantage of being able to handle rough surfaces much more ably than wheels, but this ability comes at a significant cost in complexity and power requirements.

There are many different ways in which to implement a wheeled robot. The most famous wheeled robot would have to be the "Sojourner Explorer" robot that explored the surface of Mars (shown in Figure 3-4). This robot is designed to run reliably over the uneven surface of Mars.

There are three basic types of wheeled robots. The first is similar to a "car" with four wheels (or more) arranged symmetrically about the robot with the ability to steer some of the wheels (see Figure 3-5). Sojourner is this type of robot, although it is far more complex than what you would think of as a car because of its ability to run over uneven surfaces.

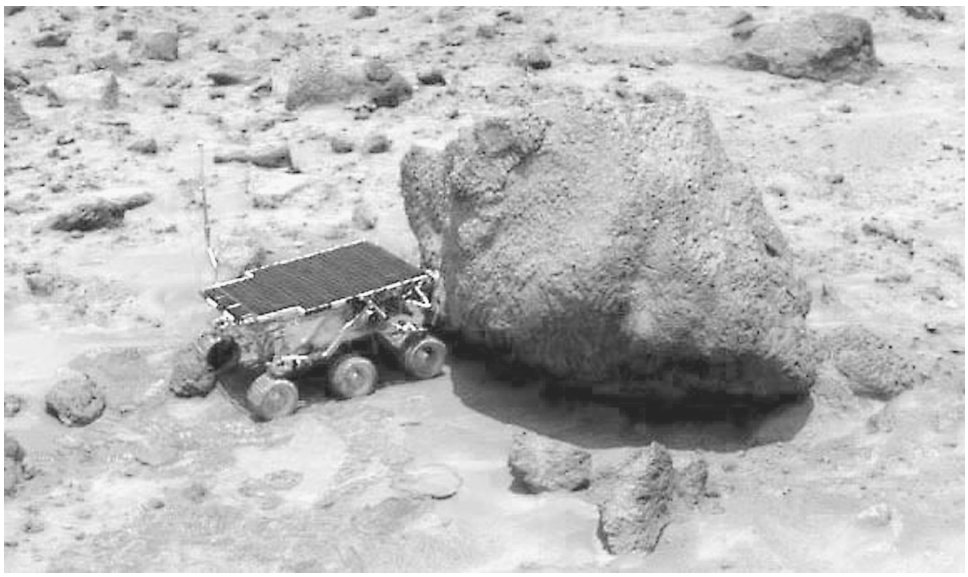

Figure 3-4 The "Sojourner Explorer" robot that explored the surface of Mars.

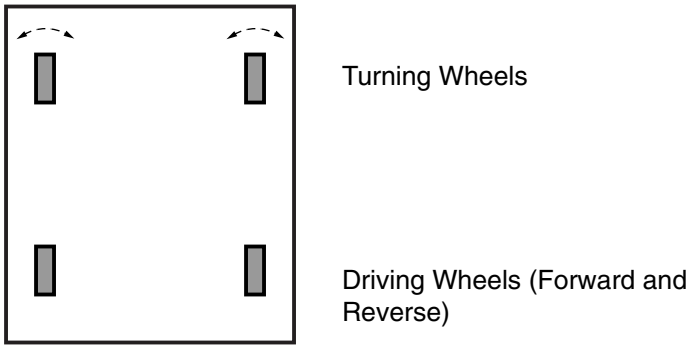

Figure 3-5 "Car"-type robot.

There are some advantages of the car-type robot for the hobbyist, namely the plethora of different toys, models, and kits that can be used as a base for this type of robot. Radio control cars are frequently used as bases for these types of robots because they already have a drive train and steering mechanism that the robot can interface with.

A popular modification of the car-type robot is the elimination of one of the wheels to create a "tricycle" robot (Figure 3-6). This type of robot can be much simpler to design from scratch than the car-type robot.

The first area that a tricycle robot simplifies the design effort compared to a car-type robot is steering. As I've shown in Figure 3-7, the inside turning wheel is at a greater angle than the outside wheel. If the angles are not correct, one of the wheels will not turn freely causing the car to wander off course or to require inadvertent braking action.

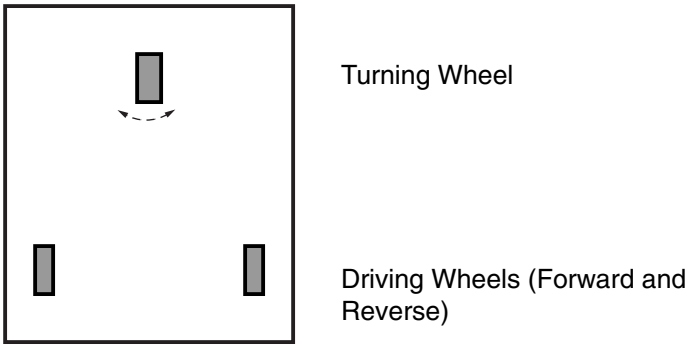

Figure 3-6 "Tricycle"-type robot.

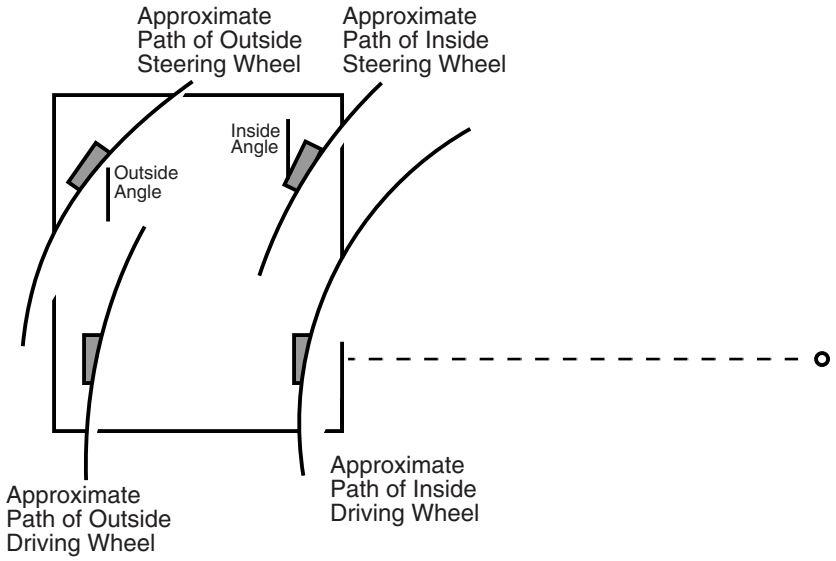

Figure 3-7 "Car"-type robot turning issues.

To avoid this problem, linkages between the steering actuator and the wheels have to be designed in such a way that the wheels turn appropriately for the radius of the turn. This is routinely done in real cars and many toys, but it can be difficult if you are designing the car yourself.

Along with the angle of the turning wheels, you'll notice that if the car were to turn a full circle, the inner wheel would roll over a shorter path than the outer wheel. In real cars, a gearbox known as a "differential" is used to balance the movement of each of the two driving wheels. In many toys and model cars, you will see that the problem is avoided by driving only one of the two rear wheels and letting the other roll freely.

The tricycle robot avoids this problem by letting the two rear wheels roll freely (and independently) if the "Turning Wheel" shown in Figure 3-6 is used as the drive

wheel instead of the rear wheels.. This is a common solution to the two problems that I've noted with the car robot and is usually not very difficult to implement.

The simplest robot to create is the "differential drive" robot (see Figure 3-8) in which two independent motors are used for driving the robot as well as steering it. This is the type of robot that was chosen for the *TAB Electronics Build Your Own Robot Kit*.

The major downside of this type of robot is its inability to work over an uneven surface. While other wheeled robots have difficulty working over rough terrain, they can be designed to do so (like the Sojourner). While the two driving wheels can be increased in size, the need for a front or rear "skip" or "castor" makes negotiating different surfaces difficult.

To move forward or backwards, the two motors run in the same direction. Rather than physically turning the wheels, the two wheels work at different speeds to effect a turn. An advantage of the differential robot is its ability to make very tight turns. If both motors are running in different directions, then the turn will rotate around an axis between the two wheels.

The motions of a differentially driven robot are described in the table below:

Movement	**Left wheel**	**Right wheel**	**Comments**
Move forward	Forward	Forward	
Move backward	Backward	Backward	
Turn left	Stopped	Forward	The basic turns will use the stopped wheel as the pivot point (axis) for the turn.
Turn right	Forward	Stopped	
Rotate left	Backward	Forward	The "rotate" turns will NOT move the robot's center of mass. The point in the middle of the two wheels will be the pivot point (axis) for the turn.
Rotate right	Forward	Backward	

Even though it is much more complex, the "caterpillar" treads of a bulldozer are actually a differential drive and can mimic many of the motions described above.

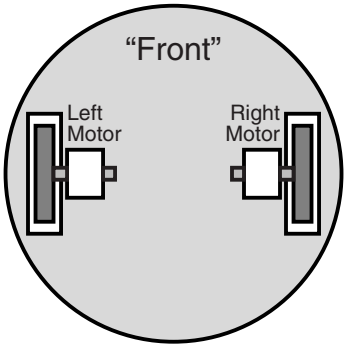

Figure 3-8 Differentially driven robot.

In addition, you will find that there is a fair amount of "wheel spin" in any wheeled robot in which the wheels turn without a corresponding movement in the robot. This can make it very difficult to gauge how far a robot has gone and/or in what direction it is pointing. This is a very important point to understand and it is not unique to differential drive robots; different types of robots have issues for determining how far they have moved and what is their current position. I will discuss navigation and position determination later as an advanced topic.

Locomotion and Actuators

Many real-life devices that you will have to control digitally in a robot for movement are electromagnetic, such as relays, solenoids, and motors. These devices cannot be driven directly by digital devices because of the current required, the binary value of digital circuits, and the "noise" generated by them. This means that special interfaces must be used to control electromagnetic devices.

The simplest method of controlling these devices is to just switch them on and off by supplying power to the "coil" in the device. The circuit shown in Figure 3-9 is true for relays (as is shown), solenoids (which are coils that draw an iron bar into them when they are energized), or a DC motor (which will only turn in one direction).

In this circuit, the digital circuit turns on the "Darlington" transistor pair causing current to pass through the relay coils closing the contacts. To open the relay, the output is turned off (or a "0" is output). The "shunt" diode across the coil is used as a "kickback" suppressor. When the current is turned off, the magnetic flux in the coil will induce a large back EMF (voltage) which has to be absorbed by the circuit or

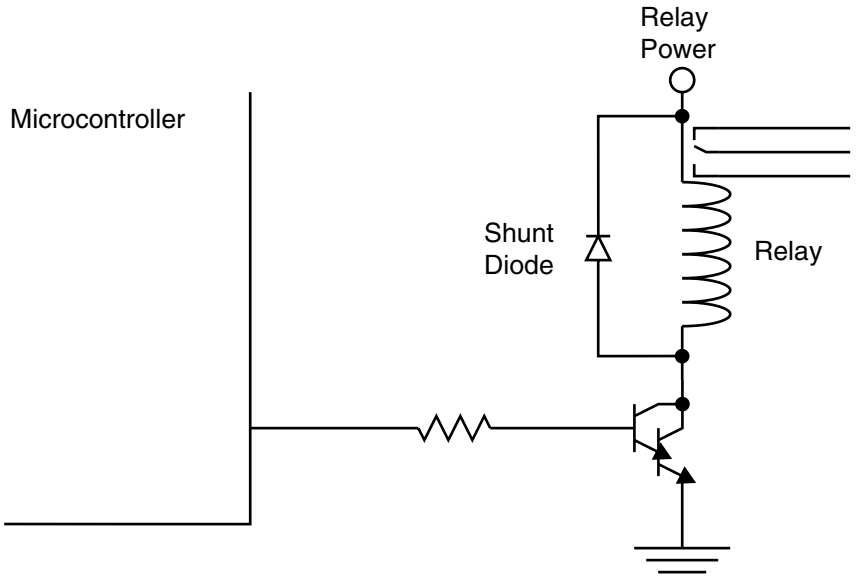

Figure 3-9 Microcontroller relay con.

there may be a voltage spike that can damage the relay power supply and even the driving digital electronics. This diode must *never* be forgotten in a circuit that controls an electromagnetic device. The kickback voltage is usually on the order of several hundred volts for a few nanoseconds. This voltage causes the diode to break down and allows current to flow, attenuating the induced voltage.

Rather than design discrete circuits to carry out this function, I like to use integrated chips for the task. One of the most useful devices is the ULN2003A (See Figure 3-10) or the ULN2803 series of chips that have Darlington transistor pairs and shunt diodes built in for multiple drivers.

Standard direct current (DC) motors can be turned on and off using the identical switching circuit presented above for controlling a relay. The problem with using these circuits is that they will run only in one direction. If you want to reverse the direction the robot is moving in, or turn (as in the case of a differential drive robot), you will have to be able to turn the motor in either direction.

In many robots, including the *TAB Electronics Build Your Own Robot*, you will see a network of switches used to control turning a motor in either direction. This is known as an "H-bridge" and is shown in Figure 3-11.

In this circuit, if all the switches are open, no current will flow and the motor won't turn. If switches "1" and "4" are closed, the motor will turn in one direction. If switches "2" and "3" are closed, the motor will turn in the other direction. Both switches on one side of the bridge should *never* be closed at the same time since this will cause the motor power supply to burn out or a fuse will blow because there is a short circuit directly between motor power and ground.

I have seen a number of hobbyist robots that implement the H-bridge motor control using four relays. While this works, I would tend to point people toward an all-transistor design. Using relays for the switches is expensive, both in terms of cost as well as power consumed (along with the motors, you have to power the relay's coils). Another problem with using relays is that the speed of the motor cannot be easily controlled.

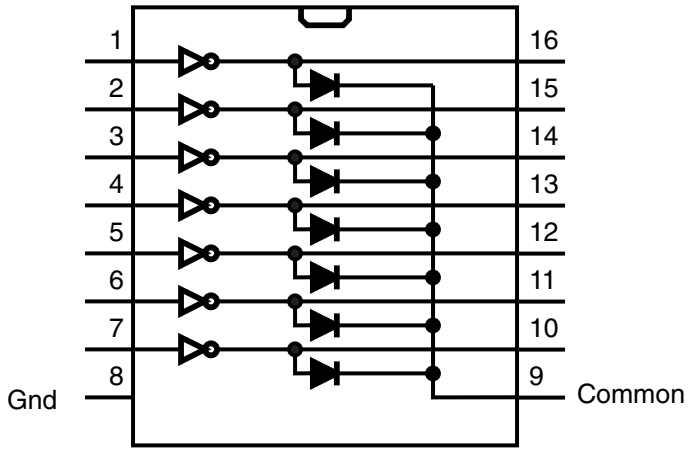

Figure 3-10 ULN2003A driver array.

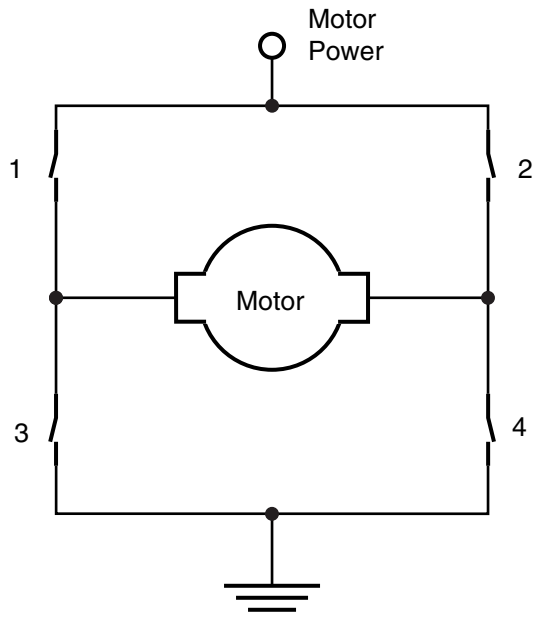

Figure 3-11 "H" bridge motor driver.

Transistors provide a better method of controlling motors. They are much cheaper than relays, more reliable, can switch much faster, and consume very little "parasitic" power. A very clever circuit, used in the *TAB Electronics Build Your Own Robot Kit,* is shown in Figure 3-12. When current is applied to either the "Forwards" or "Backwards" terminals, the NPN transistor the terminal is connected to is turned on, creating a path between a PNP transistor and the NPN transistor on the opposite side as shown in Figure 3-13.

Applying current to the other terminal will cause a current path through the motor in a different direction. As I've cautioned above, both terminals should not be energized to avoid the potential of burning out the motors and/or the driving transistors.

In the code that I developed for the TAB Electronics Build Your Own Robot Kit, I was only able to put in four different "on" duty cycle levels for the PWM. When I originally wrote the code, the PWMs duty cycles were given the values of 0, 25, 50, 75, and 100%. When this was implemented with the robot, I found that only the 75 and 100% PWM duty cycle levels resulted in acceptable motor power.

The reason for the poor power in the robot at 25% and 50% PWM duty cycle levels was due to the way in which power is transferred in a PWM. DC power is defined using the formulas:

$$\begin{aligned} \text{Power} &= V * I \\ &= (V ** 2) / R \\ &= (I ** 2) * R \end{aligned}$$

Since motors are current-based devices, the last formula is the most relevant.

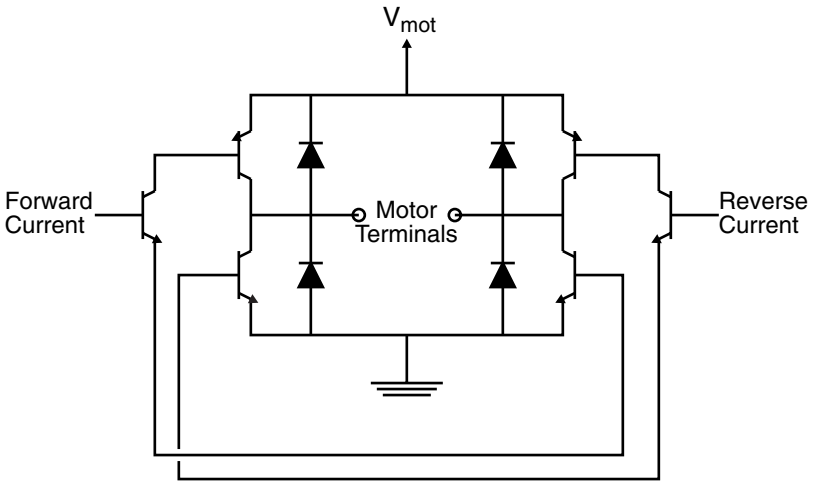

Figure 3-12 "Best" H-bridge controller for small motors.

If the PWM duty cycle is at 50%, then the current available is one-half of the total possible and according to the formula below, the power is one-quarter the power available to the motor at a 100% duty cycle PWM.

$$\begin{aligned} \text{Power} &= (I ** 2) * R \\ &= ((1/2\ I) ** 2) * R \\ &= 1/4 * (I ** 2) * R \end{aligned}$$

In the *TAB Electronics Build Your Own Robot Kit*, I "rescheduled" the PWM to being on for the set number of μsecs in the 52-μsec PWM period to the following levels, which gives the calculated power levels from 100% power:

"On" time in μsecs	**PWM duty cycle, %**	**Effective power, %**
0	0	0
33	63	40
39	75	56
45	87	75
52	100	100

As shown in the chart above, a PWM duty cycle of 75% results in only 56% of the total available power being driven to the motors. This rescheduling of the PWM gives a much more effective speed range for the motors.

The motor control used in the *TAB Electronics Build Your Own Robot* consists of the H-bridge circuit shown in Figure 3-12 and uses Microchip PICmicro® microcontroller code to provide the PWM signal described above. When the robot first powers up, the PWM duty cycle is set at 87%. The reason for using the 87% duty cycle instead of the 100% duty cycle is to conserve the battery power—using the 87% duty cycle PWM will result in significantly longer battery life without a large drop in performance of the robot.

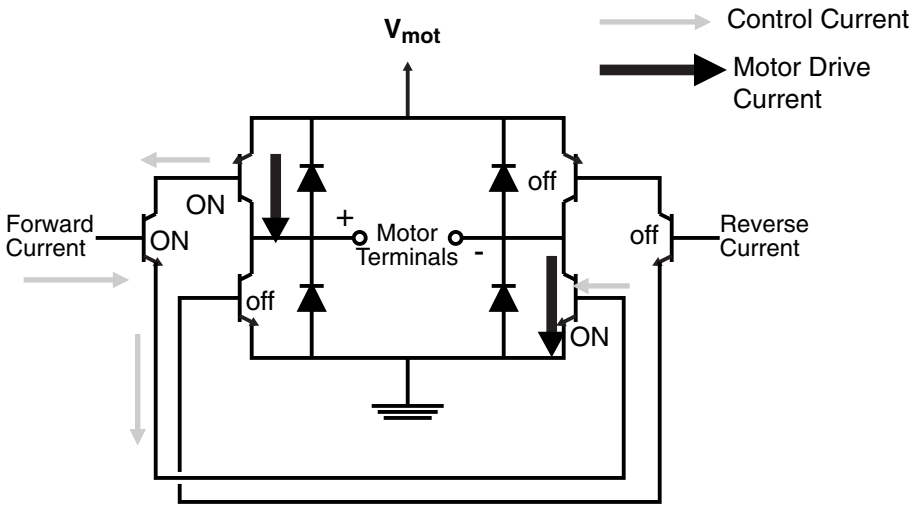

Figure 3-13 Operation of H-bridge motor controller.

Fuel Source

While there have been many, many cartoons with characters using a really long extension cord to their fans (which blow air across sails) for their electric cars, the most practical way of powering a mobile robot is to use alkaline and rechargeable "radio batteries." The different types of batteries have different characteristics that you should be aware of.

You are probably familiar with three types of batteries: alkaline-disposable "dry cells," or "radio," batteries (which I collectively refer to as "alkaline batteries"), rechargeable nickel-cadmium cells ("Ni-Cads"), and nickel-metal hydride ("NiMH"). The last type of battery is the "lithium" cell that is normally used for clocks and retaining data in CMOS devices. These batteries may appear to be similar, but they have different operating characteristics.

All types of batteries are given a "rated life" in the units of amperes-hours (AH). This rating is used to determine how long a battery will provide its rated operating characteristics with a given drain. For the 9-volt battery used with the *TAB Electronics Build Your Own Robot Kit*, the 9-volt battery that powers it will have a 560 to 1200 mAH rating, depending on which type is used. Of the three types of batteries discussed in this section, alkaline batteries have the greatest AmpHour rating, with lithium having the smallest. Lithium batteries are designed for providing a "trickle" (very small) current for a very long time.

One of the biggest differences between the battery types is the voltage output. For alkaline batteries, the voltage output on new batteries is 1.7 to 2 volts per cell. This voltage tends to drop linearly over use and 1.5 volts per cell is output when half the charge has been used up. Ni-Cads tend to output around 1.2 volts per cell, NiMH

about 1.5 volts per cell, and lithium provide much higher voltage at about 2.2 or higher volts per cell. To provide higher voltages, cells are connected in series and their voltage outputs summed together. For example, 9-volt "square" batteries have six 1.5-volt cells built into them to provide the nominal 9-volt output.

The next major difference about using batteries is the voltage output over time as current is drawn. This is different for each of the different types of cells as is shown in Figure 3-14.

Ni-Cads, NiMH, and lithium batteries tend to give constant voltage while they are being drained but "die off" suddenly when their charge is depleted. Alkaline batteries tend to have their output voltage linearly drop as the charge within the cell is depleted. This characteristic is what separates the alkaline cell from the other types and makes them unsuitable for use with the *TAB Electronics Build Your Own Robot Kit*.

Another characteristic that differs among the various types of battery cells is how long they can retain a charge. Lithium and Ni-Cads will keep their charge for a very long time if there is no drain on them. As you have probably noticed, alkaline and other "dry cells" will "leak" and loose their charge over time. Ni-Cad cells are particularly good at maintaining charge and will stay charged for 10 years or more. (In fact, this was the major selling point when Ni-Cad cells first came on the market.) Lithium cells also have good charge-retention capabilities, although they generally need replacement after five or so years even if their rated capacity hasn't been used.

Warning! Lithium cells and alkaline cells cannot be recharged using Ni-Cad- or NiMH-recharging equipment, and attempting to recharge the batteries can result in their exploding. While there are some alkaline batteries that can be recharged, only

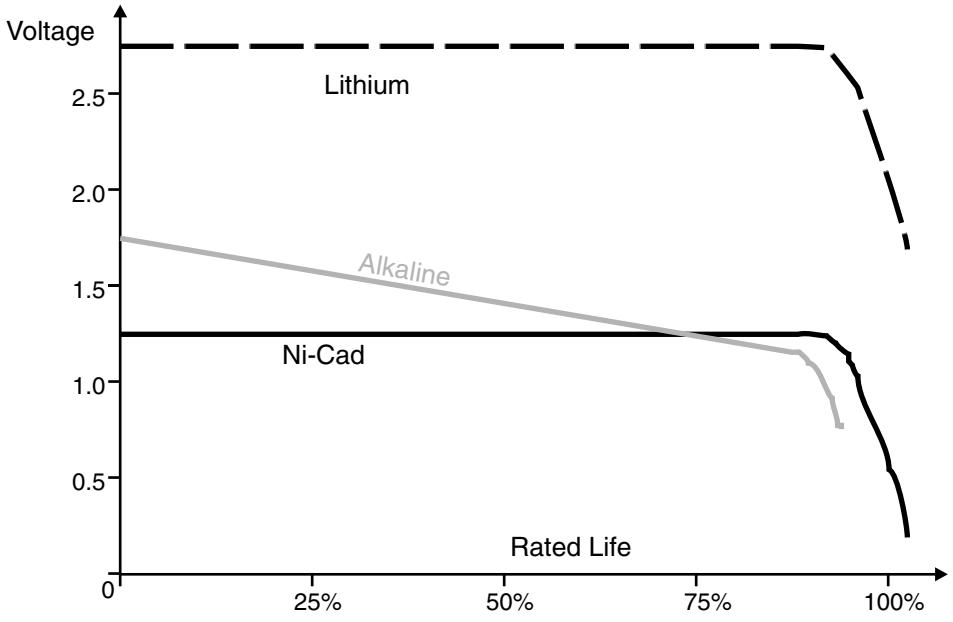

Figure 3-14 Different battery type operation.

use the manufacturer's recharging equipment. When a lithium cell is depleted—throw it out!

Ni-Cad batteries have one potential problem that you should be aware of. If they are used for the same length of time before being recharged, they can develop what is known as "memory" and will stop producing current after this length of time. To avoid this problem, Ni-Cads should be used for varying lengths of time and periodically "deep discharged" (run to almost zero volts output). This will break up the "dendrites" that build up over time and are the cause of "memory." For many people, memory is not a problem with their applications, and some of the newer Ni-Cad cells are designed in such a way that "memory" is not a problem.

Before discussing voltage regulation for digital electronics, I want to discuss one other aspect of selecting the batteries to be used for an application. If you look at the technical information for different batteries (normally available on a manufacturer's Web site), you will find that in terms of total charge (mAH rating), there is very little difference between a very expensive battery and a very cheap one.

In fact, you may have been told that expensive batteries are a "rip off" and that you will get similar performance using cheaper batteries. Looking at the technical specifications for different batteries, I have found that an expensive alkaline battery, which costs over three times more than a cheap carbon battery, has a mAH rating that is only 3% better than the carbon battery. Given this, you are probably wondering what advantage the more expensive batteries provide.

The difference comes from the amount of internal resistance (known as "iR") built into the battery. A cheap carbon battery can have tens (or even hundreds) of ohms of internal resistance, while an expensive alkaline battery has an internal resistance measured in fractions of an ohm.

In many applications where there is not a significant current drain (i.e., running a transistor radio), the inexpensive carbon battery does provide a better price-to-performance solution than an expensive alkaline battery. For applications that have a large current drain, like the *TAB Electronics Build Your Own Robot Kit*, this internal resistance is a significant problem and will either hinder the movement of the robot or will not allow enough current to flow for the motors to turn at all.

Ni-Cad rechargeable batteries *may* work, but the lower per cell output voltage and "memory" could be a problem when working with this robot.

I recommend that you *only* use NiMH batteries to keep your costs down (although you might want to initially use an alkaline battery to test out the operation of the robot). Note that I do not recommend rechargeable alkaline cells because the voltage per cell decreases after each charging cycle.

In the development of the *TAB Electronics Build Your Robot Kit* I tried a number of different manufacturer's batteries to find out which one is "best." The measurement criteria for "best" are low cost and long active robot life.

As I write this, the best nonrechargeable alkaline battery that I have found is the Eveready "Energize e^2" with the Duracell "Ultra III" being a close second. The best NiMH cell that I have found is the Radio Shack product (Radio Shack part number 23-299).

I am hesitant to recommend or warn against certain products simply because the battery marketplace is constantly changing. The three products I have listed

above all work very well in the *TAB Electronics Build Your Own Robot Kit* as I write this in August 2001. I do not know what is going to be available in the future, and I recommend that you experiment with different products to find what works best for you.

You should get an active robot "life" of anywhere between 20 minutes to half an hour using these batteries. When the motors are turned off and the Basic Stamp 2 is installed in the robot, you will find that the actual life of the robot can be up to 100 hours, depending on the initial state of the battery.

The *TAB Electronics Build Your Own Robot Kit* is unique compared with most other hobby robot kits as it only requires one battery for power. Most other robot kits will have a battery for the digital electronics and one for the motors. This is done to minimize the effect of motor-induced noise on the digital electronics and to potentially simplify the power distribution to the robot.

The single battery design was chosen for the *TAB Electronics Build Your Own Robot Kit* to minimize your cost. In the motor design, there was a lot of simulation and testing done to make sure the power and drive signals were adequately filtered and that they do not affect the operation of the digital electronics (the PICmicro microcontroller and Basic Stamp 2 and any "AppMod"-added hardware).

With this work completed, the decision was made to go with the single battery, and a power supply using a voltage regulator was added to the circuit to convert the output voltage from the 9-volt battery to 5 volts for the PICmicro microcontroller. The Basic Stamp 2 has its own regulator on board. This power supply is very simple, very reliable, and protected against surges or low power conditions.

Voltage regulators are circuits that lower an incoming voltage to a specific level that can be used by another circuit (often referred to as the "load"). Along with lowering this voltage, sufficient current must be produced to drive all the devices in the load without affecting the regulated voltage.

When I am explaining basic electronics, I often use water volume and pressure as an analogy to electrical voltage and current. This is also a useful way of describing how a voltage regulator works.

The water voltage regulator example consists of water from a higher-pressure source that must be provided at a lower pressure. A common way of doing this is to use a bowl that contains a floating block connected to a valve, which regulates the flow of water into the bowl. This is used in automobile carburetors as is shown in Figure 3-15. As water is drawn out of the bowl, it is replaced when the float drops allowing more water in from the high-pressure source as shown in Figure 3-16.

While this device is very simple and easy to understand, the actual implementation can be quite "fiddly." Issues that have to be accounted for include the ability of the float to provide enough force to close off the valve and respond quickly when a volume of fuel is drawn from the bowl.

A similar situation exists with electronic voltage regulators (which work in almost the same way as a carburetor). A "linear" voltage regulator has the block diagram shown in Figure 3-17.

The "comparator" acts as the float, which provides the input to the transistor, which acts like the valve in the carburetor. As the current drawn from the "load" increases, the comparator will sense the increased load, which causes an output voltage drop and allows the transistor to pass more power through.

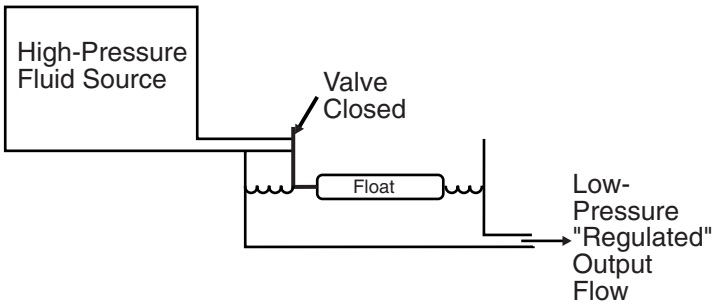

Figure 3-15 Car carburetor as a flow regulator.

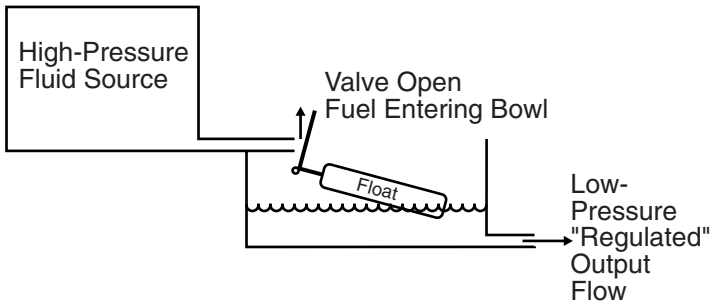

Figure 3-16 Car carburetor allowing fuel in.

While this seems very simple in theory, in practice this circuit is very hard to implement correctly.

One of the problems with this circuit is the heat that is dissipated by the transistor "switch." When the circuit is operating there is a voltage drop across this transistor along with the current being drawn across it. The power dissipation is expressed using the formula:

$$P = V \times I$$

where *V* is the voltage drop across the transistor and *I* is the current drawn by the load. For a voltage regulator that supplies one amp of current at 5 volts from a 12-volt supply, there will be a voltage drop of 7 volts across it. The actual power dissipated will be 7 watts (7 volts times 1 amp) and will require some kind of heat sink to dissipate this power to prevent the transistor and the circuit from being damaged.

Given this power loss, the linear voltage regulator is inefficient. For the low current requirement of the PICmicro and miscellaneous electronics, the voltage-level conversion power inefficiency is insignificant (especially compared to the draw of the motors). There are more efficient power supply designs, but the simple linear voltage regulator (the 78L05) used in this robot was the most expedient.

The 78L05 can supply up to 100 mA of current and is packaged in a transistor "TO-92," as shown in Figure 3-18. 78Lxx parts are more expensive than the straight 78xx versions (they can cost up to 60 cents each), but you would be hard pressed to design a circuit that can be built as cheaply as the 78Lxx.

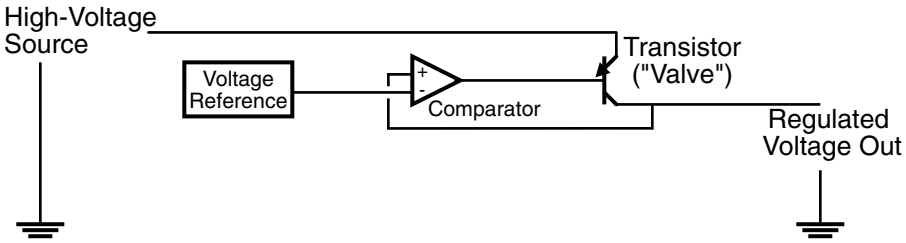

Figure 3-17 Simple regulator controlling voltage.

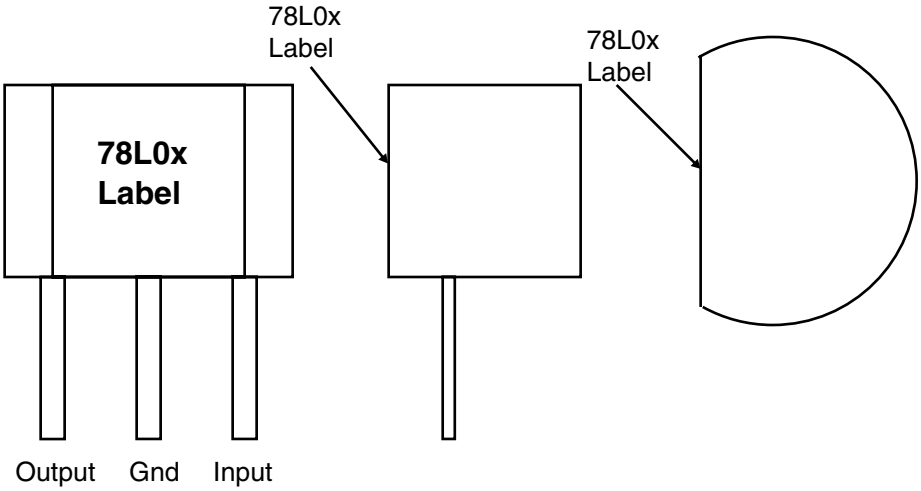

Figure 3-18 78L0x voltage regulator in TO-92 package.

Because of the relatively small output current and proportionately lower dissipated power, heat sinking is usually not required for the 78Lxx parts.

Sensors

The most efficient method of sensing collisions between a robot and other objects is to detect the objects before the collision. By detecting the object beforehand, the chance for sensing a collision caused by "noise" is minimized and the opportunity for the robot to respond is maximized before a potentially damaging collision takes place.

The science of detecting objects before a collision takes place is known as "proximity detection." It was first explored during World War II as a way of triggering antiaircraft artillery shells. Before the invention of proximity-detecting shells, the height of invading aircraft had to be estimated, and the time for the shell to reach this altitude had to be programmed into the shell's fuse. The operation of estimating the height (aircraft often changed altitude to spoil the gunner's estimates) was

fraught with errors. Along with this, the time needed to estimate the height and set the shell's fuse wasted valuable time that could be used shooting at aircraft that was bombing the gunner's positions.

The first proximity-detection shells worked by broadcasting a simple radio signal that was also received on board the shell. If the signal received by the shell varied, then the assumption was made that there was a metal object within the receiving distance of the shell that changed the (dielectric) property of the air. This object was assumed to be an enemy aircraft, and the shell was programmed to explode when the maximum change in the air's dielectric had been reached. Modern artillery shells use a miniaturized radar set that can be set to explode at a predetermined distance from an object.

Both the RF proximity detection and small radar sets can be used for detecting objects around a robot. Over the past few years there have been a number of designs developed for small and inexpensive radar sets that can be used for this type of application.

I tend to warn people away from using an RF proximity detector with a robot because its readings will change depending on the surface it is rolling on. This is not to say that an RF proximity detector is not accurate or reliable, just that it will have to be tuned each time the robot is placed on a different surface. This constant tuning means that a robot with RF proximity detection will have difficulty moving over different surfaces.

I'm sure you are familiar with I/R remote controls for your TV or other electronic devices. When I received my first one 20 years ago, I ended up trying it in different circumstances to see how well it worked. I started off by seeing how far away I could be for the remote control to work. I then tried bouncing the signal off a wall and other objects like pictures, windows, plants, and so on.

I admit doing things like this as a "guy thing," but it gave me the necessary background to try using an I/R detector as an object/collision detector. The theory I had was to reduce the current being passed through the I/R LED to the point where the reflected energy to the I/R detector would only be strong enough at a set distance. Figure 3-19 shows the I/R LED sending out a series of pulses that are reflected from an object and returned to the I/R detector.

The reason this method can detect objects at a set distance is due to a property of light sources that states that the power of the light decreases as the square of the distance. If you look at a light from 1 foot away, you will find that it is four times brighter than if you look at the light from 2 feet away. The light from 1 foot will be 16 times as bright as light seen from 4 feet away, and so on.

Using this property, the light from the I/R LED is attenuated (using a potentiometer) to the point where the brightness of the reflected light is sufficient at a set distance to trigger the I/R detector. In Figure 3-19, I show the light's strength diminishing as it travels through space by lighting the transmitted light "waves."

To prevent light from the I/R LED source from entering into the I/R detector from the side or from the rear, I have put an "opaque barrier" between the I/R LED

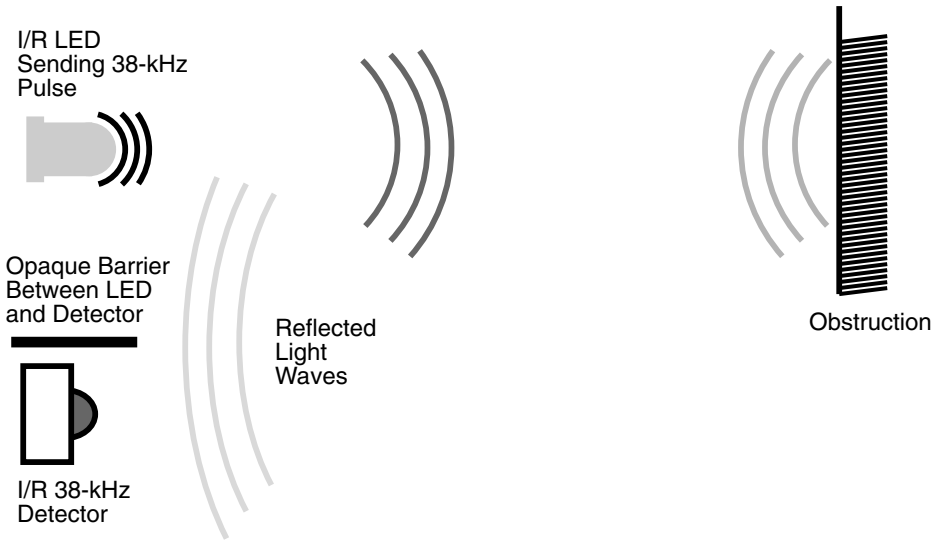

Figure 3-19 Infrared object detection.

and the I/R Detector in Figure 3-19. In the *TAB Electronics Build Your Own Robot Kit*, this opaque barrier was provided by a 47-µF capacitor, which is needed for power supply filtering of the I/R detector.

As you are undoubtedly aware, there is I/R light everywhere. The I/R detector filters out the "background" I/R light and just lets through any I/R signals that are present. It does this by having a built-in "bandpass" filter, which only passes signals that are modulated at 38 kHz. To implement the detector, I use the controller built into the robot to send out I/R light pulses at 38.6 kHz. An exact 38-kHz modulating frequency could not be generated by the PICmicro microcontroller, and the slightly different frequency lowers the detection capability of the I/R detector a bit (which is actually a good thing as it turns out).

When you look at the I/R LED and I/R detector placement on the *TAB Electronics Build Your Own Robot Kit*, you will probably be surprised that they are both offset relative to the front of the robot and that the angles are different (as shown in Figure 3-20). The reason for angling the I/R LEDs and the I/R detectors was to give the robot some ability to detect walls beside it for the wall-following behavior.

Most of the I/R LED's energy is directed to the front of the robot, which allows it to detect objects further away from it than to the side. If this method were not used, independent I/R LEDs and I/R detectors on the side of the robot would be required. This would increase the cost of the robot as well as the complexity of the software needed for the controller.

The I/R detector requires 6 or 7 cycles of the modulated I/R signal as is shown in the oscilloscope picture (Figure 3-21). To allow the I/R remote control to be used with the I/R detectors, 8 modulated pulses are sent to the I/R LEDs once every millisecond. Before sending additional pulses for collision detection, the I/R detectors are polled to see if a remote control is sending data to the robot.

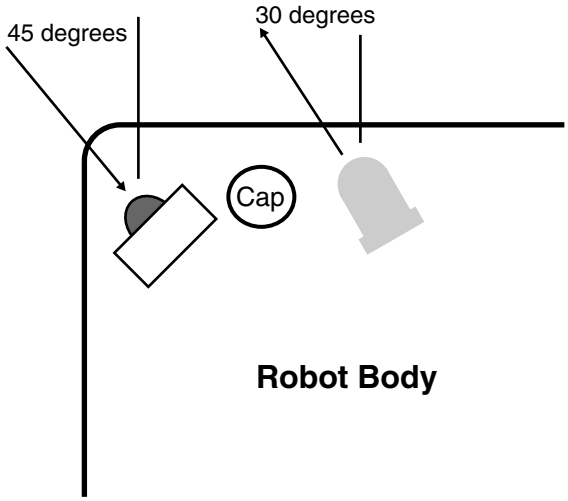

Figure 3-20 IR LED and receiver placement.

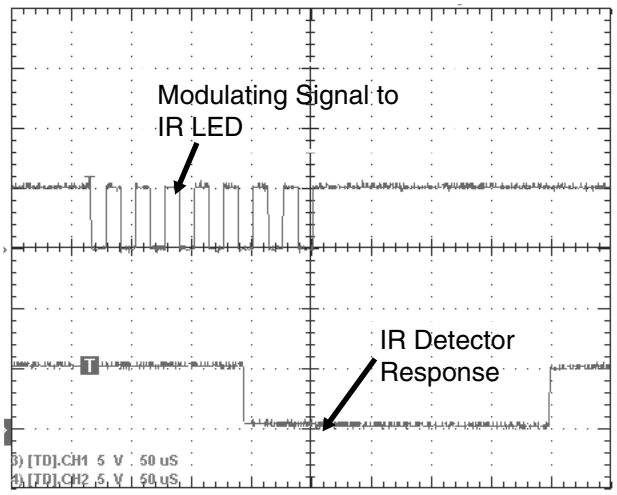

Figure 3-21 Infrared diode/detector operation.

Along with the ability to detect close objects, the *TAB Electronics Build Your Own Robot Kit* also has the built-in capability of sensing light levels in front of it. In the basic robot controller, this capability is used to work out a path to the brightest object that the robot is exposed to ("Photovore" mode).

To sense light levels, the robot has two cadmium sulphide sensors (known as "CDS" cells) placed at 45° angles from the centerline of the robot (Figure 3-22). The resistance of the CDS cells changes according to the amount of light falling on them. They are different from photocells and photodiodes. In these cells, power is gener-

ated from the light falling on them and from phototransistors, which change the amount of current that can pass through them based on the amount of light they are exposed to. By varying the angles of the two CDS cells, each one will be exposed to different levels of light and their resistance will change accordingly.

The CDS cells have a "dark" resistance of 10K that decreases as they are exposed to light. In direct sunlight, the resistance of the CDS cells falls to about 1K.

To "read" the resistance of these cells, the most common circuit would be the voltage divider, in which the variable resistance of the CDS cell affects the voltage across it. The higher the resistance of the CDS cell (i.e., the less light it is exposed to), the higher its resistance and the higher the voltage it will have across it. This can be measured with an analog-to-digital converter (ADC).

Unfortunately, the Microchip PICmicro® microcontroller chosen for the robot's controller does not have a built-in ADC. The solution to the problem was to "read" the resistance of the CDS cell rather than measure the voltage of the CDS cell in a voltage divider.

The method of reading the resistance uses the characteristics of a charged capacitor discharging through a resistor. If the charge is constant in the capacitor, then the time to discharge varies according to the exponential curve shown in Figure 3-23.

The charge in a capacitor is proportional to its voltage. If a constant voltage (i.e., from a PICmicro MCU I/O pin) can be applied to a capacitor then its charge will be constant. This means that in the voltage discharge curve shown in Figure 3-23, if the initial voltage is known along with the capacitance and resistance, then the voltage at any point in time can be predicted.

The equation in Figure 3-23

$$V(t) = VStart\ (1 - e^{** -t/RC})$$

can be reworked to find R, if V, $VStart$, t, and C are known:

$$R = -t/C * \ln\ (\ (VStart - V) / VStart)$$

Rather than calculate the value through, you can make the approximation of 0.2 msec for a resistance of 10k and a capacitance of 0.01 μF with a PICmicro microcontroller, which has a high-to-low threshold of 1.5 volts. To measure the resistance in a PICmicro MCU, I used the circuit shown in Figure 3-24.

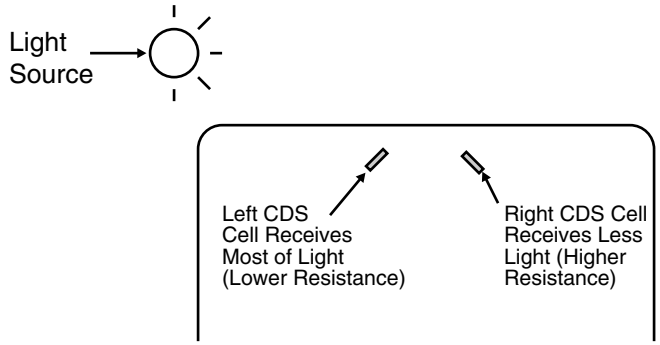

Figure 3-22 CDS cell placement on robot.

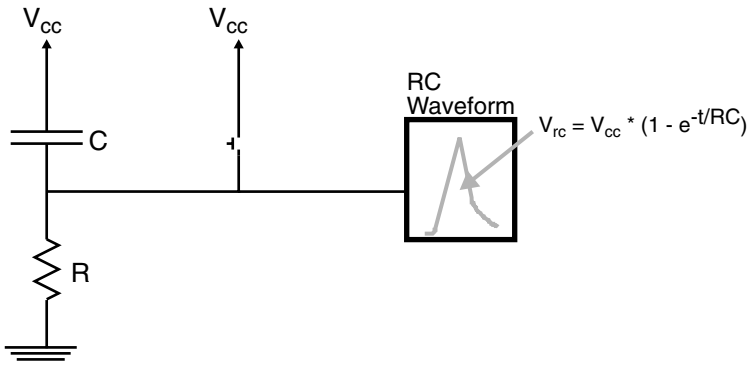

Figure 3-23 Measuring RC time delay.

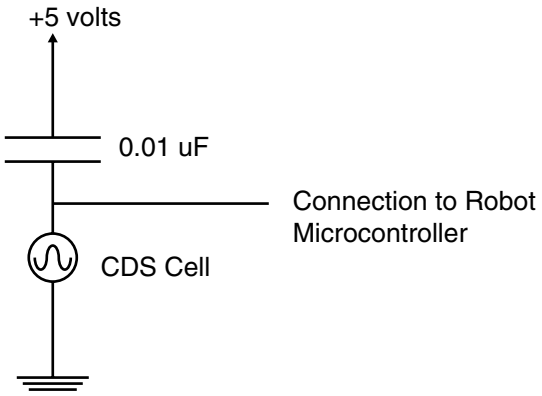

Figure 3-24 CDS cell wiring to robot microcontroller.

For this circuit, the PICmicro microcontroller I/O pin outputs a "high," which charges the capacitor (with some "leakage" through the CDS cell "resistor").

After the capacitor is charged, the pin is changed to input, and the charge in the capacitor draws through the CDS cell with a voltage determined by the $V(t)$ formula. When the pin first changes state, the voltage across the resistor will be greater than the threshold for some period of time. When the voltage across the potentiometer falls below the voltage threshold, the input pin value returned to the software will be zero. If the time required for the voltage across the pin to go from a "1" to a "0" is recorded, it will be proportional to the resistance through the CDS cell and the capacitor.

The pseudo-PBASIC code for carrying out the potentiometer read is:

```
ReadCDS                                 '  Return the Potentiometer's
Position
  pin output:  pin = 1                  '  Charge the Capacitor
  for i = 0 to charge
  next
  pin input: i = 0                      '  Let the Capacitor Discharge
ReadCDSLoop
```

```
  if pin = 0 then ReadCDSLoopDone
  i = i + 1                          '  Increment the Counter
  goto ReadCDSLoop
ReadCDSLoopDone                      '  Finished.  "i" is proportional
                                        to the
  return                             '   resistance of the CDS Cell
}  //  End ReadPot
```

The actual PICmicro microcontroller code that I developed for implementing this CDS cell read is not much more complex than the pseudo code above. Resistance or capacitance values can be read by the BS2 using the "RCTIME" function built into PBASIC. Note that when the CDS cell resistance is measured, the motors are turned off to prevent any loading of the power supply from affecting the voltage measurement of the CDS cell/capacitor circuit.

This method of reading a CDS cell's resistance is very reliable but not very accurate or particularly fast. To maximize the repeatability and minimize issues with component tolerances, the least two significant bits that are returned from the "ReadCDS" subroutine are discarded. For most applications this is not a problem, but it could be an issue if you are in a very bright room. The CDS cell's resistance may be so low that the "ReadCDS" routine returns zero.

This means that you should avoid very bright rooms and avoid using the *TAB Electronics Build Your Own Robot Kit* in direct sunlight. The robot will work well in normally lit rooms and is very responsive in dark rooms with bright light sources.

Control

An important aspect of the *TAB Electronics Build Your Own Robot Kit* is the controller built into the robot. The controller is more analogous to an animal's "peripheral nervous system" (PNS) than to its "central nervous system" (CNS), which most people know as the "brain." This is probably a surprising statement to make, but it should help you understand how the controller built into the robot works and what you can do with it.

In a human being, the peripheral nervous system is essentially all the nerves below the brain, as I've drawn in Figure 3-25. This system of nerves is responsible for control over the basic functions of the body as well as translating commands to different parts of the body and returning information from the various senses. Part of the basic functions provided by the PNS is the basic behavior of the person along with the initial response to stimulus.

The functioning of the PNS is identical to that of the controller built into the *TAB Electronics Build Your Own Robot Kit*. The controller, a Microchip PIC16C505, is responsible for controlling the operation of the motors based on commands from higher levels (i.e., you with the remote control or a Basic Stamp 2 controlling the operation of the robot). Along with controlling the motors, the controller is the interface for the sensors on the robot such as the I/R collision detection system and the light sensors.

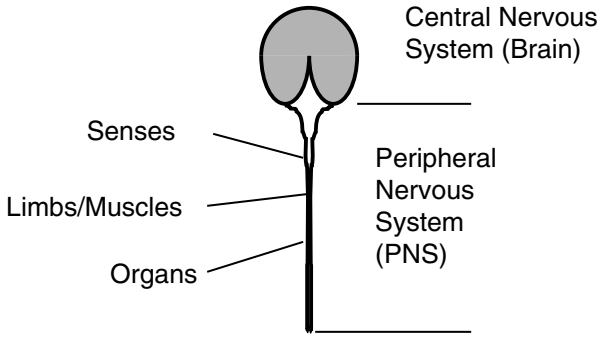

Figure 3-25 Central and peripheral nervous systems.

Just like the PNS, the robot's controller provides a "reflex" capability. Just as you stop short of walking into a wall, the robot's reflexes stop it from running into objects it encounters. This capability is somewhat unusual in a robot, and it can be disabled quite easily if the application requires it.

The four behaviors programmed into the controller are designed to be the basis of all operations carried out by the robot. This corresponds to the basic operations that are built into your body's PNS and what your brain controls.

Of the four built-in behaviors, the photovore and photophobe will stop when an object is in front of the robot. The other two behaviors, random motion and wall-following mode, never stop but rather react to obstacles in front and to the side of the robot.

There are two methods of providing higher level instructions to the controller built into the robot. The first is the remote control that is provided with the kit, which will allow you to specify a behavior, directly control the movement of the robot, and change the PWM speed of the motor. In this situation *you* are the robot's CNS.

Another option available to you is to install a Parallax Basic Stamp 2 into the socket provided in the *TAB Electronics Build Your Own Robot Kit* and have it provide commands to the robot directly. Because of the inclusion of the controller on the robot, these commands allow the Basic Stamp 2 (and you, as the programmer) to come up with software to interface to the peripheral functions of the robot directly.

If you are familiar with systems programming you may call the commands "Application Program Interfaces" (APIs), but I would like to discourage you from doing this because this changes the robot model from an organic model to a machine-based one. By changing the model, you will be changing the perspective by which the robot is controlled.

By adding the Basic Stamp 2, you are essentially adding a CNS to the robot that will allow sophisticated applications to be implemented quite easily and without a large amount of technical knowledge.

I have included the PICmicro® microcontroller source code that was used for the robot on the CD-ROM. If you are not familiar with programming or with the PICmicro microcontroller, this code will probably be overwhelming for you. The source code is provided as a reference that you can use later for your own robot designs.

Intercomputer Communications

Just as your home PC most likely takes advantage of the power and data of many other computers over the Internet, many robots (including the *TAB Electronics Build Your Own Robot*) take advantage of different computer systems that provide different capabilities. Robot computer systems typically do not implement true "network" connections, but do take advantage of small system communications methods that have been developed over the years. To close the discussion of the different systems that make up the *TAB Electronics Build Your Own Robot*, I would like to introduce you to the two methods used to interface to the controller, which are built into the robot. These two interfaces will allow you to execute advanced robotic applications or test out ideas with the robot connected to your PC.

The interface between the robot's built-in controller and a Parallax Basic Stamp 2 is a synchronous data communications protocol. In this method of transmission, a clock signal is sent along with serial data as shown in Figure 3-26.

The clock signal strobes the data into the receiver and the transfer can take place on the "rising" or "falling" edge of the clock. This protocol is available in the functions built into the BS2 and allows very simple communications between the BS2 and the *TAB Electronics Build Your Own Robot Kit*.

To communicate between your PC and the BS2, the asynchronous serial communications protocol known as "RS-232" is used. Asynchronous long-distance communications came about as a result of the Baudot "Teletype." This device, shown in Figure 3-27, mechanically (and, later, electronically) sent a string of electrical signals called "bits" to a receiving printer.

This data packet format is still used today for the electrical asynchronous transmission protocols.

The "data rate" is the number of data bits that are transmitted per second. For this example, if you were transmitting at 110 bits per second (which is a common teletype data speed), the actual data rate is 68.75 bits per second (or, assuming 5 bits per character, 13.75 characters per second).

In the data packet diagram shown in Figure 3-28, there are three control bits. The "Start Bit" is used to synchronize the receiver to the incoming data. In most "USART" (Universal Synchronous/Asynchronous Receiver/Transmitter) chips, there is

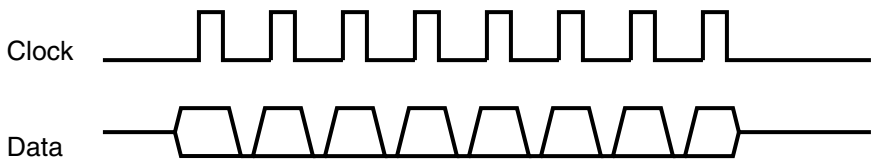

Figure 3-26 Synchronous data waveform.

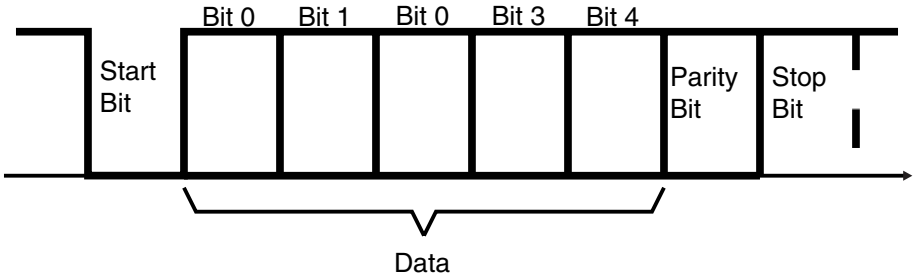

Figure 3-27 Baudot asynchronous serial data.

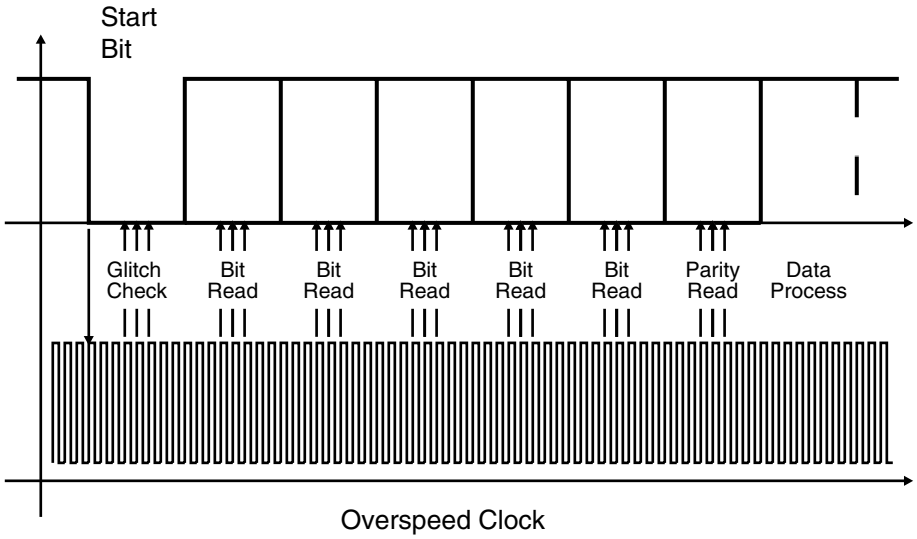

Figure 3-28 Reading an asynch data packet.

an overspeed clock running at 16 times the incoming bit speed, which samples the incoming data and verifies whether or not the data is valid.

When waiting for a character, the receiver hardware polls the line repeatedly at 1/16-bit-period intervals until a "0" (space) is detected. The receiver then waits half a cycle before polling the line again to see if a glitch was detected and not a start bit. Note that polling will take place in the middle of each bit to avoid problems with bit transitions (or if the transmitter's clock is slightly different from the receivers, the chance of misreading a bit will be minimized).

Once the start bit is validated, the receiver hardware polls the incoming data once every bit period multiple times (again to ensure that glitches are not read as incorrect data).

The "stop" bit was originally provided to give both the receiver and the transmitter some time before the next packet is transferred.

The "parity" bit is a crude method of error detection that was first brought in with teletypes. The purpose of the parity bit is to indicate whether or not the data was received correctly. An "odd" parity meant that if all the data bits and parity bits set to a "mark" are counted, the result will be an odd number. "Even" parity is checking all the data and parity bits and seeing if the number of "mark" bits is an odd number. Along with even and odd parity, there are "mark," "space," and "no" parity. "Mark" parity means that the parity bit is always set to a "1." "Space" parity always has a "0" for the parity bit, and "No" parity eliminates the parity bit all together.

Parity bits are a "crude" form of error detection because they can only detect one bit error (i.e., if two bits are in error, the parity check will not detect the problem). If you are working in a high induced-noise environment, you may want to consider using a data protocol that can detect and, ideally, correct multiple bit errors.

In the early days of computing, data could be transmitted at relatively high speed, but it couldn't be read and processed continuously. A set of "handshaking" lines and common voltage level protocols were developed for what became known as "RS-232 serial communications."

When RS-232 was first established, the typical packet contained 7 bits, which is the number of bits each ASCII character contained. To control the operation of the data transmission, the first 32 characters of the ASCII character set are defined as "special" characters (i.e., "carriage return," "backspace," etc.). Today, the most common form of asynchronous serial data packet is "8-N-1," which means 8 data bits, no parity, and one stop bit. This reflects the capabilities of modern computers to handle the maximum amount of data with the minimum amount of overhead, and with a very high degree of confidence that the data will be correct.

I will assume that the *TAB Electronics Build Your Own Robot Kit* will be connected only to an IBM-compatible PC via its serial (RS-232) port. This can be either a 9-pin or 25-pin connector at the PC, connected to the 9-pin connector on the robot via a "straight through" interface. This cable interface is very standard and very easy to implement.