Advanced Robotics with the Toddler

Student Guide

Version 1.0

Note regarding the accuracy of this text:
Many efforts were taken to ensure the accuracy of this text and the experiments, but the potential for errors still exists. If you find errors or any subject requiring additional clarification, please report this to stampsinclass@parallaxinc.com so we can continue to improve the quality of our documentation.

PARALLAX INC

Warranty

Parallax warrants its products against defects in materials and workmanship for a period of 90 days. If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Simply call for a Return Merchandise Authorization (RMA) number, write the number on the outside of the box and send it back to Parallax. Please include your name, telephone number, shipping address, and a description of the problem. We will return your product, or its replacement, using the same shipping method used to ship the product to Parallax.

14-Day Money Back Guarantee

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax will refund the purchase price of the product, excluding shipping / handling costs. This does not apply if the product has been altered or damaged.

Copyrights and Trademarks

This documentation is Copyright 2002 by Parallax, Inc. Toddler is a trademark of Parallax, pending registration. BASIC Stamp is a registered trademark of Parallax, Inc. If you decide to use the name BASIC Stamp on your web page or in printed material, you must state: "BASIC Stamp is a registered trademark of Parallax, Inc." Other brand and product names are trademarks or registered trademarks of their respective holders.

Disclaimer of Liability

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs or recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp application, no matter how life threatening it may be.

Internet Access

We maintain Internet systems for your use. These may be used to obtain software, communicate with members of Parallax, and communicate with other customers. Access information is shown below:

E-mail: stampsinclass@parallaxinc.com
Web: http://www.parallaxinc.com and http://www.stampsinclass.com

Internet BASIC Stamp Discussion List

We maintain two e-mail discussion lists for people interested in BASIC Stamps. The “BASIC Stamp” list server includes engineers, hobbyists, and enthusiasts. The list works like this: lots of people subscribe to the list, and then all questions and answers to the list are distributed to all subscribers. It’s a fun, fast, and free way to discuss BASIC Stamp issues and get answers to technical questions. This list generates about 40 messages per day. Subscribe at www.yahoogroups.com under the group name “basicstamps”.

The Stamps in Class list is for students and educators who wish to share educational ideas. This list generates about five messages per day. Subscribe at www.yahoogroups.com under the name “stampsinclass”.

Table of Contents

Preface

Walking robots remain largely unexplored in the area of hobby and education. The limitations include handling diverse terrain including stairs, more difficult programming algorithms and feedback, and generally more complex mechanical designs. This may be one of the reason most of our robots have wheels. However, people have a natural tendency to appreciate a walking robot: they seem more like human beings; they offer more entertainment value since they're fun to watch; and to make a biped robot walk successfully is a challenge worthy to pursue the concept.

The Toddler robot simplifies the walking robot concept. While the Toddler certainly won't be caring for the elderly, vacuuming the house or driving you to the store, it will provide a first exposure to the concept of a programmable biped. Two servos will prove to be quite limiting if you master this robot, but along the way you'll discover the complexities and rewards associated with learning to program a walker. Walking robots introduce embedded control in a positive, fun and friendly way.

The programming of Toddler is not trivial. When you are done with the Toddler experiments you'll be a much better BASIC Stamp programmer. We'd suggest that you first try the Boe-Bot projects as a prerequisite. If this isn't possible, take your time to absorb and understand how the program structures work and how PBASIC is utilized. Keep your BASIC Stamp Manual Version 2.0 handy to look up PBASIC syntax that isn't obviously understood. Seeing examples in a non-robotic use is often helpful.

If you need help, call or e-mail Parallax for technical support. We'd be pleased to help get your Toddler walking the way you want it to.

Recognitions

Although Parallax designed the Toddler, we recognize that the first time we have seen the two-servo concept employed for a walking robot is British robotics designer David Buckley's "Big Foot". Though the basic concept is simple, our research shows that Mr. Buckley created the ingenious use of two servos for a walker. Big Foot is a plywood kit available through Milford Instruments (www.milinst.com) of the United Kingdom. David Buckley endorses the Parallax Toddler robot and contributed to the Toddler design.

David Buckley's Big Foot	Parallax's Early Toddler Design
	

This curriculum was authored by Parallax, Inc. and Bill Wong of Pennsylvania. Bill is an editor with Electronic Design magazine and a serious BASIC Stamp robotics enthusiast. His daughter has won several County and State awards with her maze solving robotic projects.

Audience and Teacher's Guide

Students as young as 14 years old could be able to build and program the Parallax Toddler. Because of the Toddler's more extensive mechanical system and more complex programming we believe that the youngest student to have success with this kit would probably be about 12 years old. If you have experience otherwise please let us know at stampsinclass@parallaxinc.com. The Advanced Robotics with the Toddler text presently has no teacher's guide. Based on demand we may elect to produce the answers to challenge questions posed in this text.

Educational Concepts from the Toddler

Educators always ask Parallax what they will learn from our different curriculum. The Toddler is considered an advanced robotic project and generally will instruct the following concepts:

- Interaction between mechanical and electrical systems and the ability to tune hardware or adjust software to obtain desired results
- Advanced programming skills with the BASIC Stamp 2. An efficient Toddler program makes use of little-used Stamp programming tricks pertaining to the DATA statement, program routines that are reused and "configured" prior to execution, variable aliasing, general sound programming practices (constant/variable definitions that allow for program customization in just a few places rather than throughout an entire program)
- A step-wise process which starts with the basics and builds to something more complex and ultimately more useful

Copyright and Reproduction

Stamps in Class lessons are copyright © Parallax 2002. Parallax grants every person conditional rights to download, duplicate, and distribute this text without our permission. The condition is that this text, or any portion thereof, should not be duplicated for commercial use resulting in expenses to the user beyond the marginal cost of printing. That is, nobody should profit from duplication of this text. Preferably, duplication should have no expense to the student. Any educational institution wishing to produce duplicates for its students may do so without our permission. This text is also available in printed format from Parallax. Because we print the text in volume, the consumer price is often less than typical xerographic duplication charges. This text may be translated into any language with the prior permission of Parallax, Inc. obtained through stampsinclass@parallaxinc.com.

Experiment #1: Assembling the Toddler Robot

The Newest Member of the Family

No matter how easy it looks, you'll soon realize that the mechanical movements and BASIC Stamp code required to make a two-servo biped move in a distinct fashion is more complex than it's rolling counterpart, the Boe-Bot.

The Toddler is capable of doing many things a rolling robot can do if you've got the patience to tune the mechanics and software. Not only is the robot more entertaining than a rolling robot, you'll become a more proficient programmer as you learn to exploit the BASIC Stamp's capabilities. The Toddler demonstrates the importance of a PBASIC program that uses constants and variables, program pointers and EEPROM access for data storage. A well-designed program means you can easily tune the software for the right mechanical control in just a few places rather than rewriting your entire program.

The Toddler's motion is controlled using two servo motors (the type normally used in remote controlled airplanes). The Toddler's top servo motor is used to rotate the robot's center of gravity back and forth over the two feet, and the bottom motor moves both legs back and forth. The legs use a simple parallel linkage to the ankles that keeps the feet parallel to the ground. Both legs are linked together through the same motor so that as one leg move forward, the other moves backwards.

By controlling one motor at a time the robot can move forward, backward, and turn either left or right. By blending the control of both motors the robot can do move in a more coordinated fashion with smooth movements.

A surface-mounted BASIC Stamp 2 module controls the Toddler. The BASIC Stamp 2 is used throughout the Stamps in Class educational series and provides plenty of program space, speed and memory for use with a Toddler.

Let's Build the Toddler

The Toddler may be assembled a number of ways depending on the surface on which the robot walks, the type of additions you make with sensors and hardware and the speed which you program to robot to walk. The default assembly method is appropriate for hard, level surfaces and will be used to demonstrate code throughout this text.

Tools Required

You'll need a screwdriver and a pair of pliers and to build the Toddler.

A Note About Parts in the Toddler Kit

Appendix A includes a parts listing for the Toddler. These instructions refer to different pieces of hardware. If you are missing parts from your Toddler kit Parallax will replace them free of charge; if you break parts or want additional hardware for your customized Toddler you can order any piece on-line from our Component Shop (www.parallaxinc.com/componentshop).

If you have trouble identifying the type of part referred to in these instructions, see the color back cover of this text that shows each part with a colored picture and Parallax stock code.

Step #1: Install Top "Tilt" Servo

Parts Required:

- (4) 4/40 3/8" long pan-head machine screws
- (4) 4/40 nuts
- Toddler Body
- Toddler Servo Motor

Install the servo in the Toddler Body with the shaft down as shown in the picture. Position the servo squarely.

Using four (4) 4/40 3/8" pan-head machine screws and (4) 4/40 nuts, screw the tilt servo into the body. The easiest way to do this is to hold the nut with one finger while turning the screwdriver with the other hand.

Step #2: Install Bottom "Stride" Servo

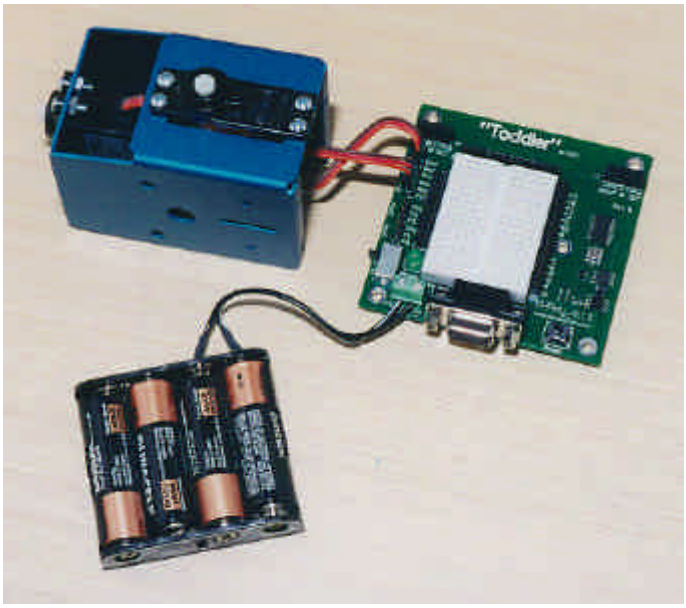
Parts required:

- (4) 4/40 3/8" long pan-head machine screws
- (4) 4/40 nuts
- (1) Toddler Servo Motor

Install the bottom servo with the shaft oriented towards the front of the robot.

Using four (4) 4/40 3/8" machine screws and (4) 4/40 nuts, screw the stride servo into the body.

Step #3: Electrically Center Servos



Parts required:

- Battery Box
- (4) AA Batteries (not included)
- Serial Cable
- BASIC Stamp software

The servos should be "centered" prior to any further Toddler assembly. This will ease any fine-tuning adjustments by allowing them to be made only in software. Don't skip this step – it will make future adjustments easier.

(Continued) Step #3: Electrically Center Servos

Plug the two servos into the Toddler printed circuit board "A – Servos – B" connector. The GND label on the board connects to the black servo lead.

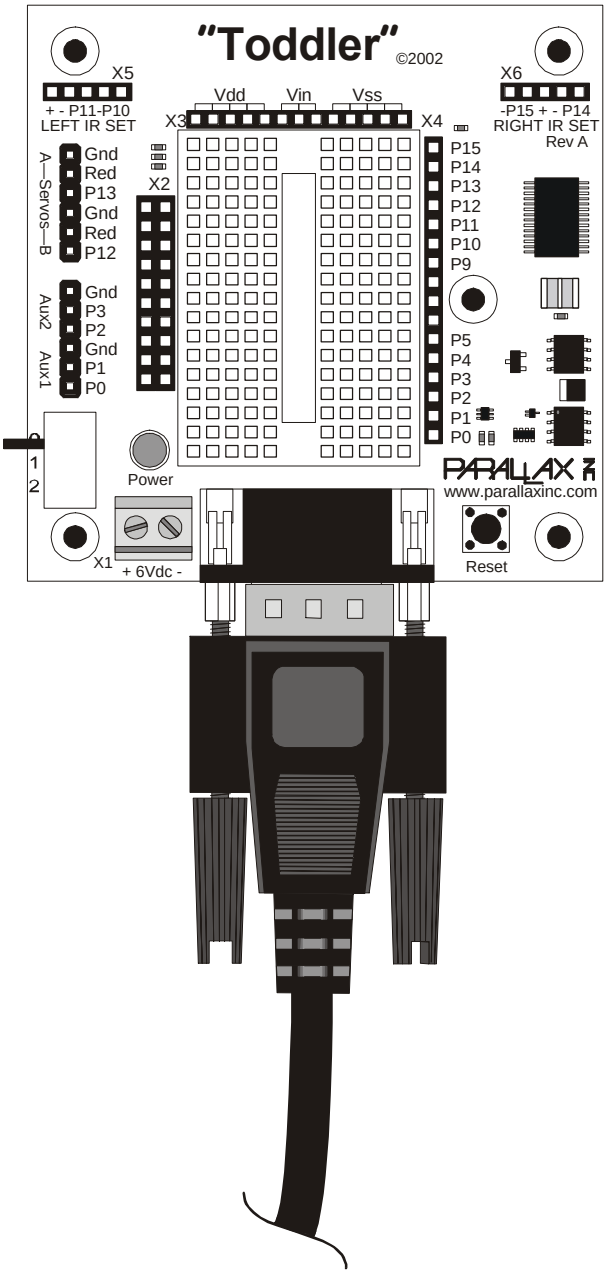
Install four batteries in the battery pack and take the entire assembly to your computer. The battery pack's white wire lead connects to the Toddler board's + terminal block. Use a screwdriver to connect these wires.

Using the serial cable, connect the Toddler board to the serial port of your computer. This connection is shown in this picture.

The Toddler has a three-position power switch. The state of each position is shown below. The three-position switch has a middle position that powers the entire circuit except servos. A complete schematic of the Toddler is included in Appendix B.

Position O – No Power
Position 1 – Power to everything except servos
Position 2 – Everything is powered

Place the power switch in Position 2. The next step is to program the BASIC Stamp.

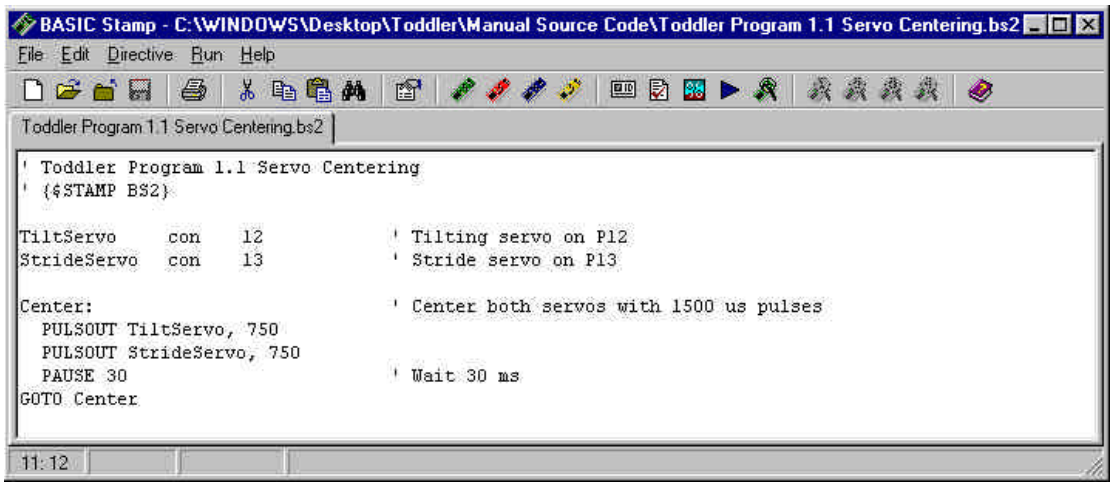


Open the BASIC Stamp Windows editor. [1] Write the following piece of code that will center both servos:[2]

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 1.1: Servo Centering
' {$STAMP BS2}

' -----[ Constants ]-------------------------------------------------------------

TiltServo             CON     12              ' Tilting servo on P12
StrideServo           CON     13              ' Stride servo on P13


' -----[ Main Code ]-------------------------------------------------------------

Center:                                       ' Center both servos with 1500 us pulses
  PULSOUT TiltServo, 750
  PULSOUT StrideServo, 750
  PAUSE 30                                    ' Wait 25 ms
GOTO Center
```

Download your code using the Run I Run menu or by pressing the button on the toolbar. This program runs in an endless loop. When the servos stop moving (this will happen almost instantaneously) move the power switch to off.

Figure 1.1: Windows Editor with Toddler Program 1.1 Servo Centering

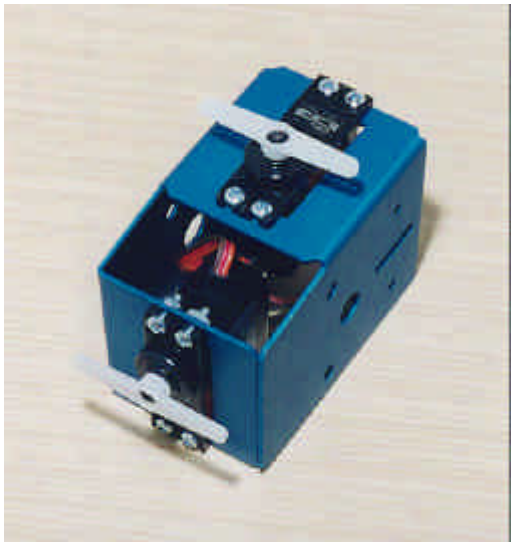


[1] The Parallax BASIC Stamp Manual 2.0 includes a "Quick Start" section that details how to open and launch the BASIC Stamp Windows editor.

[2] Source code in this text is available in a zipped file for download from www.parallaxinc.com/Toddler

Turn the power switch to position 0. Disconnect the servos from the Toddler board. Remove the batteries from the battery box and disconnect the leads from the Toddler's screw terminal.

Step #4: Install the Servo Horns



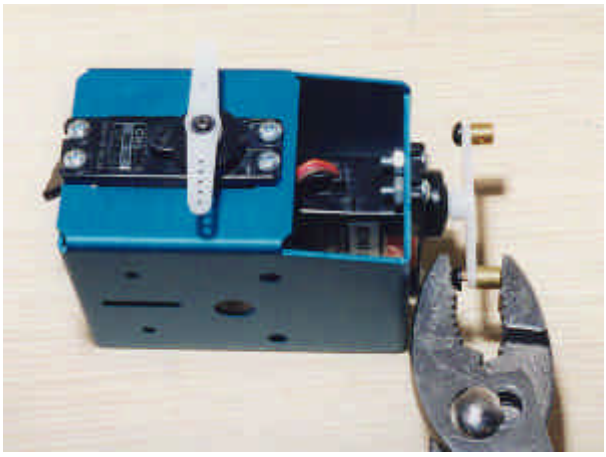
Parts required:

- Two servo horns
- (2) Small black screws to hold horn to servo

The servo horns should be installed as straight as the spline allows without turning the motor.

Secure each servo horn with the small black Phillips head screw using a screwdriver.

Step #5: Install Brass Wire Keepers on Stride Servo



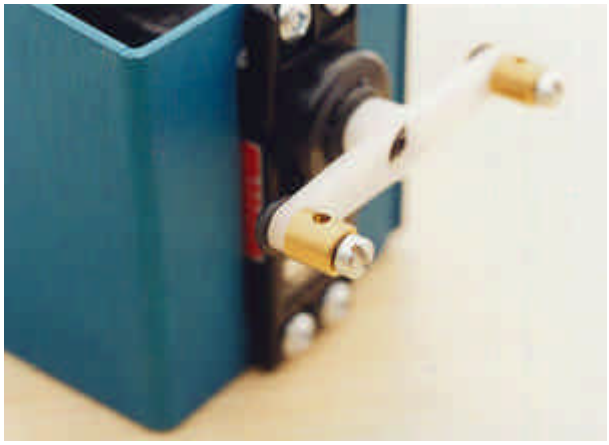
Parts required:

- (2) Brass wire keepers (brass holder, set screw and holding grommet) packaged in a bag

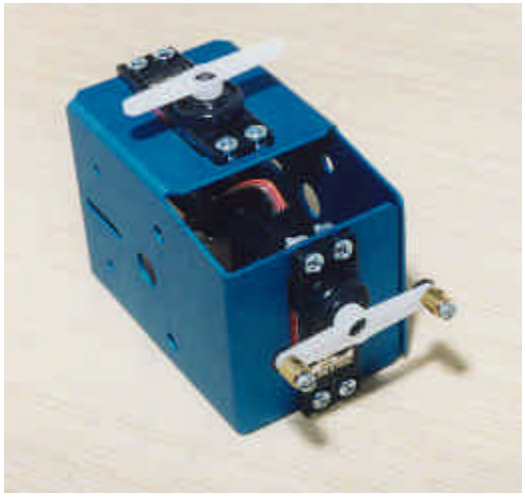
Attach the two brass wire keepers on the outermost holes of the bottom servo's control horn. Using pliers, press the rubber "keeper" onto the post of the brass wire keeper.

Put the two small screws into the threads of the brass wire keeper so they don't get lost.

Final installed brass wire keepers.



The brass wire keepers are for the bottom (stride) servo only. Don't install them on the tilt servo.



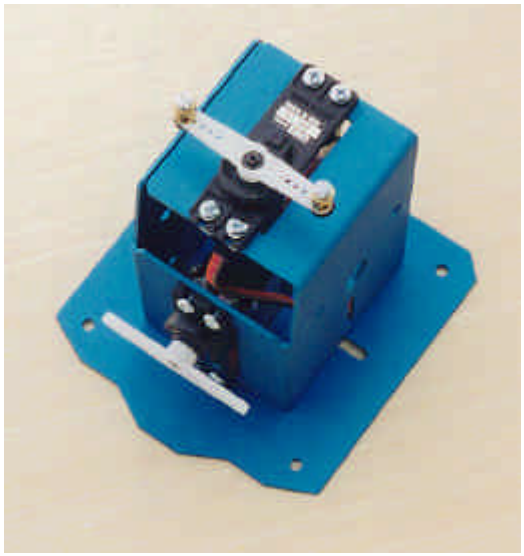
Step #7: Install Top Plate

Parts required:

- Toddler Top Plate
- (4) 4/40 3/8" flathead machine screws
- (4) 4/40 nuts

The top plate is most easily installed by turning the Toddler body upside down. Position a 4/40 nut over the hole and insert a 4/40 3/8" flathead screw through the top plate from the bottom. Hold the nut with one hand and tighten the screw with the other hand. Repeat process for three more holes.

Note: This step uses the "flathead" countersunk 4/40 screws, not the flat "panhead" screws.



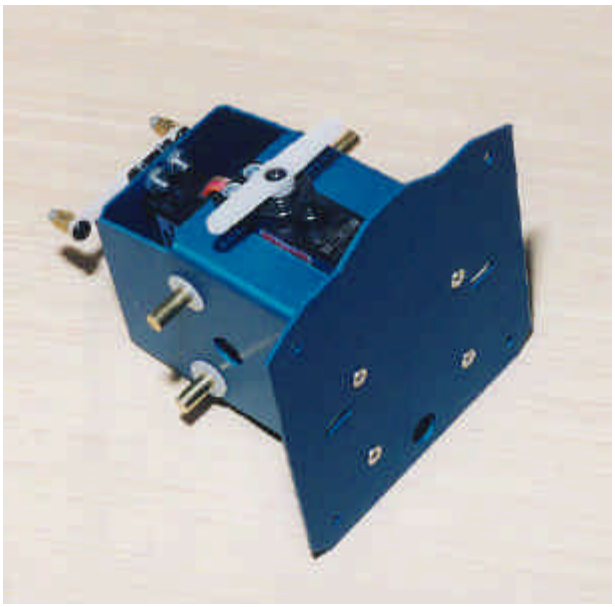
Step #8: Install Leg Rods

Parts required:

- (2) 3/16" 3" long brass rods
- (4) plastic washers

Slide the two 3" 3/16" brass rods through the two holes in the Toddler's body.

Slide a plastic washer over each rod. This washer will keep the Toddler's legs from rubbing against the body.

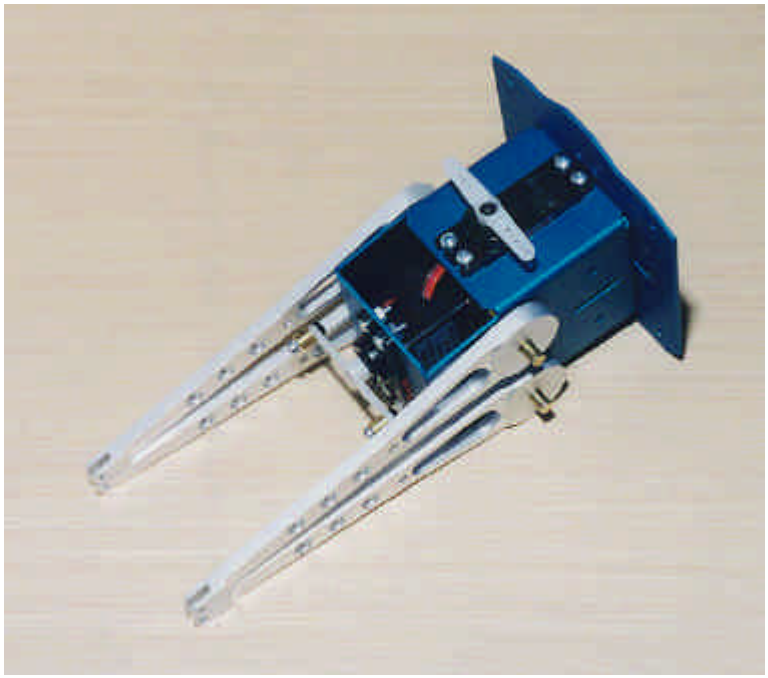


Step #9: Slide Legs onto Toddler Body

Parts required:

- (4) Toddler legs

Slide the four Toddler legs onto the ends of the brass rods going through the body.



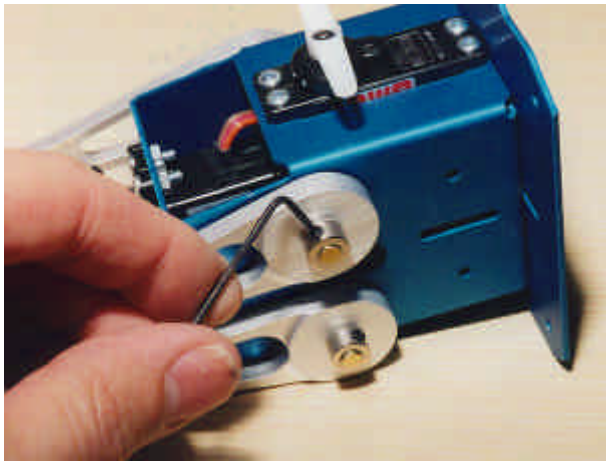
Step #10: Secure Legs to Toddler Body

Parts required:

- (4) 3/16" collars with setscrew
- L hex-key wrench in collar package

Find the package of 3/16" metal collars and L-key.

Slide the collars onto the brass rod. Tighten each collar using the L-key wrench. If the setscrew doesn't seem to tighten, slightly angle the wrench to prevent stripping of the set screw or wrench. Make sure the legs move freely when you're done.

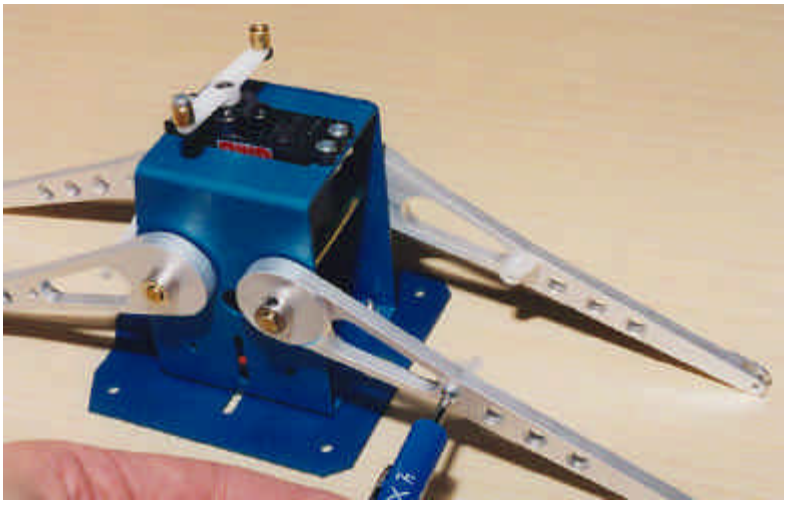


Step #11: Assemble Stride Linkages

Parts required:

- (2) 4/40 plastic wire keepers
- (2) 3/8" 4/40 panhead machine screws
- (2) brass right-angle wires

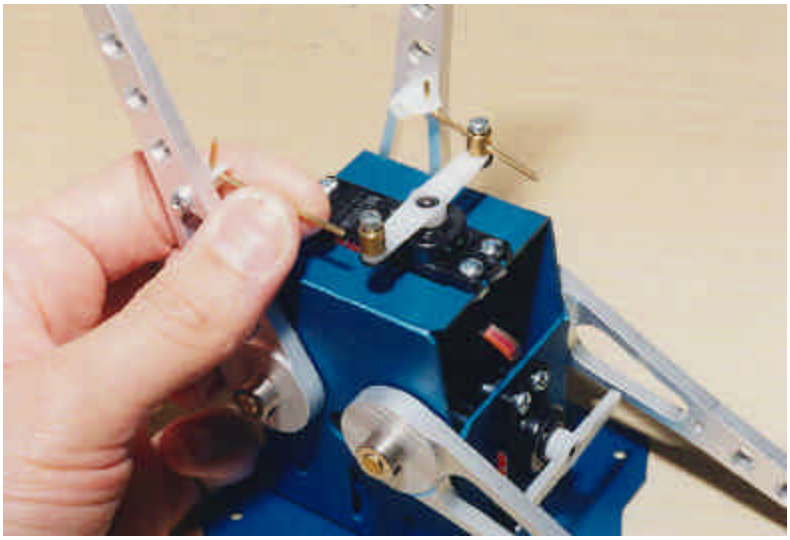
This is a two step process. First, insert a 3/8" 4/40 panhead screw through the holes on the rear left leg. Tighten the screw into the plastic right-angle bracket. Repeat for the process for the rear right leg. Rotate the body 180 degrees.



Using the two right-angle brass wires, slide the straight end through the brass wire keeper hole. Insert the short end through the top of the plastic right-angle bracket. Tighten the screw to hold the wire.

Adjust the linkages so the legs are vertical, not slanted to either side. Electrically center the servos again if necessary if the servo was accidentally moved.

Repeat for the other rear side.

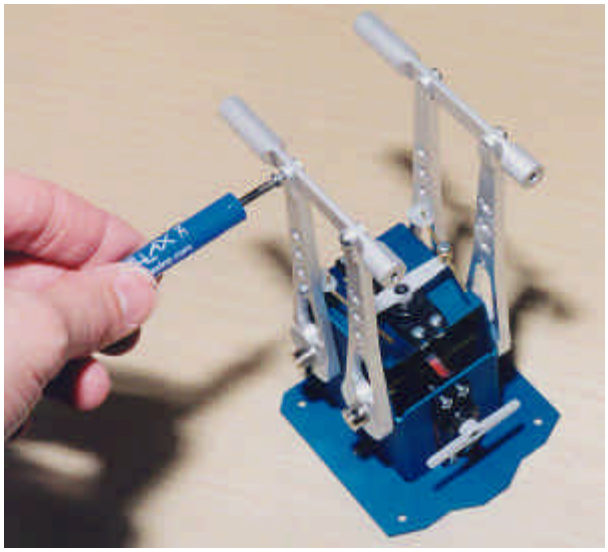


Step #12: Attach Ankles

Parts required:

- (2) ankle parts
- (4) 4/40 ¼" panhead machine screws

Attach the ankles to the legs using four 4/40 ¼" screws. The longer part of the ankle should be oriented towards the back of the Toddler's body. This placement moves the weight of the robot forward and provides better overall control.



Step #13: Attach Feet

Parts required:

- Toddler left foot
- Toddler right foot
- (4) 4/40 plastic alan-head screws
- 3/32" hex L-key wrench

Line up the ankle into the foot's 3rd hole from the instep. If it is too tight slightly bend the tabs of the feet outward.

Attach the left ankle to the left foot using two plastic 4/40 screws and the 3/32" L-key wrench. These screws have a round head that acts as a bearing surface on the robot's foot. Repeat for the right side. Ensure free tilting of each foot before proceeding to the next step.

Step #14: Install Ball Joints

Parts required:

- (4) ball joints (ball joint with post, nut)

Install a ball joint on the outermost hole of each foot. Securing the nut may require a small wrench to tighten the ball joint unless you have a pair of needle nose pliers. One way to do this is to hold the nut with a finger and turn the ball joint until tight.

Install two ball joints on the tilting servo control horn. Use the outermost two holes for these ball joints.

Step #15: Install Tilting Rods

Parts required:

- (2) .090" diameter 5.4" brass rod, 2/56 thread on each end
- (4) ball joint plastic cups with 2/56 thread

Thread two plastic ball joint cups onto the ends of the 5.4" brass rod. Place the rod next to the control horn and foot for sizing.

To properly position the ball joints make the finished piece about 1/16" longer than it needs to be as the robot stands straight up; this ensures that the out step of the feet will be firm on the ground and aids with turning.

When you've got the length about right, snap the rod onto the foot and servo control horn.

Repeat for the other side.

There is an easy way to remove the ball cups from the ball joint. If you need to make adjustments simply place a screwdriver between the ball cup and the Toddler's foot and carefully pry (snap) it off. It should pop off and can be pressed back on after you make a few turns to adjust.

Step #16: Install Tilting Rods (continued)

When you're done with this step your robot should look like this one.

When you pick up the robot, verify that the robot's feet are flat on the ground, or that the outsteps are angled slightly downward. Electronically center the servos with the BASIC Stamp if needed.

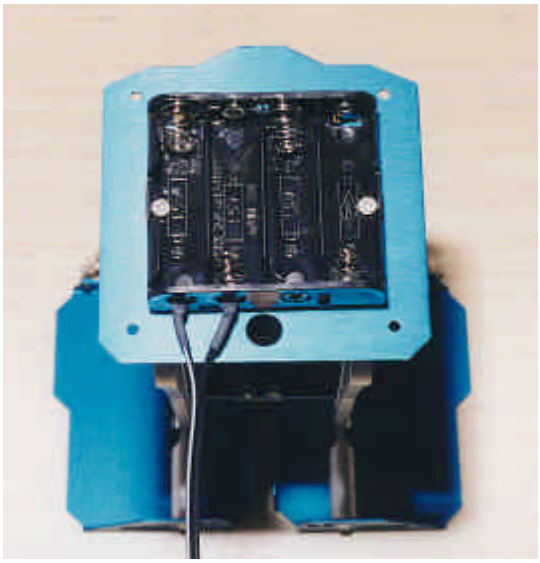
Step #17: Install Battery Pack

Parts required:

- Battery box
- (2) 4/40 3/8" long flathead countersunk machine screws
- (2) 4/40 nuts

Stand the Toddler up on its feet.

Install the plastic battery pack using two 4/40 3/8" flathead screws and nuts. The flathead screws will be countersunk into the battery pack when tightened. The screws should be out of the way of the batteries.

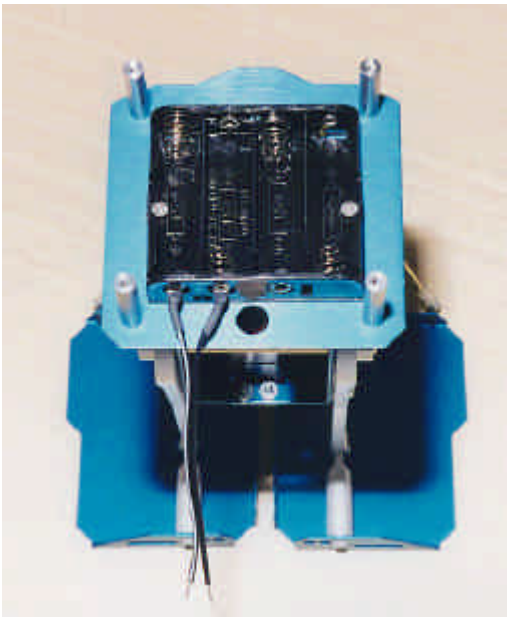


Step #18: Install Standoffs

Parts required:

- (4) 1" female/female 4/40 standoffs
- (4) 4/40 ¼" panhead machine screws

Using four 4/40 ¼" screws install the four 1" standoffs on the top plate.

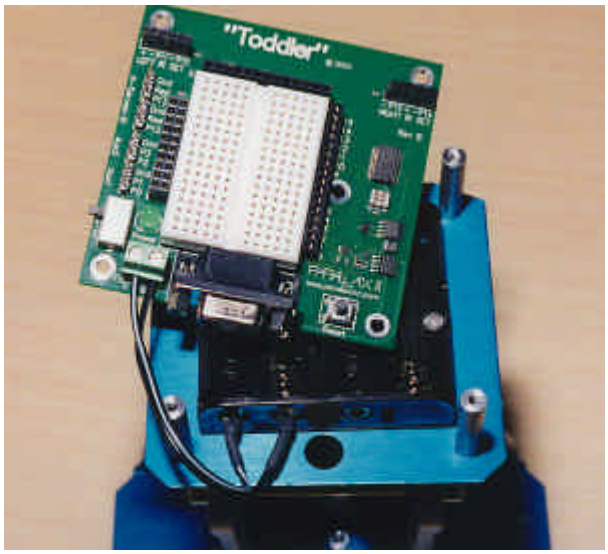


Step #19: Connect Toddler Board to Battery Pack

Parts required:

- Toddler printed circuit board

The battery pack's white lead connects to the Toddler board's + terminal. The other lead connects to the – terminal. Using a screwdriver secure both wires.

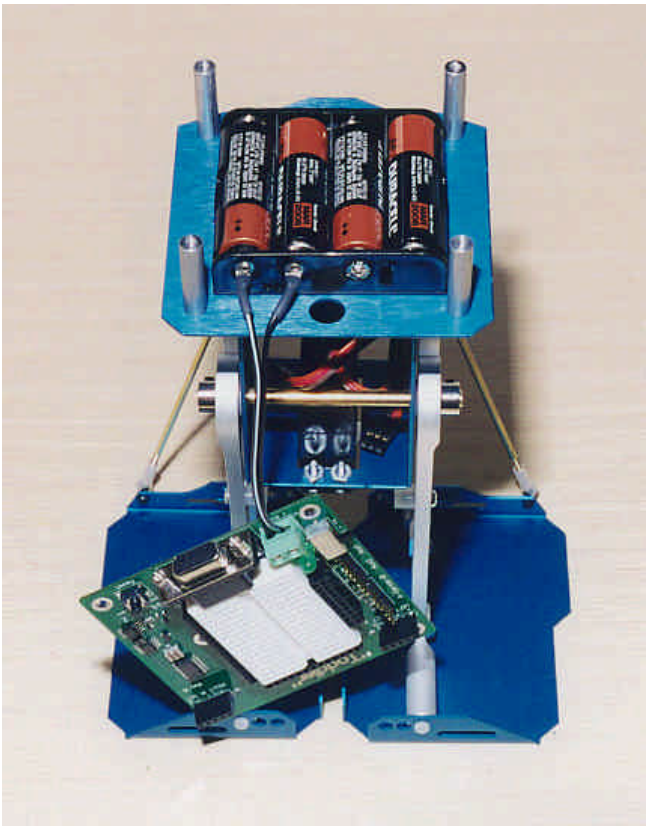


Step #20: Install Batteries

Parts required (but not included in the Toddler kit):

- (4) AA batteries

Install 4 AA batteries. Slide the Toddler's switch to Position 1 to verify that power is properly connected. The green power light will turn on.



Step #21: Mount Toddler Board

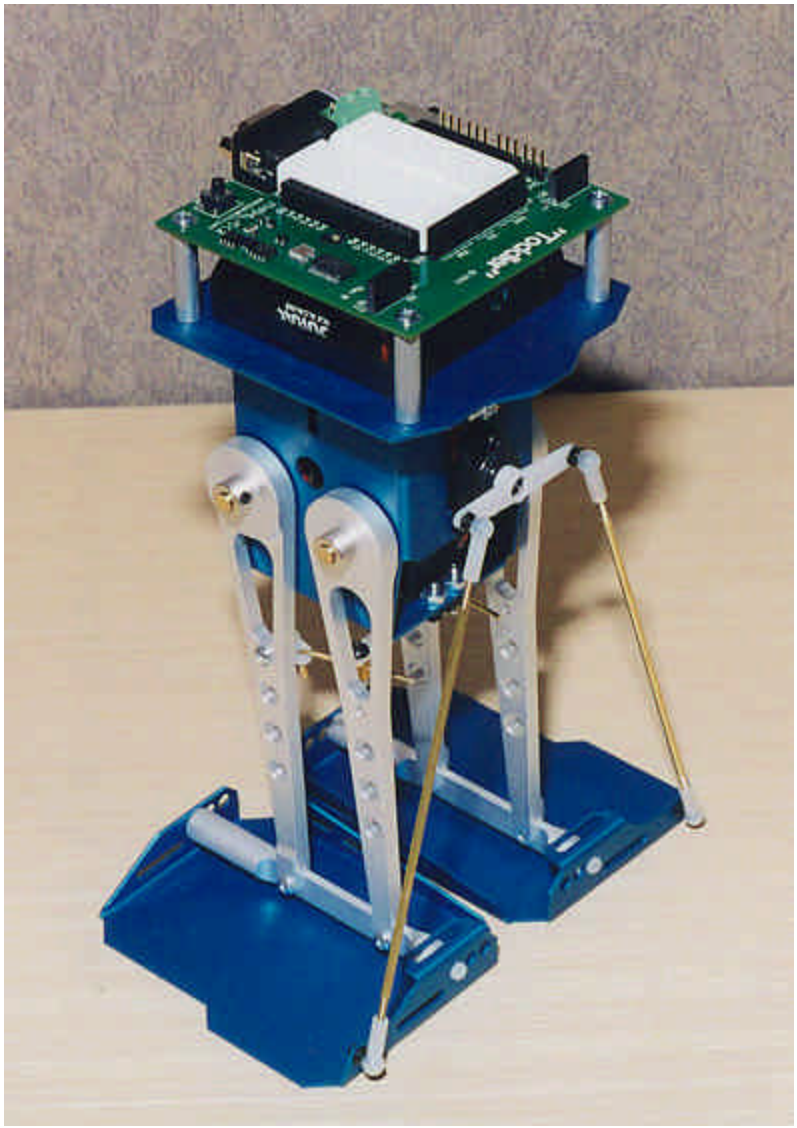
Parts required:

- (4) 4/40 ¼" panhead machine screws

Using four 4/40 ¼" panhead machine screws mount the Toddler board on the top of the standoffs.

Connect the bottom servo (stride) to P12 on the Toddler board. Connect the front servo (tilt) to P13 on the Toddler board.

You're ready to rock!



As a final step, repeat Step #3 to center the servos. The Toddler should stand flat on the ground with both feet aligned. When you pick the robot up, the feet outsteps may be slightly tilted downward. Adjustments can be made to the ball joints. The easiest way to remove the ball joint from the socket is to carefully pry it with the screwdriver.

Challenges

Experiment #1 has no challenges. Plenty of them lie ahead!

Experiment #2: Taking your First Steps

Making the Toddler walk requires some patience – the Toddler has more than 30 different individual movements. In this experiment you'll learn how to make the robot walk forward and backward by writing several routines. In Experiment #2 examples the movements are not blended – first you tilt then you stride. This is different than the way you walk.

After forward and backward movements are mastered we'll try making some turns. You'll see that linking movements requires attention to the previous step your Toddler took. For example, you can only move your left leg forward if it is off the ground.

When the basics are mastered, you'll learn to store movements and sub-movements in EEPROM and write more efficient code. All of this section is "open-loop" – there's no feedback to determine whether or not you have instructed your Toddler to lean to far left or right or even to look for obstacles.

Servo Control Basics

How a Servo Works

Normal (un-modified) hobby servos are very popular for controlling the steering systems in radio-controlled cars, boats, and planes. These servos are designed to control the position of something such as a steering flap on a radio-controlled airplane. Their range of motion is typically 90° to 270°, and they are great for applications where inexpensive, accurate high-torque positioning motion is required. The position of these servos is controlled by an electronic signal called a pulse train, which you'll get some first hand experience with shortly. An un-modified hobby servo has built-in mechanical stoppers to prevent it from turning beyond its 90° or 270° range of motion. It also has internal mechanical linkages for position feedback so that the electronic circuit that controls the DC motor inside the servo knows where to turn to in response to a pulse train.

Time Measurements and Voltage Levels

Throughout this student guide, amounts of time will be referred to in units of seconds (s), milliseconds (ms), and microseconds (us). Seconds are abbreviated with the lower-case letter "s". So, one second is written as 1 s. Milliseconds are abbreviated as ms, and it means one one-thousandth of a second. One microsecond is one one-millionth of a second. Table 2.1 shows how Milliseconds and Microseconds equate in terms of both fractions and scientific notation.

Table 2.1: Milliseconds and Microseconds and Toddler PCB Voltage Labels

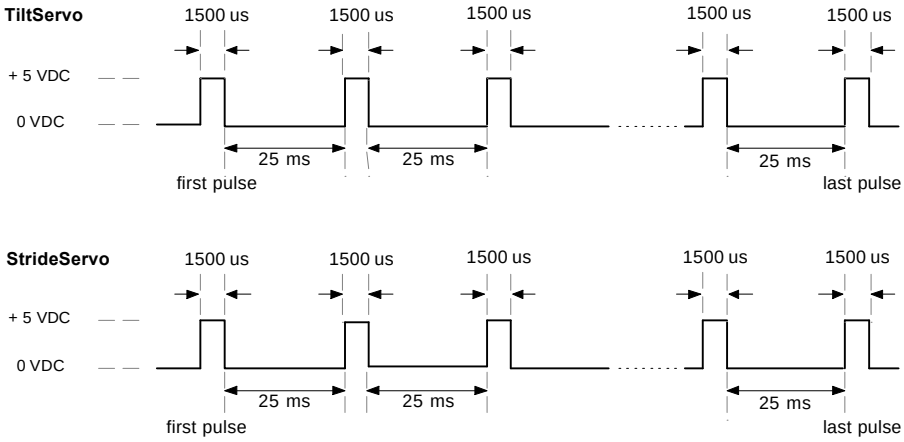
Milliseconds and Microseconds
$1\,\text{ms} = \frac{1}{1000}\,\text{s} = 1\times10^{-3}\,s$ $1\,m\text{s} = \frac{1}{1{,}000{,}000}\,\text{s} = 1\times10^{-6}\,s$
Voltages and Toddler PCB Labels
Vss = 0 V (ground) Vdd = 5 V (regulated) Vin = 6 V (unregulated)

A voltage level is measured in volts, which is abbreviated with an upper case V. The Toddler board has sockets labeled Vss, Vdd, and Vin. Vss is called the system ground or reference voltage. When the battery pack is plugged in, Vss is connected to its negative terminal. Vin is unregulated 6 V (from four AA batteries), and it is connected to the positive terminal of the battery pack. Vdd is regulated to 5 V by the Toddler's onboard voltage regulator, and it will be used with Vss to supply power to circuits built on the Toddler's breadboard.

Only use the Vdd sockets above the Toddler's breadboard for the Activities in this workbook. Do not use the Vdd on the 20-pin app-mod header.

The control signal the BASIC Stamp sends to the servo's control line is called a "pulse train," and an example of one is shown in Figure 2.1. The BASIC Stamp can be programmed to produce this waveform using any of its I/O pins. In this example, the BASIC Stamp sends a 1500 us pulse to P12 (stride servo) and P13 (tilt servo). When the pulse is done being executed the signal pin is low. Then, the BASIC Stamp sends a 25 ms pause.

Figure 2.1: Servo Pulse Train



This pulse train has a 1500 us high time and a 25 ms low time. The high time is the main ingredient for controlling a servo's motion, and it is most commonly referred to as the pulse width. Since these pulses go from low to high (0 V to 5 V) for a certain amount of time, they are called positive pulses. Negative pulses would involve a resting state that's high with pulses that drop low.

The ideal pause between servo pulses can be about 10-40 ms without adversely affecting the servo's performance.

The BASIC Stamp 2's PULSOUT command works in increments of 2 microseconds. For example, the following snippet of code makes a 1500 us pulse:

```
PULSOUT P13, 750     ' 1500 us pulse on pin 13
```

Many Ways to Move the Toddler

Programming is a cross between an art and science. There are usually many different ways a program can be written to get the same effect. Some are more efficient in program size and other are more efficient in performance.

In this chapter, we look at a number of different actions the Toddler can perform including walking forward and backward. The Toddler robot can make 36 different movements. These different movements must be linked together in order to walk. Each movement has a selection of possible precedent movements and a selection of possible follow-up movements.

We will also take a look at a number of different programs that perform these functions using different approaches. We present three approaches for programming the Toddler's movements. Most will prefer the last method. It uses more complicated programming techniques but it is more flexible and easier to use. Experienced programmers will want to jump right to the last approach but it is worth checking out all three.

Approach #1: Brute Force

This approach uses explicit subroutines for each movement. Calling these routines performs complex movements. It provides an obvious way of controlling the Toddler but enumerating all 36 movements consumes lots of precious program space. It also makes changes unwieldy. For example, implementing variable speed movements requires changes to all movement routines.

Approach #2: Data Tables

One obvious approach to consolidating 36 similar routines is to determine commonality within the routines and generating one or more that parametrically perform the same functions. Putting the parameters into data tables is one way to do this. Tables tend to be more concise in terms of construction compared to more explicit routines because the tables only contain parameters.

Approach #3: State Transitions

The DATA tables approach is really a consolidation of the first approach. The programmer must remember where the robot's feet are and call the appropriate routine or fill in the table with the proper parameters. The state transition approach is different because the Toddler keeps track of where its feet are. Transition actions are now used to move from one state to another. There are basically three tilt and three stride actions. That is significantly less than the 36 movements used in the other approaches.

Theory of Operation

Humans take to walking naturally but the actual act is extremely complex. It requires the coordinated actions of muscles and these actions are controlled by a very complex brain with a very sophisticated array of inputs from vision to touch. The Toddler is at the other end of the spectrum. It has only two control servos with a limited range of motion and essentially no feedback. Although the Toddler will not learn to walk of its own accord, it can be programmed relatively easily once you understand the basics.

Humans usually walk using a controlled fall. The body tilts slightly forward and a leg is moved in front to stop the fall. The process is repeated as the person proceeds to move forward. The effect is more noticeable when a person is running. It is easy to see why a person falls on the ground if they misstep.

It is possible for humans to shuffle along like the Toddler. In this case the foot is placed in front and the body is pulled along but it is hard to do. Try it. Tilting your body forward makes it easier but this is essentially a controlled fall.

The Toddler can fall over but its movement is done via shuffling and balance. This is necessary because of the limited range of movement. Essentially the Toddler can lean to either side or stand flat with both feet on the floor. Either the left or right foot can be in front, in which case the other is in the back, or they can be side-by-side. There are essentially 9 basic foot orientations and there are 4 transitions from each orientation to another valid orientation for a total of 36 transitions or movements.

Even with this limited range of actions, the Toddler can move about a flat surface with relative ease. Its restrictions do limit the Toddler to two basic kinds of movements though: walking in a straight line and pivoting. Still, this can get the Toddler from point A to point B.

Walking is essentially a four phase process.

1. Tilt to one side
2. Move the leg that is not on the ground
3. Tilt to the other side
4. Move the leg that is not on the ground

This process essentially takes one step. The direction the leg moves controls the direction and the distance traveled is controlled by how far the legs are moved. The speed of the Toddler depends on how fast the actions are performed and how far the legs move.

Assuming the Toddler is not programmed to tilt too far in one direction, it will remain balanced. This means the process can be stopped and restarted later at any point. This differs from humans in a controlled fall because the foot must be there to stop the fall.

Momentum plays a key part in writing programs to control the Toddler. The servo motors provide precise leg position control. They can move the legs slowly or quickly and can stop them at any location along the way. The Toddler also remains balanced even when tilted far to one side but this limit is difficult to attain when moving quickly because of momentum. Speed up the leg movement and the amount of momentum the leg has increases. Trying to stop it at the balance limit is only possible if the leg has little momentum. Too much momentum at that point and the Toddler falls over.

For turns the Toddler can only pivot. It does not have a knee or hip joint like a human. The Toddler's feet always face forward so it cannot turn its feet to change direction. This does not restrict the Toddle to straight line motion though. By pivoting, the Toddler can move from Point A to Point B in a straight line, pivot in the direction of Point C and then walk in that direction to Point C.

Pivoting is also a four-phase process.

1. Tilt to one side
2. Move the leg that is not on the ground
3. Put both feet on the ground
4. Move the legs opposite of each other

This process works because of friction. The actual pivot occurs in the last phase where both feet are on the ground. Essentially one leg pulls the Toddler forward while the other pulls it backward. This causes the Toddler to pivot. The amount of rotation is a factor of leg placement and the level of friction between the Toddler's feet and the surface it is on. Low friction results in minimal pivoting. Too much friction and the Toddler can fall over.

The Toddler is essentially restricted to flat surfaces. The type of surface affects the amount of friction between the Toddler and the surface. Wood, hard carpet and kitchen floors are a good surface while ice and rubber are not. Dense carpet works well. Putting different surfaces on the bottom of the Toddler's feet can increase the level of friction. This is typically done using tape (electrical, maybe even a small piece of grip tape). There are not hard rules on choosing surfaces and increasing friction so experiment. You may have to adjust the program controlling to the Toddler to take into account the surface. For example, you cannot assume that two pivot actions will turn the Toddler 90 degrees.

This brings up the issue of accuracy. The Toddler does a good job of moving but it is relatively inaccurate in its movements compared to its wheeled cousin, the Boe-Bot. If the Toddler goes six steps forward and six steps back it will not wind up in exactly the same spot. It may be close but it is unlikely to be exact. Add some turns and all bets are off. Getting the Toddler to walk in a square is next to impossible. It is easy to program the commands to walk in a square but due to friction and the mechanical accuracy of the Toddler's movements, the Toddler will probably not walk over its own footsteps.

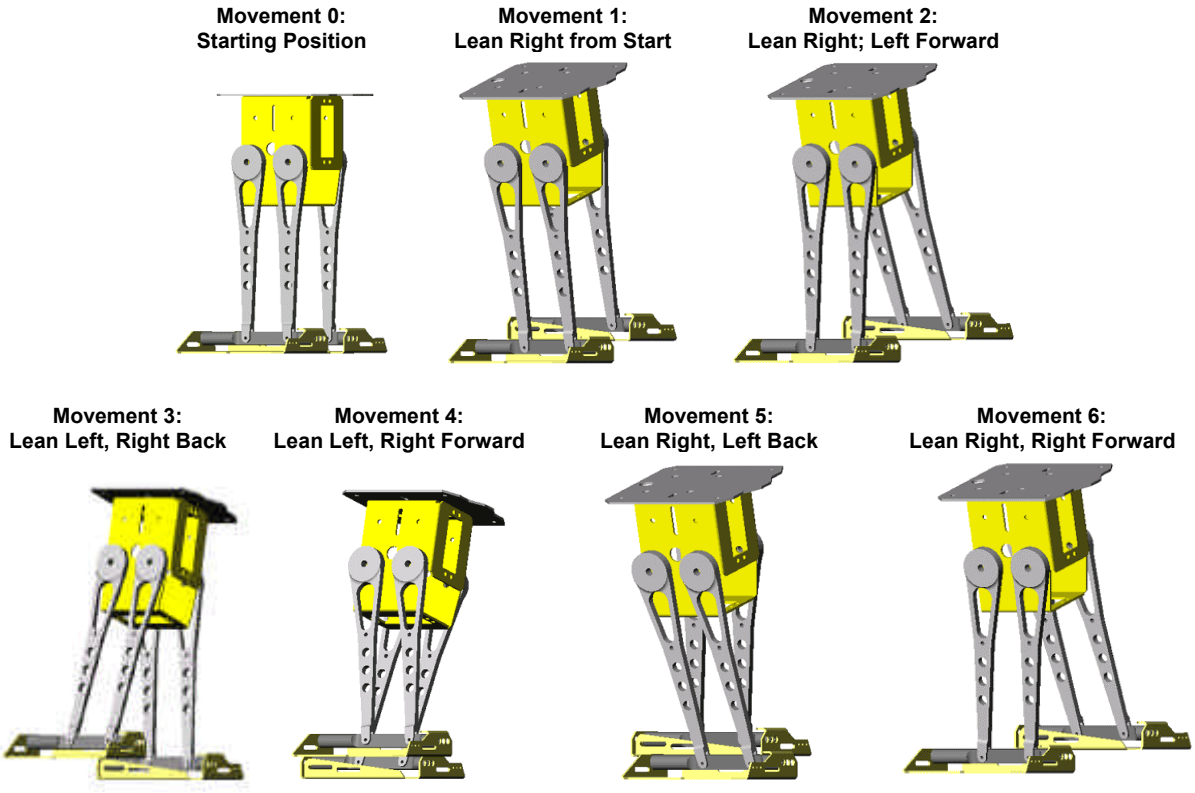
For most experiments, accuracy is not an issue. It is possible to track the Toddler's orientation using the optional Compass AppMod but tracking distance moved is a daunting task at best. These problems are beyond the scope of this book but great areas for investigation. Answering the question "where is my robot?" is one of the most challenging hobby and educational endeavors, whether it uses wheels or legs.

The lack of articulated legs prevents the Toddler from walking over obstacles. The Toddler cannot handle a grade of any significance so stay away from ramps. The Toddler can avoid obstacles by going around them. In later experiments, we examine obstacle detection using infrared devices included with the Toddler.

Activity #1: Basic Walking Movements: Approach 1

The Toddler is a walking robot so getting it moving is a good starting point. Figure 2.2 shows a possible order of operation for taking a few steps.

Figure 2.2: First Steps



Once the Toddler has walked Movements 0, 1 and 2 the process of Movements 3, 4, 5 and 6 can repeat to walk in a straight line. The code to perform the first three movements is shown in the next three pages. Movement 1 and 2 are almost identical except that one adjusts the tilt and the other the stride. Movements 1, 3 and 5 would use the same code with different values. The same is true for movements 2, 4 and 6.

The M0 routine is unique. It is designed to place the feet of the Toddler on the ground and next to each other regardless of where feet are when the routine starts. This can result in a jerking motion if the legs are not already in or close to this point.

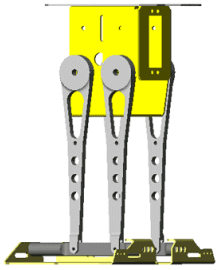
The M1 and M2 routines are representative examples of all the other movement routines in the program. The M1 movement tilts the Toddler to the right. To do so, it sends pulses to both servos. It sends the same pulse width to the stride motor so it remains stationary and the feet do not move forward or backwards. The tilt servo is sent pulses that progressively change in width causing the tilt servo to rotate which in turn causes the Toddler to lean.

The M2 routine does the same thing but in this case the tilt servo is held stationary while the stride servo is driven by a set of different width pulses causing one leg to move ahead of the other.

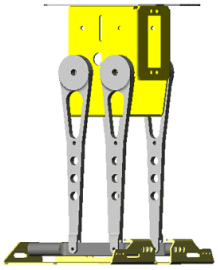
The program uses constant named definitions for the range of servo pulse width limit values while the examples with small snippets of code use numeric constant values. The effect is the same but the named constants minimize changes to the program when experimenting with different values.

Figure 2.3: Movement 0 Example (M0)

Starting Position



Ending Position



Movement 0 BASIC Stamp Code

```
M0:
 FOR Pulses = 1 TO 100 STEP 5
   PULSOUT TiltServo, 750
   PULSOUT StrideServo, 750
   PAUSE 25
 NEXT
```

Movement 0 Timing Diagram

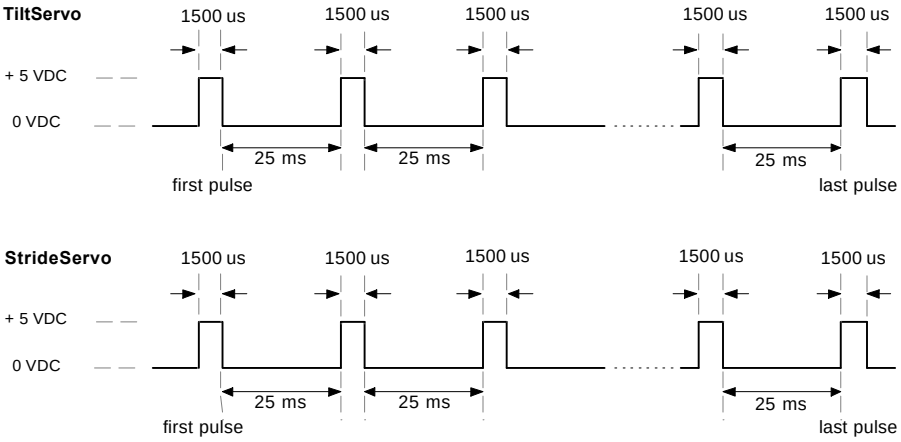
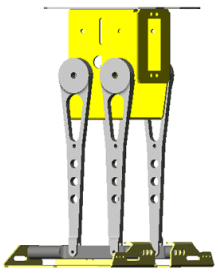
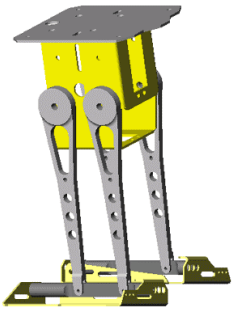


Figure 2.4: Movement 1 Example (M1)

Starting Position



Ending Position



Movement 1 BASIC Stamp Code

```
M1:
  FOR Pulses = 750 TO 620 STEP 5
    PULSOUT TiltServo, Pulses
    PULSOUT StrideServo, 750
    PAUSE 25
  NEXT
```

Movement 1 Timing Diagram

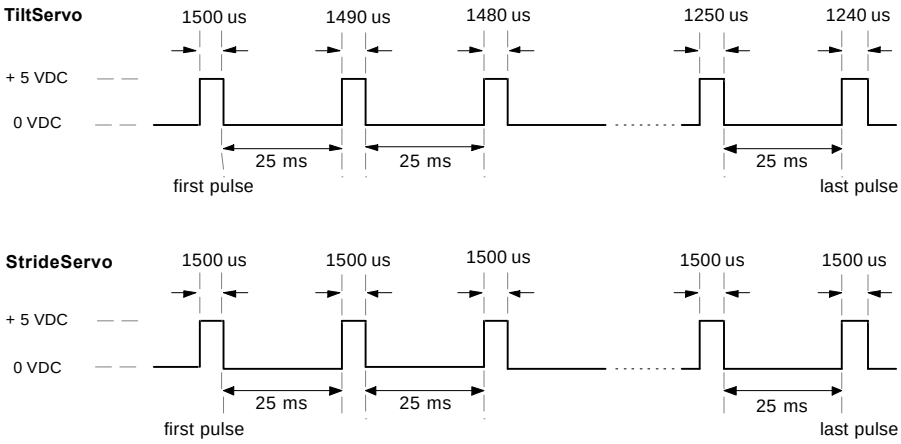
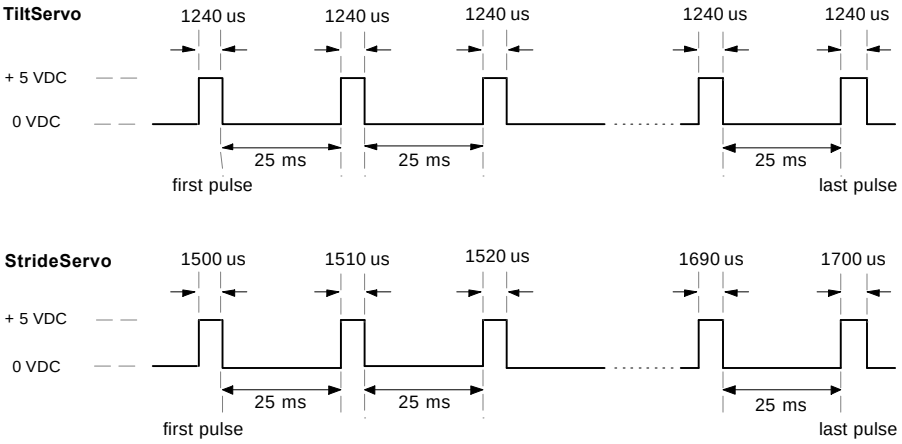


Figure 2.5: Movement 2 Example (M2)

Movement 2 BASIC Stamp Code

```
M2:
  FOR Pulses = 750 TO 850 STEP 5
    PULSOUT TiltServo, 620
    PULSOUT StrideServo, Pulses
    PAUSE 25
  NEXT
```

Movement 2 Timing Diagram



An example program that performs all the movements for walking forward is shown is shown below. It is possible to adjust the different constants in the program to make the robot walk faster or take bigger steps. Be careful of large changes because the Toddler can fall over. More on that later.

This program also cleans up its movement so both feet are centered and flat on the floor when it is done. This makes starting other programs easier since the feet are in a known position.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 2.1: First Steps Forward
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------------

MoveDelay             CON           25             ' in micrcoseconds

TiltStep              CON           5              ' TiltServo step size

RightTilt             CON           620            ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           5              ' StrideServo step size

RightForward          CON           650            ' Stride limits
StrideCenter          CON           750
LeftForward           CON           850


' -----[ Variables ]-------------------------------------------------------------

MoveLoop              VAR           Nib            ' Loop for repeat movements
Pulses                VAR           Word           ' Pulse variable


' -----[ Main Code ]-------------------------------------------------------------
'
' Take three full steps.

Main_Program:
  GOSUB M0                                         ' center servos
  GOSUB M1                                         ' tilt right
  GOSUB M2                                         ' step left
```

```
    FOR MoveLoop = 1 to 3
      GOSUB M3                                          ' tilt left
      GOSUB M4                                          ' step right
      GOSUB M5                                          ' tilt right
      GOSUB M6                                          ' step left
    NEXT

  GOSUB M3                                              ' tilt left
  GOSUB M7                                              ' center feet
  GOSUB M8                                              ' center servos
END


' -----[ Subroutines ]------------------------------------------------------

M0:
  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
  RETURN

M1:
  FOR Pulses = CenterTilt TO RightTilt STEP TiltStep
    PULSOUT TiltServo, Pulses
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
  RETURN

M2:
  FOR Pulses = StrideCenter TO LeftForward STEP StrideStep
    PULSOUT TiltServo, RightTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN

M3:
  FOR Pulses = RightTilt TO LeftTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, LeftForward
    PAUSE MoveDelay
  NEXT
  RETURN

M4:
  FOR Pulses = LeftForward TO RightForward STEP StrideStep
    PULSOUT TiltServo,LeftTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
```

```
  NEXT
  RETURN

M5:
  FOR Pulses = LeftTilt TO RightTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, RightForward
    PAUSE MoveDelay
  NEXT
  RETURN

M6:
  FOR Pulses = RightForward TO LeftForward STEP StrideStep
    PULSOUT TiltServo,RightTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN


M7:
 FOR Pulses = LeftForward TO StrideCenter STEP StrideStep
    PULSOUT TiltServo,LeftTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN

M8:
  FOR Pulses = LeftTilt TO CenterTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
RETURN
```

Note that the program is downloaded to the Toddler using the serial cable with the power switch in either the download (1) or run (2) position. The cable can remain connected to the Toddler while it is walking if there is sufficient length to allow it to move freely about the PC. You may want to hold the cable near the Toddler as it does make it slight less stable. The power on the Toddler should be turned off when disconnecting the cable. The Toddler can then walk on its own when the power is turned on since the program is downloaded into non-volatile memory. You can also program Toddler with the switch in Position 1, hold the reset button and move it to Position 2 so it will walk.

Three constants could be modified to make it walk quicker: `MoveDelay` and `TiltStep` and `StrideStep`. Decreasing `MoveDelay` means there will be less pause between the servo pulses. Increasing `TiltStep` and `StrideStep` means the servo pulse width changes will be larger (making for fewer pulses to complete the step).

If the Toddler isn't starting with both feet firmly planted squarely on the ground, or if you would like to experiment with larger step distances you could modify the `CenterTilt` and `StrideCenter` values. This would result in a need to also modify the right and left limits for both the tilt and stride.

These parameters are from the following code snippet in Toddler Program 2.1.

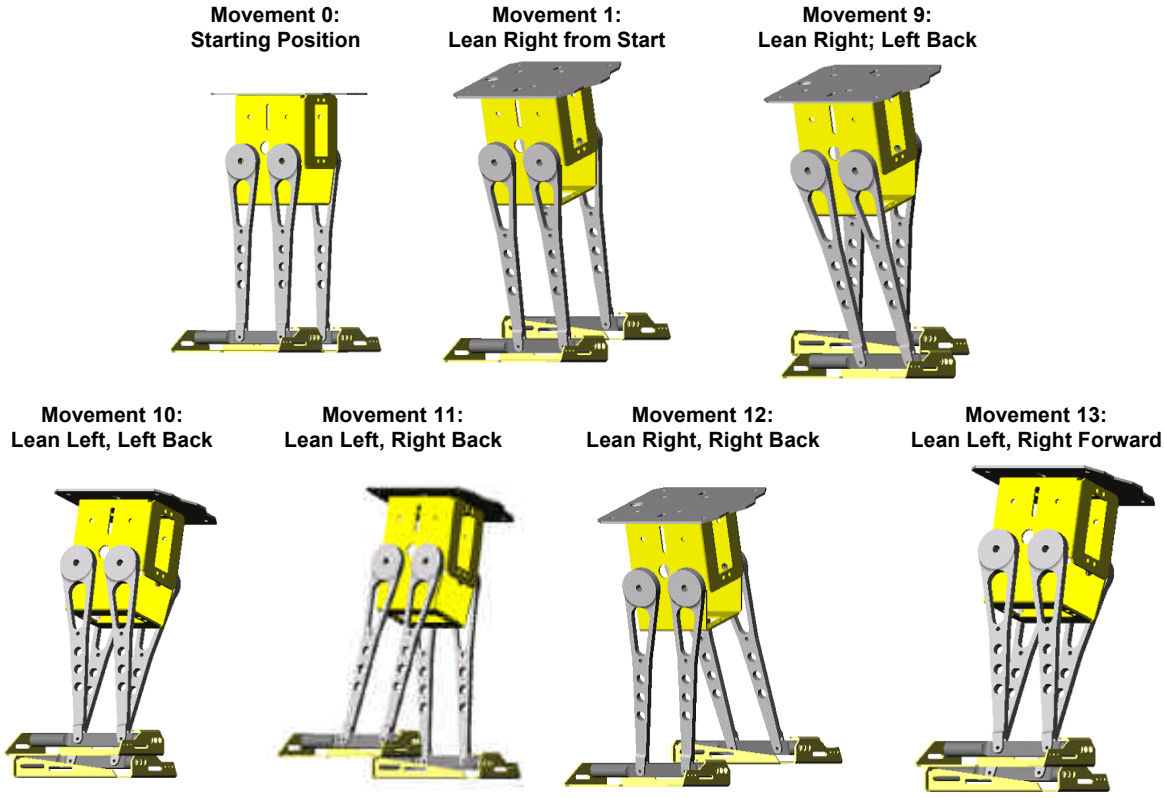
```
RightTilt             CON           620           ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           5             ' StrideServo step size

RightForward          CON           650           ' Stride limits
StrideCenter          CON           750
LeftForward           CON           850
```

Activity #2: Walking Backwards: Approach 1

The Toddler robot can walk backward as well as forward but it is not simply a matter of using the steps in the prior program in reverse order. The Toddler moves in the reverse fashion but the functions necessary to do this will be different. With just over half a dozen routines, the last sample program is relatively simple. Changing it to handling a different direction is not too difficult. Keep in mind that the two other approaches to performing these tasks are presented later in this chapter.

In the prior program, the subroutine for each step was numbered sequentially. In this program, the steps will be slightly different so we can use different routine names. The starting movement is the same as the prior program but the second step will be Movement 9 that matches the M9 routine.

Figure 2.6: First Steps – In Reverse



As with the forward walking program, the Toddler starts with Movements 0, 1 and 9. The process of Movements 10, 11, 12 and 13 can repeat to walk in a straight line but backwards. An example program is shown is shown below. Adjust the different constants in the program to make your robot walk faster or take bigger steps. The program also cleans up its movement so both feet are centered and flat on the floor.

Note that the routines with the same name have been extracted from the first sample program. A program that used this approach but required more sophisticated actions would need more routines with the potential of requiring all 36.

```
' -----[ Title ]-----------------------------------------------------------
' Toddler Program 2.2: First Steps Backward
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------

MoveDelay             CON           25             ' in micrcoseconds

TiltStep              CON           5              ' TiltServo step size

RightTilt             CON           620            ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           5              ' StrideServo step size

RightForward          CON           650            ' Stride limits
StrideCenter          CON           750
LeftForward           CON           850


' -----[ Variables ]-------------------------------------------------------

MoveLoop              VAR           Nib            ' Loop for repeat movements
Pulses                VAR           Word           ' Pulse variable


' -----[ Main Code ]-------------------------------------------------------
'
' Take three full steps.
```

```
Main_Program:
  GOSUB M0                                          ' center servos
  GOSUB M1                                          ' tilt right
  GOSUB M9                                          ' step right

    FOR MoveLoop = 1 to 3
      GOSUB M10                                     ' tilt left
      GOSUB M11                                     ' step left
      GOSUB M12                                     ' tilt right
      GOSUB M13                                     ' step right
    NEXT

  GOSUB M10                                         ' tilt left
  GOSUB M14                                         ' center feet
  GOSUB M8                                          ' center servos
END


' -----[ Subroutines ]------------------------------------------------------

M0:
  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
  RETURN

M1:
  FOR Pulses = CenterTilt TO RightTilt STEP TiltStep
    PULSOUT TiltServo, Pulses
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
  RETURN

M8:
  FOR Pulses = LeftTilt TO CenterTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT
  RETURN

M9:
  FOR Pulses = StrideCenter TO RightForward STEP StrideStep
    PULSOUT TiltServo, RightTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN
```

```
M10:
  FOR Pulses =  RightTilt TO LeftTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, RightForward
    PAUSE MoveDelay
  NEXT
  RETURN

M11:
  FOR Pulses = RightForward TO LeftForward STEP StrideStep
    PULSOUT TiltServo,LeftTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN

M12:
  FOR Pulses = LeftTilt TO RightTilt STEP TiltStep
    PULSOUT TiltServo,Pulses
    PULSOUT StrideServo, LeftForward
    PAUSE MoveDelay
  NEXT
  RETURN

M13:
  FOR Pulses = LeftForward TO RightForward STEP StrideStep
    PULSOUT TiltServo,RightTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN

M14:
  FOR Pulses = RightForward TO StrideCenter STEP StrideStep
    PULSOUT TiltServo,LeftTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  RETURN
```

Activity #3: Using a DATA Table to Store Movements: Approach 2

The length of the two prior programs is similar but more complex programs could grow larger simply because more movement routines would be necessary. Moving information in tables is one way of simplifying the programming task. This next sample application employs both fixed size and variable length tables.

The fixed size table is used to store the information about each movement. The variable length tables are used to store a sequence of movements. This allows complex movements to be of arbitrary length. The tables are accessed using the PBASIC READ command. See the BASIC Stamp Manual for this command's syntax.

The fixed size table contains four byte entries.[3] The first byte is the starting tilt value and the second byte is the ending tilt value. The third and fourth bytes are the starting and ending stride values. The READ command can only access byte values (i.e., one "letter" at a time) but the servo pulse width values are greater than 255 (we're using values between 600 and 900), the limit of an unsigned byte. This is why the values are divided by 10 before storing them in the table and when they must be multiplied by 10 before they are used. This approach is more complex to execute but it cuts the table size in half. Named constants simplify the table construction.

The `Movement` routine assumes that each entry is used to tilt or move the Toddler. It checks the first two bytes to see if they are the same and handles the entry accordingly.

Only eight (8) movements are employed in this program but it is easy to see how all 36 movements could be easily added to the table.

The fixed table is actually a two dimensional entity. The first dimension is the four bytes for each entry. The second is each movement entry. The indexing for this second dimension is handled using a LOOKUP statement. Since the table entries are a fixed size, it is possible to index the array numerically using a statement like:

```
READ M1 + (( Dx – 1 ) * 4 ) , T1
```

Using the LOOKUP statement does provide some flexibility if the order of the array indexing is changed in the future or if the size of the entry changes.

The variable length tables store the sequence of movement numbers for a particular action. The index of one of these tables is passed to the single movement routine. It is possible to repeat a sequence of actions using one of these tables but a FOR . . . NEXT loop is used for controlling each step since the loop limit value is easy to change. This keeps the program from increasing in size as functional requirements change.

[3] If you are interested in learning about the difference between bits, nibbles, bytes and words see the Basic Analog and Digital text.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 2.3: First Steps Forward Using Tables
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]--------------------------------------------------------

MoveDelay             CON           25             ' in micrcoseconds

TiltStep              CON           10             ' TiltServo step size

RightTilt             CON           620            ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10             ' StrideServo step size

RightForward          CON           650            ' Stride limits
StrideCenter          CON           750
LeftForward           CON           850


' -----[ Variables ]--------------------------------------------------------

MoveLoop              VAR           Nib            ' Loop for repeat movements
Pulses                VAR           Word           ' Pulse variable

Dx                    VAR           Pulses
Mx                    VAR           Word
T1                    VAR           Byte
T2                    VAR           Byte
S1                    VAR           Byte
S2                    VAR           Byte


' -----[ Main Code ]--------------------------------------------------------
'
' Take three full steps.
'
' The following state tables are lists of movement state numbers.
' A zero indicates the end of a list.
' These are used with the Movement routine.

StartForward          DATA          1, 2, 0
WalkForward           DATA          3, 4, 5, 6, 0
FinishForward         DATA          3, 7, 8, 0
```

```
Main_Program:
  GOSUB M0                    ' center servos

  Mx = StartForward
  GOSUB Movement

    FOR MoveLoop = 1 to 3
      Mx = WalkForward
      GOSUB Movement
    NEXT

  Mx = FinishForward
  GOSUB Movement              ' tilt left
END


' -----[ Subroutines ]------------------------------------------------------
'
' ----- Constants and tables for Movement routine  -----
'
' Note: DATA is stored as bytes so the value must be less than 256.
'       Dividing values by 10 keeps the values within this range.

CT              CON     CenterTilt/10
RT              CON     RightTilt/10
LT              CON     LeftTilt/10

SC              CON     StrideCenter/10
LF              CON     LeftForward/10
RF              CON     RightForward/10


M1              DATA    CT, RT, SC, SC
M2              DATA    RT, RT, SC, LF
M3              DATA    RT, LT, LF, LF
M4              DATA    LT, LT, LF, RF
M5              DATA    LT, RT, RF, RF
M6              DATA    RT, RT, RF, LF
M7              DATA    LT, LT, LF, SC
M8              DATA    LT, CT, SC, SC



' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = table index, table ends in 0

Movement:
  READ Mx, Dx                 ' read state table number
  Mx = Mx + 1
```

```
  IF Dx = 0 THEN DoReturn     ' skip if no more states

  LOOKUP Dx,[M1, M1, M2, M3, M4, M5, M6, M7, M8],Dx

  READ Dx,   T1               ' read table entry
  READ Dx+1, T2
  READ Dx+2, S1
  READ Dx+3, S2

  IF T1 = T2 THEN MovementStride

  FOR Pulses = T1*10 TO T2*10 STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, S1*10
    PAUSE MoveDelay
  NEXT
  GOTO Movement

MovementStride:
  FOR Pulses = S1*10 TO S2*10 STEP StrideStep
    PULSOUT TiltServo,   T1*10
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT
  GOTO Movement


' ----- M0: Move feet to initial center position -----

M0:
  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, StrideCenter
    PAUSE MoveDelay
  NEXT

DoReturn:
  RETURN
```

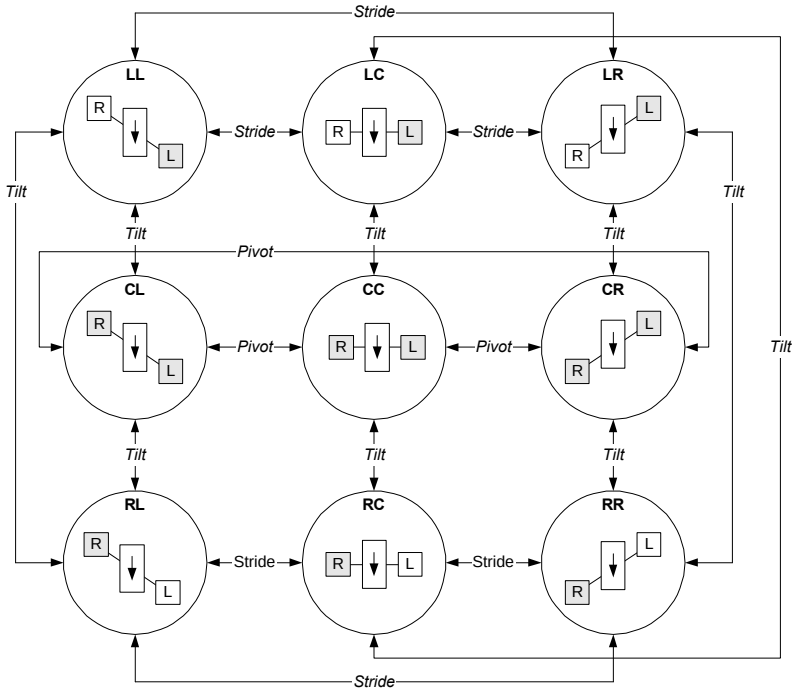
Adding support for walking backward is easier with this program. Three things need to be changed. First, the extra movement entries must be added to the fixed size table and the LOOKUP command. Second, a variable length table must be added for stepping through a backward foot-step. Finally, the table name and a call to the movement routine must be added. This is significantly better than adding more routines to the program.

Activity #4: Using State Transitions For Movements: Approach 3

Now we take a look at a completely different way of handling movement control. The prior two approaches essentially required the programming to string together a series of subroutine calls in the proper order. In those cases, the programmer knew what the prior state was and would only use movement routines that were relevant. It was relatively simple for the basic actions examined thus far but it gets very tedious as the actions become more complex.

The state transition approach presented here changes the kinds of routines used from ones that have a known starting and ending state to routines that indicate a desired ending state. The program keeps track of the current state and adjusts accordingly. The reason that this approach simplifies things is that there are really only two kinds of actions, tilt and stride, that the Toddler can perform and only three logical orientations with each of these for a total of nine states. The following diagram shows these states.

Figure 2.7: State Transition Approach



Each circle represents a state. The center shows the orientation of the Toddler's legs and body. The arrow indicates the front of the Toddler. The dark background in a rectangle indicates that a leg is on the floor. The white background indicates the leg is in the air. The two letters at the top of the circle provide a representation for the Toddler positioning. The first letter indicates the tilt (T) and the second indicates stride (S) or what foot is in front. The letters can be L, C, or R. C indicates the respect servo motor is centered. For the tilt, C indicates both feet are on the ground. For stride, C indicates that both feet are inline along the center of the body's axis. The reason for naming each state will become apparent soon.

There are also bi-directional arrows showing valid transitions from one state to another. The label on the line indicates the type of change that occurs. The Pivot designation is the same as Stride but we make the distinction because this will cause the Toddler to pivot.

Nine states, each with four possible transitions yields 36 distinct transitions. If the transition routines are labeled using their starting and ending state names with the format TSxTS then movement M0 is CCxRC as it starts with both legs together in state CC and ends up leaning to the right in state RC. Using this naming convention makes programming easier in the prior examples with the code looking like this.

```
' -----[ Main Code ]-----------------------------------------------------------
'
' Take three full steps.

Main_Program:
  GOSUB M0             ' center servos

  GOSUB CCxRC          ' tilt right
  GOSUB RCxRL          ' step left

    FOR MoveLoop = 1 to 3
      GOSUB RLxLL      ' tilt left
      GOSUB LRxLR      ' step right
      GOSUB LRxRR      ' tilt right
      GOSUB RRxRL      ' step left
    NEXT

  GOSUB RLxLL          ' tilt left
  GOSUB LLxLC          ' center feet
  GOSUB LCxCC          ' center servos
END
```

The advantage of the name change is obvious. The last two letters of the prior routine are the starting two letters of the next routine.

Of course, this is still relatively cryptic and without the comments the actions would definitely be confusing. The problem is that this approach requires knowledge of the prior state.

Now take a look at the next sample code snippet. It does away with the comments (not always a good idea but fine for this discussion) and the routines being called indicate what action is being performed.

```
' -----[ Main Code ]------------------------------------------------------------
'
' Take three full steps.

Main_Program:
  GOSUB M0             ' center servos

  GOSUB TiltRight
  GOSUB StideLeft

    FOR MoveLoop = 1 to 3
      GOSUB TiltLeft
      GOSUB StrideRight
      GOSUB TiltRight
      GOSUB StrideLeft
    NEXT

  GOSUB TiltLeft
  GOSUB StrideCenter
  GOSUB TiltCenter
END
```

So what happened? You need to take a look at the next program listing to see how these routines were implemented but essentially two variables, `CurrentTilt` and `CurrentStride`, were added to keep track of where the feet are. The routine simply applies the designated change.

There are only six routines that need be used. Any can be applied in any order although only four will cause a change at any time.

Why? Because two of the six will cause the Toddler to stay in the state that it is currently in. For example, if the Toddler is tilting to the left then calling the `TiltLeft` routine will leave it in the same position. There will be a delay while it executes the routine but it turns out to be a very short period of time. It is not noticeable when watching the Toddler move.

The following program implements the routines used in the prior code snippet but it retains some of the ideas employed in the last full program listing (Program 2.3) that used tables. They are still worthwhile although they are used in a slight different fashion here.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 2.4: First Steps Forward
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]--------------------------------------------------------

MoveDelay             CON           25             ' in micrcoseconds

TiltStep              CON           10             ' TiltServo step size

RightTilt             CON           620            ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10             ' StrideServo step size

RightStride           CON           650            ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]--------------------------------------------------------

MoveLoop              VAR           Nib            ' Loop for repeat movements
Pulses                VAR           Word           ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses
Mx                    VAR           Word


' -----[ Main Code ]-------------------------------------------------------------
'
' Take three full steps.
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL             CON     0
```

```
TC              CON     1
TR              CON     2

SL              CON     3
SC              CON     4
SR              CON     5

xx              CON     255


WalkForward             DATA            TR, SL, TL, SR, xx
WalkBackward            DATA            TR, SR, TL, SL, xx
TurnLeft                DATA            TL, SR, TC, SL, xx
FinishForward           DATA            TR, SC, TC, xx


Main_Program:
  GOSUB ResetCC

    FOR MoveLoop = 1 to 3
      Mx = WalkForward
      GOSUB Movement
    NEXT

    FOR MoveLoop = 1 to 3
      Mx = TurnLeft
      GOSUB Movement
    NEXT

    FOR MoveLoop = 1 to 3
      Mx = WalkBackward
      GOSUB Movement
    NEXT

  Mx = FinishForward
  GOSUB Movement
END


' -----[ Subroutines ]------------------------------------------------------------

' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = table index, table ends in xx

Movement:
  READ Mx, Dx                       ' read next action
  Mx = Mx + 1

  IF Dx = xx THEN MovementDone      ' skip if end of list

  GOSUB DoMovement                  ' execute movement
  GOTO Movement                     ' loop until done
```

```
DoMovement:
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                      ' will fall through if invalid index
MovementDone:
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN
```

```
' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
  RETURN
```

Ok, we cheated. There is a little turning done in the program and that will be covered in more detail in the next experiment. The reason for including it here is that the pivot action is not explicitly done by a routine. Instead the general movement routine handles all actions. The pivot action is defined within the `TurnLeft` table.

The variable table handling is also changed to allow access to the six action functions like `TiltLeft`. Each is implemented as a routine that exits via a `RETURN` instruction. This is why the `DoMovement` routine is called via `GOSUB` rather than putting the `BRANCH` statement within a loop.

Also note the refinement with the `LOOKUP` statement. The named constant `xx` is used to terminate a variable length table. The value 255 is out of range for the values used and it allows the first value to be 0.

This approach will be the one used in subsequent programs in the book. Extending tables like `WalkForward` should be significantly easier since there are only six valid values. It is also possible to conserve space by using only 4 bits to store each value but this makes table definition and extraction very difficult due to limitations of PBASIC. Still, it is an option should a program become code-space constrained.

Challenges

1. Increase the walking speed and determine the maximum speed before the Toddler falls over.
2. Determine whether the Toddler operates the same on different surfaces such as carpet, wood and tile.
3. The Toddler can start moving its left or right foot first. Try changing the programs so that it moves the opposite foot first.
4. Have the Toddler perform a little dance using a more complex series of steps such as moving the foot in the air forward and backward a few times.

Experiment #3: Turning Around

The Toddler is a bit stiff. It can only move its feet forward and backward but it cannot turn its feet relative to it's body. This doesn't stop it from being able to turn. While walking in a straight line for the Toddler is somewhat similar to a person, turning is very different. The closest thing to turning like the Toddler for a person is trying to turn on ice with flat shoes.

The process of turning right on ice is relatively simple. Put your left foot forward and place it on the ground. Pull the left foot towards you. Pull it back and you pivot to the right. If the ice is wet and slippery then it may take quick a number of attempts to turn 90 degrees. Put the right foot forward to turn left.

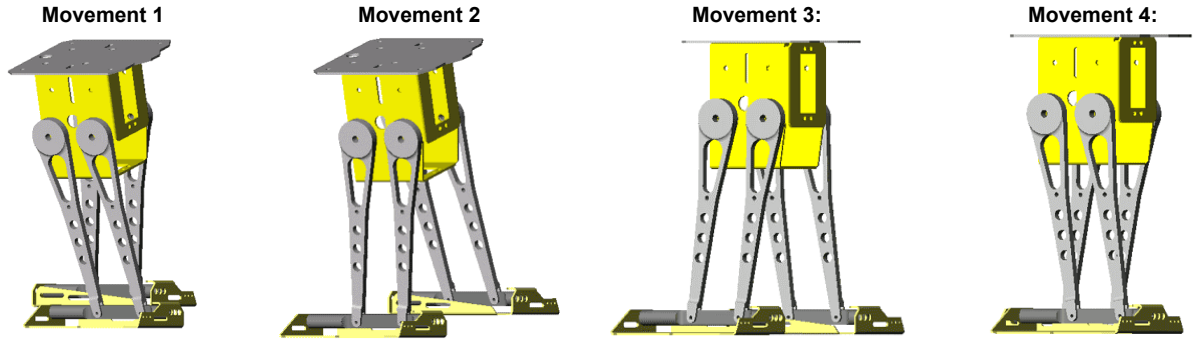
The standard Toddler does not do well on ice but it uses the same principle on other surfaces. The Toddler's feet are smooth metal that provides the slick surface. Turning works best when the surface the Toddler sits on provides some friction. If the surface is too slippery then it is possible to modify the Toddler's feet to provide more friction. This is typically done using a tape that has a rougher surface than the Toddler's metal feet. The entire foot need not be covered. There only needs to be enough coverage to add friction for the turn. The tape should not have an affect on straight-line movement.

Activity #1: Making a Turn

The Toddler turns by placing both feet flat on the ground and sliding them in opposite directions. Moving the feet in opposite directions is somewhat counterproductive because some of the forces are in opposite directions. The actual movement is more of a pivot than a turn.

The Toddler pivots only a small fraction of a circle at a time. It may take five to ten movements to turn 90 degrees (unless you put a small piece of grip tape on the feet). Twice that to turn around. The basic turning process is four movements like those shown below in Figure 3.1. A left turn works the same way except the tilt and leg movements are reversed.

Figure 3.1: Walking in a Circle (Right Hand Turns)



The first three movements place the left foot in front while the pivoting action occurs in the last movement. The following program performs this process multiple times. It performs both a right and a left turn. The key additions to Program 2.4 is the `TurnRight` table entry. The next section addresses two other entries that are presented in this program. These are `WideTurnLeft` and `PivotRight`.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 3.1: Turning
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------

TiltServo             CON           13            ' Tilt servo on P12
StrideServo           CON           12            ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------

MoveDelay             CON           25            ' in micrcoseconds

TiltStep              CON           10            ' TiltServo step size

RightTilt             CON           620           ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10            ' StrideServo step size

RightStride           CON           650           ' Stride limits
```

```
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]----------------------------------------------------------

MoveLoop              VAR           Nib           ' Loop for repeat movements
Pulses                VAR           Word          ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses
Mx                    VAR           Word


' -----[ Main Code ]-----------------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL            CON     0
TC            CON     1
TR            CON     2

SL            CON     3
SC            CON     4
SR            CON     5

xx            CON     255


WalkForward           DATA          TR, SL, TL, SR, xx
WalkBackward          DATA          TR, SR, TL, SL, xx

TurnLeft              DATA          TL, SR, TC, SL, xx
WideTurnLeft          DATA          TL, SR, TC, SL, TR, SL, TL, SR, xx

TurnRight             DATA          TR, SL, TC, SR, xx
PivotRight            DATA          TR, SL, TC, SR, TL, SL, TC, SR, xx

FinishForward         DATA          TR, SC, TC, xx


Main_Program:
  GOSUB ResetCC

    FOR MoveLoop = 1 to 5
      Mx = TurnRight
      GOSUB Movement
    NEXT
```

```
    FOR MoveLoop = 1 to 5
      Mx = TurnLeft
      GOSUB Movement
    NEXT

    FOR MoveLoop = 1 to 5
      Mx = PivotRight
      GOSUB Movement
    NEXT

    FOR MoveLoop = 1 to 5
      Mx = WideTurnLeft
      GOSUB Movement
    NEXT

  Mx = FinishForward
  GOSUB Movement
END


' -----[ Subroutines ]------------------------------------------------------

' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = table index, table ends in xx

Movement:
  READ Mx, Dx                       ' read next action
  Mx = Mx + 1

  IF Dx = xx THEN MovementDone      ' skip if end of list

  GOSUB DoMovement                  ' execute movement
  GOTO Movement                     ' loop until done

DoMovement:
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                    ' will fall through if invalid index
MovementDone:
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt
```

```
TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo, Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN

StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo, CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
  RETURN
```

Activity #2: Different Turns

The basic turns will get the Toddler where it wants to go but there are many variations on this theme. For example, turning in place can be useful in tight places. The `PivotRight` table entry shows how this can be done. In this case pivoting is accomplished by performing one turning movement by moving the leg forward. This is immediately followed with the same type of movement but it starts by stepping backward first. The combination results in two turn actions and a new orientation while leaving the Toddler in approximately the same position.

The other table entry included in the program is `WideTurnLeft`. This takes the Toddler around a circle with a wider radius. The trick is adding a forward step after each turning action. A close look at the Toddler's path will show that the movement is not really an arc but rather the perimeter of a polygon with rounded corners. Still, this is close enough to an arc that most people will think the Toddler is going around a circle.

Hopefully the simplicity of the state transition approach is apparent with Program 3.1. It is identical to Program 2.4 except for the additional tables and calls that utilize these tables.

Challenges

1. Only a few turning variations were presented in the sample programs. Add the table entries needed to perform the actions not included.

2. The Toddler is symmetrical in construction and movement. It can make a turn going forwards or backwards. Create a program that can perform the actions presented but going backwards instead.

3. Full leg movements were used in the sample applications. Determine what happens if the movements are shorter. For example, instead of moving from state CR to CL, try moving from CR to CC.

4. WideTurnLeft turns the Toddler to the left but the turn radius is wider than TurnLeft. Make the Toddler's turn even wider. Hint: there are two ways of doing this. One is related to forward movement. The other is related to turn movement.

Experiment #4: Coordinated Walking

Walking and turning are useful operations but they don't get the Toddler very far. It is possible to string together a number of actions using multiple GOSUB statements but this can get tedious. It is also less efficient than the approach presented in this experiment.

The program in Experiment #3 used tables to store a series of basic movements. More complex actions can be done using very long tables but an alternative is to utilize these tables from a higher-level table. Instead of indicating whether the Toddler leans left or right, a movement table will include actions such as turn right, walk backward 10 steps, pivot left and walk forward 10 steps.

The second activity in this experiment uses this approach to move the Toddler in more complex paths than the earlier experiments but first we take a look at how to determine if a table is part of one set or another. This will allow the `Movement` routine to determine whether a table is a basic set of movement commands or if the table contains more complex commands. The `Movement` routine can then process the commands accordingly.

Activity #1: Which Table?

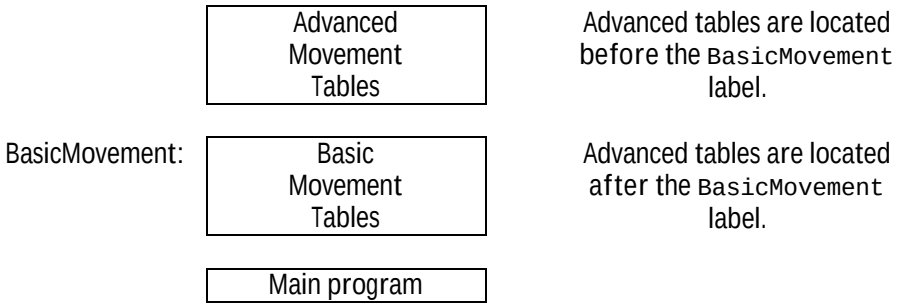
There are advantages to using high-level actions with the Toddler such as making a left turn versus low-level actions such as leaning left and moving the left foot forward. Both are needed and prior examples have shown how low level actions can be combined in tables to provide a higher level of abstraction. Taking this approach to the next level requires a different set of tables whose elements reference the lower level tables.

Using two types of tables is possible using two different routines but there is an advantage to using a single routine for both. This allows a program to freely mix the use of these two types of tables in the program. The program in this activity shows how this can be done. The approach is then used in the next activity's program.

This program does not make the Toddler walk but it does use the BASIC Stamp to run the program. It uses the BASIC Stamp editor's DEBUG window to display the output generated by DEBUG statements in the program. Most programmers familiar with the BASIC Stamp will already know about the DEBUG statement but check out the BASIC Stamp Manual if you are not. Also, the serial cable will remain connected to the Toddler for this experiment.

This program assumes that two kinds of tables will be used with the program and that each type of table will be in its own area of the program memory. There is no restriction that tables be adjacent, only that they be above or below the boundary that is designated by the `BasicMovements` label.

Table 4.1: Coordinated Walking Table Structure



The following program uses a `Movement` routine that has the index of the table in the `Mx` variable. It uses the `DEBUG` statement to output a string in the debug window on the PC that indicates whether index is for an advanced or basic movement. In the next activity, the `Movement` routine will be replaced by one that actually interprets the tables to make the Toddler move.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 4.1: Which Table
' {$STAMP BS2}


' -----[ Constants ]-------------------------------------------------------------


' -----[ Variables ]-------------------------------------------------------------

Mx              VAR     Word


' -----[ Movement Support Codes ]------------------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL              CON     0
TC              CON     1
TR              CON     2
```

```
SL              CON     3
SC              CON     4
SR              CON     5

xx              CON     255


' -----[ Movement Value Tables ]------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle          DATA            7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle         DATA            7, bRightTurn, bRightTurn, bForward, xx

WalkForward3            DATA            3, bForward, xx
WalkForward8            DATA            8, bForward, xx


' -----[ Basic Movement Codes ]--------------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish                 CON             0
bForward                CON             1
bBackward               CON             2
bLeftTurn               CON             3
bRightTurn              CON             4
bPivotLeft              CON             5
bPivotRight             CON             6


' -----[ Basic Movement Tables ]--------------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements          CON             Forward

Forward                 DATA            1, TR, SL, TL, SR, xx
Backward                DATA            1, TR, SL, TL, SR, xx

LeftTurn                DATA            1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn               DATA            1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft               DATA            3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight              DATA            3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                  DATA            1, TR, SC, TC, xx
```

```
' -----[ Main Code ]------------------------------------------------------------

Main_Program:
  Mx = LeftSemicircle
  GOSUB Movement

  Mx = WalkForward3
  GOSUB Movement

  Mx = PivotRight
  GOSUB Movement

  Mx = WalkForward8
  GOSUB Movement

  Mx = Finish
  GOSUB Movement
END


' -----[ Subroutines ]------------------------------------------------------------

Movement:
  IF Mx < BasicMovements THEN BasicMovementTable

  debug hex Mx, " is an advanced movement table",cr
  RETURN

BasicMovementTable:
  debug hex Mx, " is a basic movement table",cr
  RETURN
```

This approach to memory partitioning takes into account that PBASIC allocates space for DATA statements in the order that they appear in the program. Not all programming languages do this so the technique should only be applied when the appropriate support is available. It is handy in this instance because other approaches to differentiating data are more cumbersome. This approach can be prone to programmer errors if the tables are not grouped properly but keeping the tables relatively close to each other in the program text makes it easy to spot such problems.

Activity #2: Figure 8s and Square Dancing

Now we take a look at making the Toddler execute more complex movements using the dual table types presented in Activity #1. One set of tables handles low-level actions such as tilting and leg movements. The second set of tables handles higher-level actions such as turning a corner and walking in a large circle.

In this Activity, the program makes the Toddler walk in a Figure 8 and a large square. The use of a high-level action table allows easy creation of more complex movement sequences. In this case, the use of higher level sequences like `LeftSemicircle` cause the Toddler to execute a large number of basic foot movements.

The program implements a more sophisticated version of the `Movement` routine than found in prior Activities. In this case, the `Mx` variable can contain an index for either a basic or an advanced table. The `Movement` routine will execute the appropriate table. The decoding process is a bit complex so we have included `DEBUG` statements to help present the execution process. The `DEBUG` statements are actually comments in the listing but they can be changed by doing a "Replace All" in the editor from " 'debug" to "debug". The converse will change the lines back to comments.

The `DEBUG` statements are only useful when the Toddler is connected to the PC since the display of information is done on the PC. While it is possible to keep the Toddler connected to the PC while it is walking, these more advanced movement sequences move the Toddler in large areas. A laptop or a long serial cable may be necessary to handle these larger movement areas.

What we have found useful is to instead use the `DEBUG` version while running the Toddler with the power switch in the download mode. In this case, the Toddler's servos do not move but the program continues to execute. The `DEBUG` statements show what the Toddler would be doing if the power switch was in the RUN mode (Position 2).

When running in the download mode, the Toddler will continue to send pulses to the servos even though the servos receive no power and hence do not rotate. The delay does slow down debug presentation though. It is possible to add a `RETURN` statement immediately after the `DoMovement` label to eliminate this thereby making the debug process go faster. Just make sure to comment out or remove the `RETURN` statement or the Toddle will not move.

Now for the code.

```
' -----[ Title ]-----------------------------------------------------------------
' Toddler Program 4.2: Advanced Walking
' {$STAMP BS2}


' -----[ I/O Definitions ]--------------------------------------------------------

TiltServo             CON           13            ' Tilt servo on P12
StrideServo           CON           12            ' Stride servo on P13


' -----[ Constants ]--------------------------------------------------------------

MoveDelay             CON           25            ' in micrcoseconds
```

```
TiltStep              CON           10            ' TiltServo step size

RightTilt             CON           620           ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10            ' StrideServo step size

RightStride           CON           650           ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]----------------------------------------------------------

FigureLoop            VAR           Nib
MoveLoop              VAR           Byte          ' Loop for repeat movements
MoveLoopLimit         VAR           Byte

SubMoveLoop           VAR           Byte          ' Loop for repeat submovements
SubMoveLoopLimit      VAR           Byte

Pulses                VAR           Word          ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses

Mx                    VAR           Word
MxCurrent             VAR           Word

Sx                    VAR           Word
SxCurrent             VAR           Word


' -----[ Movement Support Codes ]----------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL            CON     0
TC            CON     1
TR            CON     2

SL            CON     3
SC            CON     4
SR            CON     5
```

```
xx              CON     255


' -----[ Movement Value Tables ]-------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle          DATA            7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle         DATA            7, bRightTurn, bRightTurn, bForward, xx

WalkForward3            DATA            3, bForward, xx
WalkForward8            DATA            8, bForward, xx


' -----[ Basic Movement Codes ]---------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish                 CON             0
bForward                CON             1
bBackward               CON             2
bLeftTurn               CON             3
bRightTurn              CON             4
bPivotLeft              CON             5
bPivotRight             CON             6


' -----[ Basic Movement Tables ]---------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements CON              Forward

Forward                 DATA            1, TR, SL, TL, SR, xx
Backward                DATA            1, TR, SL, TL, SR, xx

LeftTurn                DATA            1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn               DATA            1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft               DATA            3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight              DATA            3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                  DATA            1, TR, SC, TC, xx


' -----[ Main Code ]-------------------------------------------------------------

Main_Program:
  GOSUB ResetCC
```

```
    ' Make a Figure 8

    FOR FigureLoop = 1 to 5
      Mx = LeftSemicircle
      GOSUB Movement

      Mx = WalkForward3
      GOSUB Movement

      Mx = RightSemicircle
      GOSUB Movement

      Mx = WalkForward3
      GOSUB Movement
    NEXT


    ' Make a big polygon

    FOR FigureLoop = 1 to 5
      Mx = PivotRight
      GOSUB Movement

      Mx = WalkForward8
      GOSUB Movement
    NEXT

  Mx = Finish
  GOSUB Movement
END


' -----[ Subroutines ]------------------------------------------------------

' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                            ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit                    ' read movement table repeat count
  MxCurrent = Mx + 1
```

```
StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                          ' Mx = start of movement table

 'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                            ' skip if movement table
    SxCurrent = Mx                          ' SxCurrent = submovement table index
      GOTO StartSubMovement                 ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent                    ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                            ' skip if end of list
 'debug "  ", dec SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                            ' lookup submovement table index
StartSubMovement:                           ' start executing submovement table
      READ SxCurrent, SubMoveLoopLimit
                                            ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
        Sx = SxCurrent

 'debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr

SubMovementLoop:
          READ Sx, Dx                       ' read next submovent action
          Sx = Sx + 1

          IF Dx = xx THEN SubMovementDone
                                            ' skip if end of list
          GOSUB DoMovement                  ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      GOTO MovementLoop

MovementDone:
  NEXT
  RETURN

DoMovement:
 'debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                            ' will fall through if invalid index
  RETURN
```

```
' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride
```

```
  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
  RETURN
```


The `Main_Program` now executes so the Toddler performs two large movements: a figure 8 and a square. The `Movement` routine is called to execute high-level tables that include commands such as left turn. The `bLeftTurn` value is used in the table because the `DATA` statements only store bytes. These values are used with a `LOOKUP` statement in the `Movement` routine to select the appropriate low level table to use for basic movements. One high-level table entry causes the Toddler to execute many low level movements.

The `Movement` routine still handles low-level tables using the technique outlined in Activity #1. For example, the `Finish` table uses only a few basic movements. If the `Movement` routine did not handle both types of tables then either a low level routine would have to be called or a high level table would have to be created.

Challenges

1. A challenge in the Experiment #5 asked to extend the movement routine so it could handle repetitions of subsections in a table as in.

```
SpecialMovement    DATA    4, TL, SR, TC, SL, xy
                   DATA    2. TR, SL, TL, SR, xy
                   DATA    2. TL, SR, TC, SL, xx
```

   Provide this same type of facility for both types of movement tables.

2. Implement the prior challenge. Then reduce the multiple calls to Movement shown below

```
      Mx = LeftSemicircle
      GOSUB Movement

      Mx = WalkForward3
      GOSUB Movement

      Mx = RightSemicircle
      GOSUB Movement

      Mx = WalkForward3
      GOSUB Movement
```

   to the following

```
      Mx = Figure8
      GOSUB Movement
```

   Hint: The Figure 8 table will be a combination of the four tables listed above.

3. Generate a set of symmetrical useful high level and low level movement tables. High level movements might include TurnAroundLeft, TurnAroundRight, WalkForward1Foot, and WalkBackward1Foot. Low level movements might include ReversePivotRight and ReversePivotLeft.

Experiment #5: Following Light

The photoresistors in your kit can be used to make your Toddler detect variations in light level. With some programming, your Toddler can be transformed into a photophile (a creature attracted to light), or a photophobe (a creature that tries to avoid light).

To sense the presence and intensity of light you'll build a couple of photoresistor circuits on your Toddler. A photoresistor is a light-dependent resistor (LDR) that covers the spectral sensitivity similar to that of the human eye. The active elements of these photoresistors are made of Cadmium Sulfide (CdS). Light enters into the semiconductor layer applied to a ceramic substrate and produces free charge carriers. A defined electrical resistance is produced that is inversely proportional to the illumination intensity. In other words, darkness produces high resistance, and high illumination produces very small amounts of resistance.

The specific photoresistors included in the Toddler kit are from EG&G Vactec (#VT935G). If you need additional photoresistors they are available from Parallax's Component Shop as well as from many electronic component suppliers. See Appendix A: Toddler Parts Lists and Sources. The specifications of these photoresistors are shown in Figure 5.1:

Figure 5.1: EG&G Vactec Photoresistor Specifications

Resistance (Ohms)					Peak Spectral Response	V_{MAX}	Response Time @ 1 fc (ms, typ.)	
10 Lux 2850K			Dark					
Min	Typ.	Max.	Min.	Sec.	nm		Rise (1-1/e)	Fall (1/e)
20K	29.0K	38K	1M	10	550	100	35	5

Luminance is a scientific name for the measurement of incident light. The unit of measurement of luminance is commonly the "foot-candle" in the English system and the "lux" in the metric system. While using the photoresistors we won't be concerned about lux levels, just whether or not luminance is higher or lower in certain directions. The Toddler can be programmed to use the relative light intensity information to make navigation decisions. For more information about light measurement with a microcontroller, take a look at Earth Measurements Experiment #4, Light on Earth and Data Logging.

Activity #1: Building and Testing Photosensitive Eyes

Figure 5.2 shows the capacitors and photoresistor used in this experiment along with their schematic symbols. Both capacitors are nonpolar, meaning that terminals 1 and 2 as shown may be swapped without affecting the circuit. In addition to the capacitors, you'll also need two (2) 220 ohm resistors (color code red, red, brown).

(2) Photoresistors
(2) 0.1 µF capacitors
(2) 0.01 µF capacitors
(2) 220 ohm resistors (not pictured)
(misc.) jumper wires

Figure 5.2: Photoresistor and capacitor circuit symbols and parts.

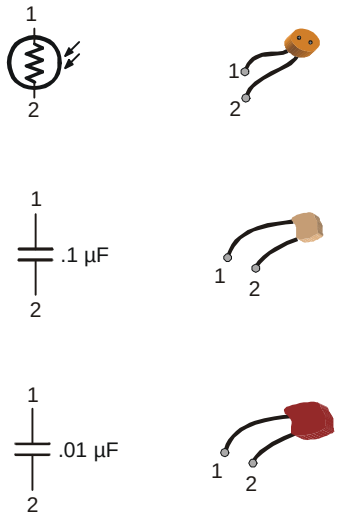
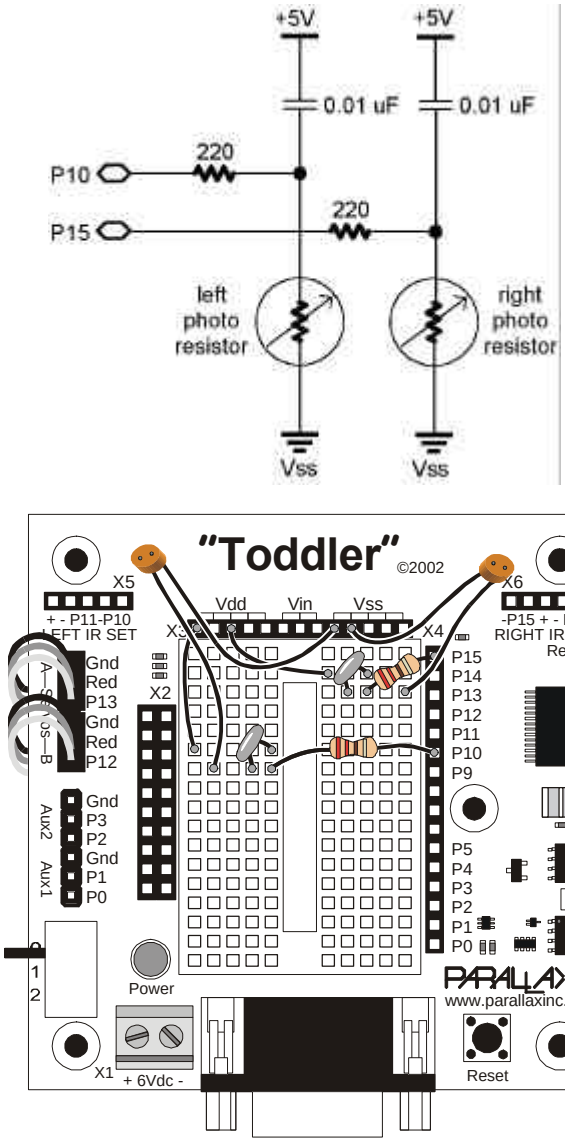


Figure 5.3 shows the resistor/capacitor (RC) circuit for each photoresistor. A photoresistor is an analog device. Its value varies continuously as luminance, another analog value, varies. The photoresistor's resistance is very low when it's light-sensitive surface is placed in direct sunlight. As the light level decreases, the photoresistor's resistance increases. In complete darkness, the photoresistor's value can increase to more than 1 M Ohm. Even through the photoresistor is analog, its response to light is nonlinear. This means if the input source (luminance) varies at a constant rate, the photoresistor's value does not necessarily vary at a constant rate.

Figure 5.3: Photoresistor Schematic and Pictorial



Programming to Measure the Resistance

The circuit in Figure 5.3 was designed for use with the PBASIC RCTIME command. This command can be used with an RC circuit where one value, either R or C, varies while the other remains constant. The RCTIME command lends itself to measuring the variable values because it takes advantage of a time varying property of RC circuits.

For one of the RC circuits shown in Figure 5.3, the first step in setting up the RCTIME measurement is charging the lower plate of the capacitor to 5 V. Setting the I/O pin connected to the lower capacitor plate by the 220 Ohm resistor high for a few ms takes care of this. Next, the RCTIME command can be used to take the measurement of the time it takes the lower plate to discharge from 5 to 1.4 V. Why 1.4 V? Because that's the BASIC Stamp I/O pin's threshold voltage. When the voltage at an I/O pin set to input is above 1.4 V, the value in the input register bit connected to that I/O pin is "1." When the voltage is below 1.4 V, the value in the input register bit is "0."

In this circuit RCTIME measures the time it takes the voltage at the lower plate of the capacitor in one of the Figure 5.3 RC circuits to drop from 5 to 1.4 V. This discharge is directly proportional to the photoresistor's resistance. Since this resistance varies with luminance (exposure to varying levels of light), so does the time. By measuring this time, relative light exposure can be inferred. See the BASIC Stamp Manual for a detailed discussion of RCTIME.

The RCTIME command changes the I/O pin from output to input. As soon as the I/O pin becomes an input, the voltage at the lower plate of the capacitor starts to fall according to the time equation just discussed. The BASIC Stamp starts counting in 2 us increments until the voltage at the capacitor's lower plate drops below 1.4 V.

✓ TIP	For Best Results: Eliminate direct sunlight; it's too bright for the photoresistor circuits. They perform best in lower light levels while seeking indirect natural light.

Run Program Listing 5.1. It demonstrates how to use the RCTIME command to read the photoresistors. This program makes use of the Debug Terminal, so leave the serial cable connected to the Toddler board while Program is running.

```
' Toddler Program 5.1: Photoresistor RCTime Display
' {$Stamp bs2}                                      ' Stamp Directive.

'----- Declarations --------------

  left_photo   var word                             ' For storing measured RC times of
  right_photo  var word                             ' the left & right photoresistors.

  left_pin     con 10
  right_pin    con 15


'----- Initialization ------------

  debug cls                                         ' Open and clear a Debug Terminal.

'----- Main Routine --------------

main:

  ' Measure RC time for left photoresistor.

  high left_pin                                     ' Set detector to output-high.
  pause 3                                           ' Pause for 3 ms.
  rctime left_pin,1,left_photo                      ' Measure RC time on left detector.

  ' Measure RC time for right photoresistor.

  high right_pin                                    ' Set detector to output-high.
  pause 3                                           ' Pause for 3 ms.
  rctime right_pin,1,right_photo                    ' Measure RC time on right detector.

  ' Display RC time measurements using Debug Terminal.

  debug home, "L  ", dec5 left_photo, "  R  ", dec5 right_photo

goto main
```

5

How The Photoresistor Display Works

Two word variables, `left_photo` and `right_photo` are declared for storing the RC time values of the left and right photoresistors. The `main` routine then measures and displays the RC times for each RC circuit. The code for reading the right RC circuit is shown below. First, the I/O pin `left_pin` is set to output-high. Next, a 3 ms pause allows enough time for the capacitor to charge. After 3 ms, the lower plate of the capacitor is close enough to 5 V and is ready for the `rctime` measurement. The `rctime` command measures the RC time on I/O pin `right_pin`, with a beginning state of "1" (5 V), and stores the result in the `right_photo` variable.

Remember, the value stored in `right_photo` is a number. This number tells how many 2 us increments passed before the voltage at the lower plate of the capacitor passed below the I/O pin's 1.4 V threshold.

```
high right_pin
pause 3
rctime right_pin,1,right_photo
```

Try replacing one of the 0.01 μF capacitors with a 0.1 μF capacitor. Which circuit fares better in bright light, the one with the larger (0.1 μF) or the one with the smaller (0.01 μF) capacitor? What is the effect as the surroundings get darker and darker? Do you notice any symptoms that would indicate that one or the other capacitor would work better in a darker environment?

Make sure to restore your circuit to its original state before moving on to the next activity.

Activity #2: A Light Compass

If you focus a flashlight beam in front of the Toddler, the circuit and programming techniques just discussed can be used to make the Toddler turn so that it's pointing at the flashlight beam. Make sure the photoresistors are pointed so that they can make a light comparison. Aside from each being pointed 45° outward from the center-line of the Toddler, they also should be oriented so they are pointing 45° downward from horizontal. In other words, point the faces of the photoresistors down toward the table top. Then, use a bright flashlight to make the Toddler track the direction of the light.

Programming the Toddler to Point at the Light

Getting the Toddler to track a light source is a matter of programming it to compare the value measured at each photoresistor. Remember that as the light gets dimmer, the photoresistor's value increases. So, if the photoresistor value on the right is larger than that of the photoresistor on the left, it means it's brighter on the left. Given this situation, the Toddler should turn left. On the other hand, if the `RCTIME` of the photoresistor on the left is larger than that of the photoresistor on the right, the right side is brighter and the Toddler should turn right.

To keep the Toddler from changing directions too often, a parameter for deadband is introduced. Deadband is a range of values wherein the system makes no attempt at correction. If the numbers go above or below the deadband, then the system corrects accordingly. The most convenient way to measure for deadband is to subtract the left `RCTime` from the right `rctime`, or visa versa, then take the absolute value. If this absolute value is within the deadband limits, then do nothing; if otherwise, program an appropriate adjustment.

- Enter and run Program Listing 5.2.
- Shine a bright flashlight in front of the Toddler. When you move the flashlight, the Toddler should rotate so that it's pointing at the flashlight beam.
- Instead of using a flashlight, use your hand to cast a shadow over one of the photoresistors. The Toddler should rotate away from the shadow.

```
' Toddler Program 5.2: Light Compass
' {$Stamp bs2}                                      ' Stamp Directive.

' -----[ I/O Definitions ]-------------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------------
```

```
MoveDelay             CON           15            ' in micrcoseconds

TiltStep              CON           5             ' TiltServo step size

RightTilt             CON           620           ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           5             ' StrideServo step size

RightStride           CON           650           ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]----------------------------------------------------------

FigureLoop            VAR           Nib
MoveLoop              VAR           Byte          ' Loop for repeat movements
MoveLoopLimit         VAR           Byte

SubMoveLoop           VAR           Byte          ' Loop for repeat submovements
SubMoveLoopLimit      VAR           Byte

Pulses                VAR           Word          ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses

Mx                    VAR           Word
MxCurrent             VAR           Word

Sx                    VAR           Word
SxCurrent             VAR           Word


' -----[ Movement Support Codes ]---------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL            CON     0
TC            CON     1
TR            CON     2

SL            CON     3
SC            CON     4
```

```
SR              CON     5

xx              CON     255


' -----[ Movement Value Tables ]------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle DATA           7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle DATA          7, bRightTurn, bRightTurn, bForward, xx

WalkForward3    DATA          3, bForward, xx
WalkForward8    DATA          8, bForward, xx


' -----[ Basic Movement Codes ]--------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish         CON           0
bForward                CON           1
bBackward               CON           2
bLeftTurn               CON           3
bRightTurn              CON           4
bPivotLeft              CON           5
bPivotRight             CON           6


' -----[ Basic Movement Tables ]--------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements CON            Forward

Forward         DATA          1, TR, SL, TL, SR, xx
Backward        DATA          1, TR, SR, TL, SL, xx

LeftTurn        DATA          1, TL, SR, TC, SL, xx
RightTurn       DATA          1, TR, SL, TC, SR, xx

PivotLeft       DATA          3, TL, SR, TC, SL, xx
PivotRight      DATA          3, TR, SL, TC, SR, xx

Finish          DATA          1, TR, SC, TC, xx


'----- Photodetector Declarations --------------
```

```
  left_photo   var word                      ' For storing measured RC times of
  right_photo  var word                      ' the left & right photoresistors.

  left_pin     con 10
  right_pin    con 15


'----- Initialization ------------

  output 2                                   ' Set P2 to output.
  freqout 2, 2000, 3000                      ' Declare a variable for counting.

  GOSUB ResetCC                              ' Initialize feet

'----- Main Routine --------------

main:

  ' Measure RC time for left photoresistor.

  high left_pin                              ' Set detector to output-high.
  pause 3                                    ' Pause for 3 ms.
  rctime left_pin,1,left_photo               ' Measure RC time on left detector.

  ' Measure RC time for right photoresistor.

  high right_pin                             ' Set detector to output-high.
  pause 3                                    ' Pause for 3 ms.
  rctime right_pin,1,right_photo             ' Measure RC time on right detector.

  ' Take the difference between right_photo and left_photo, then decide what to do.
   DEBUG home, "Left = ", dec left_photo, " Right = ",dec right_photo,cr

    if abs(left_photo-right_photo) < 4 then main
    if left_photo > right_photo then turn_right
    if left_photo < right_photo then turn_left

'----- Navigation Routines -------

Turn_left:                                   ' turn left towards light
  Mx = PivotLeft
  GOSUB Movement
  goto main                                  ' go back to main routine.

Turn_right:                                  ' turn right towards light
  Mx = PivotRight
  GOSUB Movement
  goto main                                  ' go back to main routine.


' -----[ Subroutines ]------------------------------------------------------

' ----- Movement: Move feet using DATA table referenced by Mx -----
```

```
'
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                            ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit                    ' read movement table repeat count
  MxCurrent = Mx + 1

StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                          ' Mx = start of movement table

 'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                            ' skip if movement table
    SxCurrent = Mx                          ' SxCurrent = submovement table index
      GOTO StartSubMovement                 ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent                    ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                            ' skip if end of list
 'debug "  ", dec SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                            ' lookup submovement table index
StartSubMovement:                           ' start executing submovement table
      READ SxCurrent, SubMoveLoopLimit
                                            ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
        Sx = SxCurrent

 'debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr

SubMovementLoop:
          READ Sx, Dx                       ' read next submovent action
          Sx = Sx + 1

          IF Dx = xx THEN SubMovementDone
```

```
                                            ' skip if end of list
          GOSUB DoMovement                  ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      IF Mx < BasicMovements THEN MovementLoop

MovementDone:
  NEXT
  RETURN

DoMovement:
 'debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                            ' will fall through if invalid index
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride
```

```
MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
  RETURN
```


How the Light Compass Works

Program 5.2 takes RC time measurements and first checks to see if the difference between the values returned by the `rctime` commands fall in the deadband using the command:

```
if abs(left_photo - right_photo) < 2 then main
```

If the difference between RC times is within the deadband, the program jumps to the `Main:` label. If the measured difference in RC times is not within the deadband, two `IF...THEN` statements decide which routine to call, `turn_left` or `turn_right`.

```
if left_photo > right_photo then turn_right
if left_photo < right_photo then turn_left
```

These routines use the movement routines initially presented in the prior chapter. The Toddler can make smaller turns.

Your Turn

In a darker area, not only will the photoresistor values be larger, so will the difference between them. You may have to increase the deadband in low ambient light to detune the Toddler to small and changing variations in light. The lower the light levels, the less you need the PAUSE statements. If the Toddler's performance starts to decrease, it's probably because the time between pulses has exceeded 40 ms. The first line of defense for this problem is to reduce the `PAUSE Period` in each subroutine to zero. The second line of defense is to check photoresistors during alternate pulses. That way, after the first pulse, the right photoresistor could be checked. Then, after the second pulse, the left photoresistor could be checked. You can try your hand at developing code that does this in the Challenges section.

The deadband value is currently set to "2" in the expression:

```
if abs(left_photo-right_photo) < 2 then main
```

- Experiment with different ambient light levels and their effect on deadband by trying this experiment in lighter and darker areas. In lighter areas, the deadband value can be made smaller, even zero. In darker areas, the deadband value should be increased.

- Swap the conditions in the second and third `if...then` statement in Program 5.2. Then re-run the program. Now your Toddler points away from the light.

Activity #3: Following The Light

Programming the Toddler to follow light requires that only a few modifications to Program Listing 5.2 be made. The main change is that measurements within the deadband resulted in no motion in Program Listing 5.2. In Program Listing 5.3, when the difference between RC times falls within the deadband, it results in forward motion. Let's see how it works.

```
' Toddler Program 5.3: Follow The Light
' {$Stamp bs2}                                    ' Stamp Directive.

' -----[ I/O Definitions ]-------------------------------------------------

TiltServo             CON           13            ' Tilt servo on P12
StrideServo           CON           12            ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------

MoveDelay             CON           25            ' in micrcoseconds

TiltStep              CON           10            ' TiltServo step size
```

```
RightTilt               CON             620             ' Tilt limits
CenterTilt              CON             750
LeftTilt                CON             880


StrideStep              CON             10              ' StrideServo step size

RightStride             CON             650             ' Stride limits
CenterStride            CON             750
LeftStride              CON             850


' -----[ Variables ]------------------------------------------------------------

FigureLoop              VAR             Nib
MoveLoop                VAR             Byte            ' Loop for repeat movements
MoveLoopLimit           VAR             Byte

SubMoveLoop             VAR             Byte            ' Loop for repeat submovements
SubMoveLoopLimit        VAR             Byte

Pulses                  VAR             Word            ' Pulse variable

CurrentTilt             VAR             Word
CurrentStride           VAR             Word
NewValue                VAR             Word

Dx                      VAR             Pulses

Mx                      VAR             Word
MxCurrent               VAR             Word

Sx                      VAR             Word
SxCurrent               VAR             Word


' -----[ Movement Support Codes ]------------------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL              CON     0
TC              CON     1
TR              CON     2

SL              CON     3
SC              CON     4
SR              CON     5

xx              CON     255
```

```
' -----[ Movement Value Tables ]------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle        DATA          7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle       DATA          7, bRightTurn, bRightTurn, bForward, xx

WalkForward3          DATA          3, bForward, xx
WalkForward8          DATA          8, bForward, xx


' -----[ Basic Movement Codes ]--------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish               CON           0
bForward              CON           1
bBackward             CON           2
bLeftTurn             CON           3
bRightTurn            CON           4
bPivotLeft            CON           5
bPivotRight           CON           6


' -----[ Basic Movement Tables ]-------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements        CON           Forward

Forward               DATA          1, TR, SL, TL, SR, xx
Backward              DATA          1, TR, SL, TL, SR, xx

LeftTurn              DATA          1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn             DATA          1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft             DATA          3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight            DATA          3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                DATA          1, TR, SC, TC, xx


'----- Photodetector Declarations --------------

  left_photo          var word                    ' For storing measured RC times of
  right_photo         var word                    ' the left & right photoresistors.

  left_pin            con 10
  right_pin           con 15
```

```
'----- Initialization ------------

  output 2                                  ' Set P2 to output.
  freqout 2, 2000, 3000                     ' Declare a variable for counting.
  low 12                                    ' Set P12 and 13 to output-low.
  low 13

  GOSUB ResetCC                             ' Initialize feet

'----- Main Routine --------------

main:

  ' Measure RC time for left photoresistor.

  high left_pin                             ' Set detector to output-high.
  pause 3                                   ' Pause for 3 ms.
  rctime left_pin,1,left_photo              ' Measure RC time on left detector.

  ' Measure RC time for right photoresistor.

  high right_pin                            ' Set detector to output-high.
  pause 3                                   ' Pause for 3 ms.
  rctime right_pin,1,right_photo            ' Measure RC time on right detector.

  ' Take the difference between right_photo and left_photo, then decide what to do.

    if abs(left_photo-right_photo) > 8 then check_dir

' Check if difference between RC times is within the deadband, 2 in this case.
  ' If yes, then forward.  If no then skip to check_dir subroutine.

  walk_forward:
    Mx = Forward
    GOSUB Movement
    goto main

  ' Jump to either right_turn or left_turn depending on which RC time is larger.

  check_dir:
    if left_photo > right_photo then turn_right
    if left_photo < right_photo then turn_left


'----- Navigation Routines -------

turn_left:                                  ' turn left towards light
  Mx = PivotLeft
  GOSUB Movement
  goto main                                 ' go back to main routine.
```

```
Turn_right:                                 ' turn right towards light
  Mx = PivotRight
  GOSUB Movement
  goto main                                 ' go back to main routine.


' -----[ Subroutines ]------------------------------------------------------

' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                            ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit                    ' read movement table repeat count
  MxCurrent = Mx + 1

StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                          ' Mx = start of movement table

 'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                            ' skip if movement table
    SxCurrent = Mx                          ' SxCurrent = submovement table index
      GOTO StartSubMovement                 ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent                    ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                            ' skip if end of list
 'debug "  ", dec SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                            ' lookup submovement table index
StartSubMovement:                           ' start executing submovement table
      READ SxCurrent, SubMoveLoopLimit
                                            ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
```

```
        Sx = SxCurrent

 'debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr

SubMovementLoop:
          READ Sx, Dx                        ' read next submovent action
          Sx = Sx + 1

          IF Dx = xx THEN SubMovementDone
                                             ' skip if end of list
          GOSUB DoMovement                   ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      IF Mx < BasicMovements THEN MovementLoop

MovementDone:
  NEXT
  RETURN

DoMovement:
  'debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                             ' will fall through if invalid index
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
```

```
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
RETURN
```

How the Light Follower Program Works

As in the previous program, the first `IF...THEN` statement tests for a difference in RC time measurements within the deadband. This statement has been modified so that it skips the `walk_forward` routine if the difference between RC times falls outside the deadband. On the other hand, if the difference in RC times is within the deadband, the forward pulse is executed. After the forward pulse, the program is directed back to `main` and the RC times are checked again.

```
if abs(left_photo-right_photo) > 2 then check_dir

walk_forward:
  Mx = Forward
```

```
  GOSUB Movement
  goto main
```

If the difference between RC times is not within the deadband, the program skips to the `Check_dir` label. The `IF...THEN` statements following the `Check_dir` label are used to decide whether to apply a pulse to the left or a pulse to the right depending on the inequality of the `right_photo` and `left_photo` values. In this way, the program either applies a single forward pulse or a single turn pulse each time the photoresistors are checked.

```
check_dir:
  if left_photo > right_photo then turn_right
  if left_photo < right_photo then turn_left
```

Challenges

1. Repeat the previous Your Turn exercise. You can now lead your Toddler around with a flashlight.

2. Instead of pointing the photoresistors at the surface directly in front of the Toddler, point them upward and outward. With the photoresistors adjusted this way, the Toddler will roam on the floor and try to always find the brightest place.

3. Depending on the luminance gradient, you may have to increase the deadband to smooth out the Toddler's light roaming. Alternatively, the deadband may need to be decreased to make it more responsive to seeking out the brighter areas.

Experiment #6: Object Avoidance with Infrared

Using Infrared Headlights to See the Road

Today's hottest products seem to have one thing in common: wireless communication. Personal organizers beam data into desktop computers, and wireless remotes let us channel surf. With a few inexpensive and widely available parts, the BASIC Stamp can also use an infrared LED and detector to detect objects to the front and side of your traveling Toddler.

What's Infrared?

Infra means below, so Infra-red is light (or electromagnetic radiation) that has lower frequency, or longer wavelength than red light. Our IR LED and detector work at 980 nm. (nanometers) which is considered near infrared. Night-vision goggles and IR temperature sensing use far infrared wavelengths of 2000-10,000 nm., depending on the application.

Color	Approximate Wavelength
Violet	400 nm
Blue	470
Green	565
Yellow	590
Orange	630
Red	780
Near infra-red	800-1000
Infra-red	1000-2000
Far infra-red	2000-10,000nm

Detecting obstacles doesn't require anything as sophisticated as machine vision. A much simpler system will suffice. Some robots use RADAR or SONAR (sometimes called SODAR when used in air instead of water). An even simpler system is to use infrared light to illuminate the robot's path and determine when the light reflects off an object. Thanks to the proliferation of infrared (IR) remote controls, IR illuminators and detectors are easily available and inexpensive.

The Toddler infrared object detection scheme has a variety of uses. The Toddler can use infrared to detect objects without bumping into them. As with the photoresistors, infrared can be used to detect the difference between black and white for line following. Infrared can also be used to determine the distance of an object from the Toddler. The Toddler can use this information to follow objects at a fixed distance, or detect and avoid high ledges.

Infrared Headlights

The infrared object detection system we'll build on the Toddler is like a car's headlights in several respects. When the light from a car's headlights reflects off obstacles, your eyes detect the obstacles and your brain processes them and makes your body guide the car accordingly. The Toddler uses infrared LEDs for headlights. They emit infrared, and in some cases, the infrared reflects off objects, and bounces back in the direction of the Toddler. The eyes of the Toddler are the infrared detectors. The infrared detectors send signals to the BASIC Stamp indicating whether or not they detect infrared reflected off an object. The brain of the Toddler, the BASIC Stamp, makes decisions and operates the servo motors based on this input.

The IR detectors have built-in optical filters that allow very little light except the 980 nm. infrared that we want to detect onto its internal photodiode sensor. The infrared detector also has an electronic filter that only allows signals around 38.5 kHz to pass through. In other words, the detector is only looking for infrared flashed on and off at 38,500 times per second. This prevents interference from common IR interference sources such as sunlight and indoor lighting. Sunlight is DC interference (0 Hz), and house lighting tends to flash on and off at either 100 or 120 Hz, depending on the main power source in the country where you reside. Since 120 Hz is way outside the electronic filter's 38.5 kHz band pass frequency, it is, for all practical purposes, completely ignored by the IR detectors.

The FREQOUT Trick

Since the IR detectors only see IR signals in the neighborhood of 38.5 kHz, the IR LEDs have to be flashed on and off at that frequency. A 555 timer can be used for this purpose, but the 555 timer circuit is more complex and less functional than the circuit we will use in this and the next chapter. For example, the method of IR detection introduced here can be used for distance detection; whereas, the 555 timer would need additional hardware to do distance detection.

A pair of Toddler enthusiasts found an interesting trick that made the 555 timer scheme unnecessary. This scheme uses the `FREQOUT` command without the RC filter that's normally used to smooth the signal into a sine-wave. Even though the highest frequency `FREQOUT` is designed to transmit is 32768 Hz, the unfiltered `FREQOUT` output contains a harmonic with useful properties for a 38.5 kHz IR detector. More useful still is the fact that you can use a command such as `FREQOUT Pin, Period, 38500` to send a 38.5 kHz harmonic that the IR detector will detect.

Figure 6.1 shows (a) the signal sent by the command `FREQOUT Pin, Period, 27036`. Tuned electronic receivers, such as the IR detectors we'll be using, can detect components of this signal that are called harmonics. The `FREQOUT` signal's two dominant low frequency harmonics are shown in Figures 6.1 (b) and (c). Figure 6.1 (b) shows the fundamental harmonic, and Figure 6.1 (c) shows the third harmonic. These harmonics are actually components of the unfiltered `FREQOUT` pulses shown in Figure 6.1 (a). The third harmonic shown in Figure 6.1 (c) can be controlled directly by entering commands such as `FREQOUT Pin, Period, 38500` (instead of `27036`) for 38.5 kHz, or `FREQOUT Pin, Period, 40000` for 40 kHz, etc.

Figure 6.1: FREQOUT Example Properties

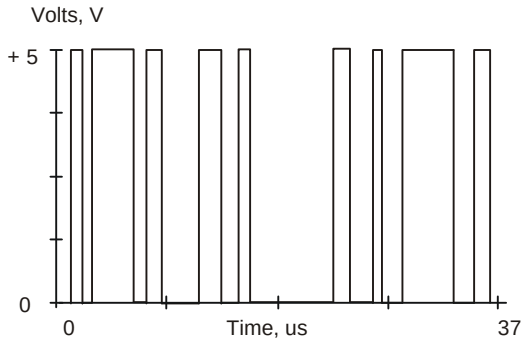
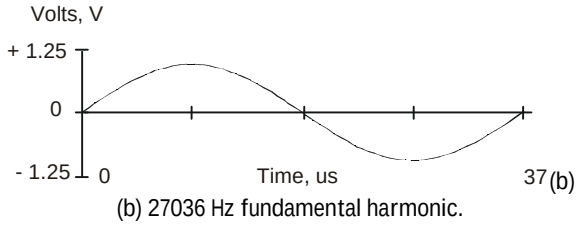
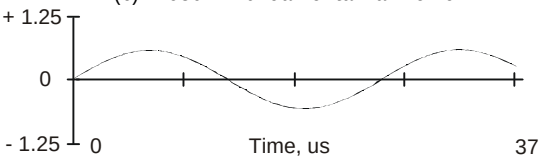


Figure 6.2: (a) Unfiltered freqout pulses sent by `freqout pin, period, 27036`



(b) 27036 Hz fundamental harmonic.



(c) 38500 Hz third harmonic.

6

Even though the "freqout" trick works, there is an additional problem. The BASIC Stamp does not multitask. The reason this is a problem is because the IR detector only sends the low signal indicating that it has detected an object while it is receiving the 38.5 kHz IR. Otherwise, it sends a high signal. Fortunately, it takes the detector long enough to rebound from its low output state that the BASIC Stamp can capture the value. The reason that the detector's output takes so long to rebound is related to its tendency toward slower responses when it receives a signal with unequal high and low times, of which the signal in Figure 5.2 (a) has many.

Activity #1: Building and Testing the New IR Transmitter/Detector

The circuit requires just a few parts:

(1) Piezoelectric speaker
(2) Shrink wrapped IR LEDs
(2) IR detectors
(misc) wires

Figure 6.2 shows the part schematic and pictorials. Figure 6.3 is the schematic. Build this circuit on your Toddler board. Note that the 220 ohm resistors are already built into the Toddler PCB; just plug in the infrared LEDs and your Toddler will be ready.

Figure 6.2: Infrared LED and Receiver

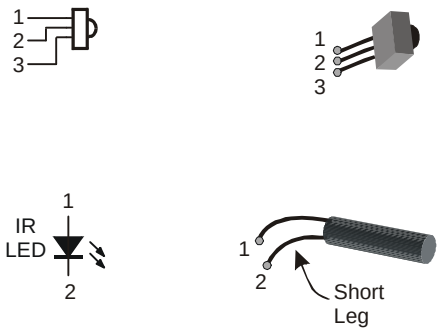


Figure 6.3: Infrared Detector Schematic
Note: the 220 ohm resistors are built into the Toddler PCB

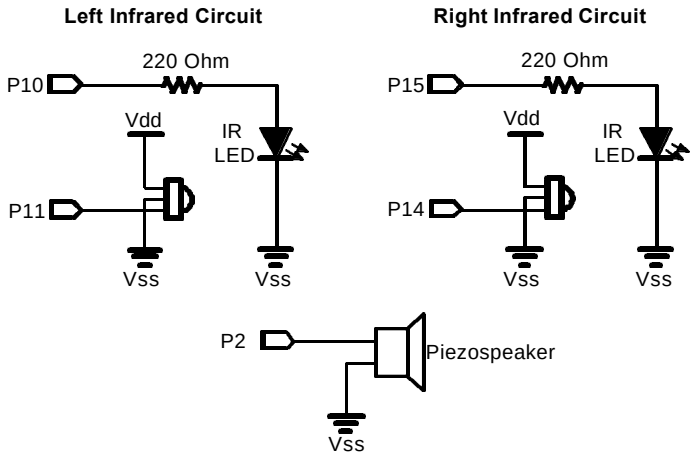
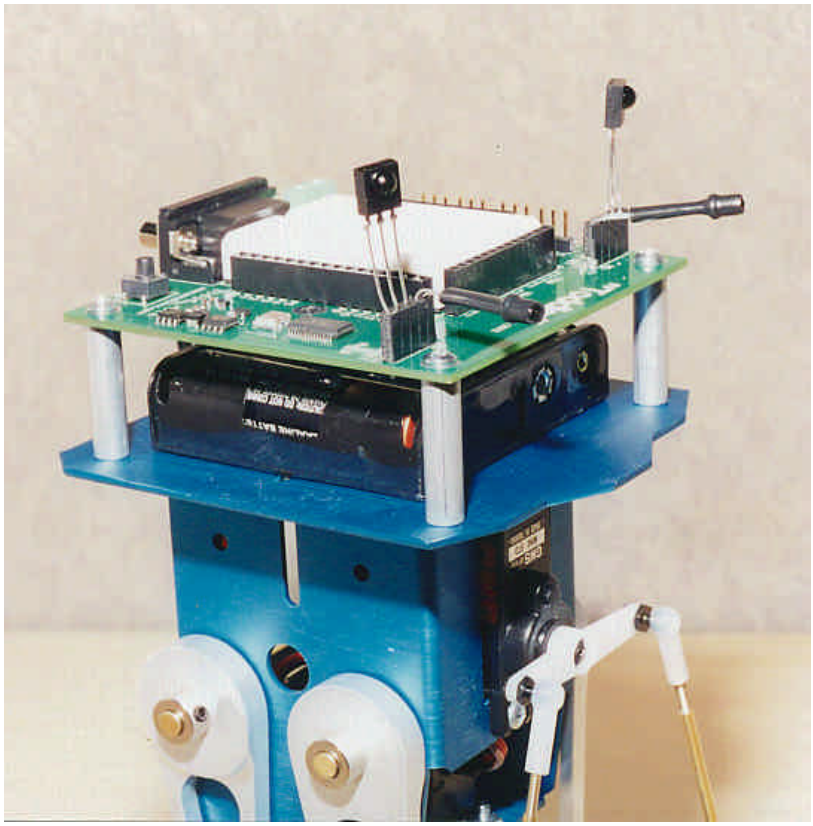


Figure 6.4: Finished Infrared Circuit on Toddler



One IR pair (IR LED and detector) is mounted on each corner of the Toddler breadboard.

Testing the IR Pairs

The key to making each IR pair work is to send 1 ms of unfiltered 38.5 kHz FREQOUT harmonic followed immediately by testing the signal sent by the IR detector and saving its output value. The IR detector's normal output state when it sees no IR signal is high. When the IR detector sees the 38500 Hz harmonic sent by the IR LED, it's output will drop from high to low. Of course, if the IR does not reflect off an object, the IR detector's output simply stays high. Program 6.1 shows an example of this method of reading the detectors.

- Enter and run Program Listing 6.1.

- This program makes use of the BASIC Stamp Editor's DEBUG Terminal, so leave the serial cable connected to the Toddler while Program Listing 6.1 is running.

```
' Toddler Program 6.1: IR Pairs Display
' {$Stamp bs2}                                    ' Stamp Directive.

'----- Declarations --------------

  left_IR_det  var  bit                           ' Two bit variables for saving IR
  right_IR_det var  bit                           ' detector output values.

  left_pin     con    10
  right_pin    con    15

  left_in      var    in11
  right_in     var    in14

'----- Initialization ------------

  output left_pin                                 ' signals to function as outputs
  output right_pin

'----- Main Routine --------------

main:
                                                  ' Detect object on the left.
  freqout left_pin, 1, 38500                      ' Send freqout signal - left IR LED.
  left_IR_det = left_in                           ' Store IR detector output in RAM.
                                                  ' Detect object on the right.
  freqout right_pin, 1, 38500                     ' Repeat for the right IR pair.
  right_IR_det = right_in

  debug home, "Left= ", bin1 left_IR_det
  pause 20
  debug "  Right= ", bin1 right_IR_det
  pause 20

goto main
```

- While program Listing 6.1 is running, point the IR detectors so nothing nearby could possibly reflect infrared back at the detectors. The best way to do this is to point the Toddler up at the ceiling. The DEBUG output should display both left and right values as equal to "1."

- By placing your hand in front of an IR pair, it should cause the DEBUG Terminal display for that detector to change from "1" to "0." Removing your hand should cause the output for that detector to return to a

"1" state. This should work for each individual detector, and you also should be able to place your hand in front of both detectors and make both their outputs change from "1" to "0."

- If the IR Pairs passed all these tests, you're ready to move on; otherwise, check your program and circuit for errors.

How the IR Pairs Display Program Works

Two bit variables are declared to store the value of each IR detector output. The first `FREQOUT` command in the `MAIN` routine is different. The command `FREQOUT left_pin, 1, 38500` sends the on-off pattern shown in Figure 6.2 via left IR LED circuit by causing it to flash on and off rapidly. The harmonic contained in this signal either bounces off an object, or not. If it bounces off an object and is seen by the IR detector, the IR detector sends a low signal to I/O pin `left_in`. Otherwise, the IR detector sends a high signal to `left_in`. So long as the next command after the `FREQOUT` command is the one testing the state of the IR detector's output, it can be saved as a variable value in RAM. The statement `left_IR_det = left_in` checks `left_in`, and saves the value ("1" for high or "0" for low) in the `left_IR_det` bit variable. This process is repeated for the other IR pair, and the IR detector's output is saved in the `right_IR_det` variable. The `DEBUG` command then displays the values in the debug window.

Your Turn

- Experiment with detuning your IR pairs by using frequencies above 38.5 kHz. For example, try 39.0, 39.5, 40.0, 40.5 and 41 kHz. Note the maximum distance that each will detect by bringing an object progressively closer to the IR pairs and noting what distance began to cause the IR detector output to switch from "1" to "0."

Activity #2: Object Detection and Avoidance

The IR pairs provide range information that the Toddler can be use to avoid obstacles. A simple program can simply avoid obstacles providing a random walk around a room without causing a collision. Obstacles must be high enough to be detected by the Toddler's IR detectors.

Real-Time IR Navigation

Program Listing 6.2 checks the IR pairs and delivers one of four different pulses based on the sensors. Each of the navigational routines is just a single pulse in the `Forward`, `Left_turn`, `Right_turn` or `Backward` directions. After the pulse is applied, the sensors are checked again, then another pulse is applied, etc. This program also makes use of some programming techniques you will find very useful.

```
' Toddler Program 6.2:  Object Detection And Avoidance
' {$Stamp bs2}                                  ' Stamp Directive.

' -----[ I/O Definitions ]-------------------------------------------------------

TiltServo             CON           13           ' Tilt servo on P12
StrideServo           CON           12           ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------------

MoveDelay             CON           25           ' in micrcoseconds

TiltStep              CON           10           ' TiltServo step size

RightTilt             CON           620          ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10           ' StrideServo step size

RightStride           CON           650          ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]-------------------------------------------------------------

FigureLoop            VAR           Nib
MoveLoop              VAR           Byte         ' Loop for repeat movements
MoveLoopLimit         VAR           Byte

SubMoveLoop           VAR           Byte         ' Loop for repeat submovements
```

```
SubMoveLoopLimit      VAR           Byte

Pulses                VAR           Word          ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses

Mx                    VAR           Word
MxCurrent             VAR           Word

Sx                    VAR           Word
SxCurrent             VAR           Word


' -----[ Movement Support Codes ]-------------------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL            CON     0
TC            CON     1
TR            CON     2
SL            CON     3
SC            CON     4
SR            CON     5
xx            CON     255


' -----[ Movement Value Tables ]---------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle        DATA          7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle       DATA          7, bRightTurn, bRightTurn, bForward, xx

WalkForward3          DATA          3, bForward, xx
WalkForward8          DATA          8, bForward, xx


' -----[ Basic Movement Codes ]----------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish               CON           0
bForward              CON           1
```

```
bBackward             CON           2
bLeftTurn             CON           3
bRightTurn            CON           4
bPivotLeft            CON           5
bPivotRight           CON           6


' -----[ Basic Movement Tables ]-------------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements        CON           Forward

Forward               DATA          1, TR, SL, TL, SR, xx
Backward              DATA          1, TR, SR, TL, SL, xx

LeftTurn              DATA          1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn             DATA          1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft             DATA          3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight            DATA          3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                DATA          1, TR, SC, TC, xx



'----- Declarations --------------
                                                    ' The lower 2 bits of the
  sensors var nib                                   ' sensors variable are used to store
                                                    ' IR detector values.
  left_pin     con    10
  right_pin    con    15

  left_in      var    in11
  right_in     var    in14

'----- Initialization ------------

  output 2                                          ' Set all I/O lines sending freqout
  output left_pin                                   ' signals to function as outputs
  output right_pin
  freqout 2, 2000, 3000                             ' Program start/restart signal.

  GOSUB ResetCC                                     ' Initialize feet

'----- Main Routine --------------

main:
   FREQOUT left_pin,1,38500                         ' Send freqout signal - left IRLED.
   sensors.bit0 = left_in                           ' Store IR detector output in RAM.
                                                    ' Detect object on the right.
   FREQOUT right_pin,1,38500                        ' Repeat for the right IR pair.
   sensors.bit1 = right_in
```

```
   PAUSE 18                                          ' 18 ms pause(2 ms lost on freqout).

   ' By loading the IR detector output values into the lower 2 bits of the sensors
   ' variable, a number btwn 0 and 3 that the branch command can use is generated.

   LOOKUP sensors,[Backward,PivotLeft,PivotRight,Forward],Mx

   GOSUB Movement
   GOTO main

' -----[ Subroutines ]------------------------------------------------------
'
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                                   ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit                           ' read movement table repeat count
  MxCurrent = Mx + 1

StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                                 ' Mx = start of movement table

 debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                                   ' skip if movement table
    SxCurrent = Mx                                 ' SxCurrent = submovement table index
      GOTO StartSubMovement                        ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent                           ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                                   ' skip if end of list
 debug "  ", hex SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                                   ' lookup submovement table index
StartSubMovement:                                  ' start executing submovement table
```

```
      READ SxCurrent, SubMoveLoopLimit
                                                ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
        Sx = SxCurrent

 debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr

SubMovementLoop:
          READ Sx, Dx                           ' read next submovent action
          Sx = Sx + 1

          IF Dx = xx THEN SubMovementDone
                                                ' skip if end of list
          GOSUB DoMovement                      ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      IF Mx < BasicMovements THEN MovementLoop
                                                ' exit if submovement table
MovementDone:
  NEXT
  RETURN

DoMovement:
 debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                                ' will fall through if invalid index
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT
```

```
  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
RETURN
```

How IR Roaming by Numbers in Real-Time Works

- Look up the LOOKUP command in Appendix C: PBASIC Quick Reference or in the BASIC Stamp Manual.

This Program listing declares the SENSORS variable, which is one nibble of RAM. Of the four bits in the sensors variable, only the lowest two bits are used. Bit-0 is used to store the left detector's output, and bit-1 is used to store the right detector's output.

```
declarations:
  sensors var nib
```

The main routine starts with the FREQOUT commands used to send the IR signals, but the commands following each freqout command are slightly different from those used in the previous program. Instead of saving the bit value at the input pin to a bit variable, each bit value is stored as a bit in the SENSORS variable. Bit-0 of SENSORS is set to the binary value of in8, and bit-1 of the sensors variable is set to the binary value of in0. After setting the values of the lower two bits of the sensors variable, it will have a decimal value between "0" and "3." The BRANCH command uses these numbers to determine to which label it sends the program.

```
main:
   FREQOUT left_pin,1,38500
   sensors.bit0 = left_in

   FREQOUT right_pin,1,38500
   sensors.bit1 = right_in

   PAUSE 18

   LOOKUP sensors,[Backward,PivotLeft,PivotRight,Forward],Mx

   GOSUB Movement
   GOTO main
```

The four possible binary numbers that result are shown in Table 6.1. Also shown is the lookup action that occurs based on the value of the state argument.

Table 6.1: IR Detector States as Binary Numbers

Binary Value of state	Decimal Value of State	What the Value Indicates, Branch Action Based on State
0000	0	`left_in = 0` and `right_in = 0`, Both IR detectors detect object, pulse servos `backward`.
0001	1	`left_in = 0` and `right_in = 1`, Left IR detector detects object, pulse `right_turn`
0010	2	`left_in = 1` and `right_in = 0`, Right IR detector detects object, pulse for `left_turn`
0011	3	`left_in = 1` and `right_in = 1`, Neither IR detector detects object, pulse `forward`.

The `Mx` variable is set to the appropriate movement table index. The `Movement` routine then performs the appropriate sequence of commands.

Challenges

You can rearrange the address labels in the lookup command so that the Toddler does different things in response to obstacles. One interesting activity is to try replacing the `Backward` address with the `Forward` address. There will be two instances of `Forward` in the `Lookup` address list, but this is not a problem. Also, swap the `Left_turn` and `Right_turn` addresses.

- Try making the changes just discussed.

If you stop your hand, the Toddler will run into it. Because of this, one Toddler cannot be programmed to follow another without some way of distance detection. If the one in front stops, the one in back will crash into it. This problem will be fixed as an example in the next chapter.

Experiment #7: Staying on the Table

What's a Frequency Sweep?

In general, a frequency sweep is what you do when checking your favorite radio stations. Set the station for one frequency, and check the output. If you don't like the song that's playing, change the frequency and check the output again.

Activity #1: Testing the Frequency Sweep

The Toddler can be programmed to send different IR frequencies, and to check for object detection at each frequency. By keeping track of the frequencies for which the IR detector reported an object, its distance can be determined. The left axis of the graph in Figure 7.1 shows how the sensitivity of the IR detector's electronic filter decreases as it receives frequencies greater than 38.5 kHz. The filter essentially causes the IR detector to become less able to detect IR at these frequencies. Another way to think about it is that you have to move an object closer if you want it to be detected at a less sensitive frequency. Since the detector is less sensitive, it will take brighter IR (or a closer object) to make the detector see the signal.

Figure 7.1: Relative IR Sensitivity to Frequency

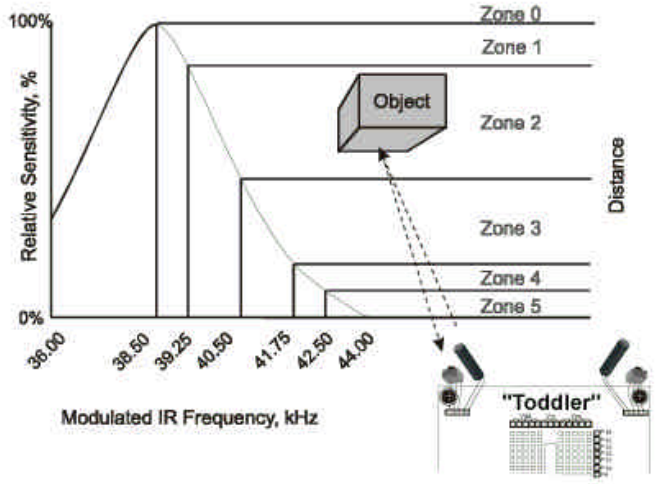


Figure 7.1 compares the left axis of the graph (IR frequency) to the relative sensitivity of the IR detector. The right side of the graph shows how the relative sensitivity of the IR detector relates to distance detection. As detector sensitivity decreases with the increase in frequency, the object must be closer for the IR signal to be detected. Why closer? When the detectors are made less sensitive by sending higher frequencies, it's like giving them darker and darker lenses to look through. Just as a flashlight beam appears brighter when reflected off an object that's closer to you, IR reflected off a closer object appears brighter to the IR detectors.

The right axis of Figure 7.1 shows how different frequencies can be used to indicate in which zone a detected object is located. By starting with a frequency of 38.5 kHz, whether or not an object is in Zone 1-5 can be determined. If an object is not yet detected, it must be beyond the detector limit (Zone 0). If an object is detected, by testing again at 39.25 kHz, the first datum about distance is collected. If 38.5 kHz is detected the object but 39.25 kHz did not, the object must be in Zone 1. If the object was detected at both frequencies, but not at 40.5 kHz, we know it's in Zone 2. If all three frequencies detected the object, but it was not detected at 41.75 kHz, we know it is in Zone 3. If all four frequencies detected the object, but not 42.5 kHz, we know it's in Zone 4. If all the frequencies detected the object, we know it's in Zone 5.

The frequency sweep technique used in this chapter works fairly well for the Toddler, and the components are only a fraction of the cost of common IR distance sensors. The trade off is that the accuracy of this method is also only a fraction of the accuracy of common IR distance sensors. For basic Toddler tasks that require some distance perception, such as following another Toddler, this interesting technique does the trick. Along with adding low-resolution distance perception to the Toddler's senses, it also provides an introduction to the concepts of filters and frequency response.

Build It!

- ❑ Use the same IR detection circuit from Chapter 6, shown in Figure 6.4, for this activity.

Programming the IR Distance Gage

Programming the BASIC Stamp to send different frequencies involves a `FOR...NEXT` loop. The `Counter` variable can be used to give the `FREQOUT` command different frequencies to check. This program introduces the use of arrays. Arrays are used in Program 7.1 to store the IR detector outputs at the different frequencies. For the `l_values` variable, the Zone 0 output is stored in bit-0 of `l_values`. The Zone 1 output is stored in bit-1 `l_values.bit1`, and so on, all the way through Zone 5, which is stored in bit-5 of `l_values`. The same measurements are taken for `r_values`.

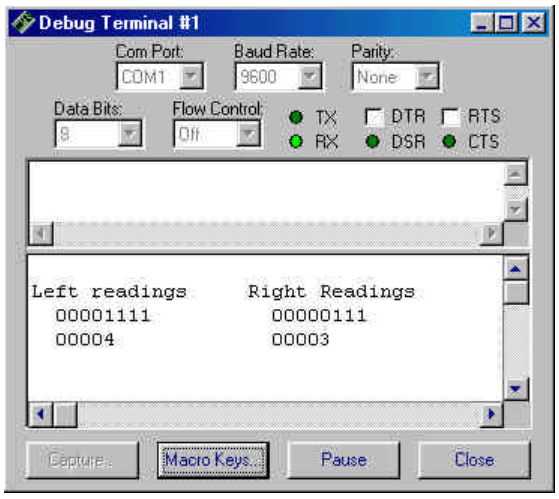
- Enter and run Program Listing 7.1.
- This program makes use of the Debug Terminal, so leave the serial cable connected to the Toddler while Program Listing 7.1 is running.

```
' Toddler Program 7.1: IR Distance Gage
' {$Stamp bs2}                              ' Stamp Directive.

'----- Declarations --------------

  counter             var           nib     ' Multipurpose counting variable.
  l_values            var           byte    ' Two vars for storing left & right
  r_values            var           byte    ' freq sweep IR detector outputs.
  IR_freq             var           word    ' Stores frequency arg for freqout.

  left_pin     con     10
  right_pin    con     15

  left_in      var     in11
  right_in     var     in14

'----- Initialization ------------

  output left_pin                           ' Set all I/O lines sending freqout
  output right_pin                          ' signals to function as outputs.

'----- Main Routine --------------

  main:

    l_values = 0                            ' Reset l_values and r_values to 0.
    r_values = 0

    ' Load sensor outputs into l_values and r_values using a for...next loop,
    ' and a lookup table, and bit addressing.

    for counter = 0 to 4

      lookup counter,[37500,38250,39500,40500,41500], IR_freq

      freqout left_pin,1, IR_freq
      l_values.lowbit(counter) = ~left_in

      freqout right_pin,1, IR_freq
      r_values.lowbit(counter) = ~right_in

    next

    ' Display l_values and r_values in binary and ncd format.

    debug home, cr, cr, "Left readings     Right Readings", cr
```

```
    debug "  ",bin8 l_values, "          ", bin8 r_values, cr
    debug "  ",dec5 ncd(l_values), "             ", dec5 ncd(r_values), cr, cr

  goto main
```

When the Toddler is placed facing a nearby wall (3 to 5 cm.), the Debug Terminal should display something similar to Figure 7.2. As the Toddler is moved closer to and further from the wall, the numbers displayed by the Debug Terminal should change increase and decrease. Each "1" represents a zone so that when you see five 1's the object is nearest to the Toddler.

Figure 7.2: Frequency sweep data in binary and NCD format.



- Place the Toddler so that it faces the wall with its IR LEDs about 1 cm. away from the wall. The left and right readings should both be at "4" or "5." If not, make sure each IR detector is facing in the same direction as its IR LED.

- Gradually back the Toddler away from the wall. As the Toddler is backed away from the wall, the left and right readings should gradually decrease to "0."

- If either or both sides stay at all zeros or all ones, it indicates a possible mistake in either your wiring or in the program. If this is the case, unplug your battery pack from the Toddler. Then, check your wiring and PBASIC code for errors.

The maximum detection distance is 20 to 30 cm., depending on the reflectivity of the wall. Some tinkering with how far left/right each IR pair is pointing may be required to get the numbers to be the same at a given distance. A high level of precision IS NOT necessary for these activities.

✓ TIP

Use a wire stripper to unsheathe about 1 cm. of insulation from a jumper wire. Slide the insulation up one of the IR LED leads. This will protect the leads from touching each other during adjustment.

How the Distance Gauge Program Works

- Look up the LOOKUP command in the BASIC Stamp Manual before continuing.

Counter is a nibble variable that is used to index a FOR...NEXT loop. The FOR...NEXT loop is used for checking the IR detectors at various frequencies. The l_values and r_values variables store the outputs for the left and right IR detectors at the various frequencies used. Each variable stores five binary measurements. Since the IR detector outputs are tested at a variety of frequencies, IR_freq is a variable that can store the value of the frequency that gets sent each time through the frequency testing loop.


```
declarations:

  counter     var          nib
  l_values    var          byte
  r_values    var          byte
  IR_freq     var          word
```

The main routine contains two routines, one for frequency sweep and another for displaying the data collected. The first step in the frequency sweep is setting l_values and r_values to zero. This is important since individual bits in each variable are modified. Clearing l_values and r_values starts each variable with a clean slate. Then individual bits can be set to "1" or "0," depending on what the IR detectors report.

```
main:

  l_values = 0
  r_values = 0
```

The `for…next` loop is where the frequency sweep occurs. The `lookup` command checks the `counter` value to determine which frequency to copy to the `IR_freq` variable. When `counter` is "0," 37500 gets copied to `IR_freq`. When **`counter`** is "1," 38250 is copied to `IR_freq`. As the value of `counter` is incremented from "0" to "4" by the `for...next` loop, each successive value in the `lookup` table is copied to `IR_freq`.

```
for counter = 0 to 4

   lookup counter,[37500,38250,39500,40500,41500],IR_freq
```

Note that the lookup table begins the frequency sweep at 37500 (most sensitive) and ends at 41500 (least sensitive). You might be wondering why the numbers in the lookup table don't match the frequency values from Figure 7.1. It's true that if the BASIC Stamp could transmit a 50% duty cycle pulse train (pulses with the same high time and low time) at these frequencies, they would have to match the frequencies specified for the IR detector's filter. However, the `FREQOUT` command introduces other factors that affect the amplitude of the harmonics transmitted by the IR LEDs. The math involved in predicting the optimum `frequency` arguments to use is very advanced and is well outside the scope of this text. Even so, the best frequencies for a given distance can be determined experimentally. The list of values we are using are known to be reliable.

The left sensor is checked by using `freqout` to send the current value of `IR_freq`. Next, the `.lowbit()` argument is used to address each successive bit in `l_values`. When `counter` is "0," the `.lowbit(counter)` argument addresses bit-0 of `l_values`. When `counter` is "1," the `.lowbit(counter)` argument addresses bit-1 of `l_values`, and so on. Before writing the value of `in8` to `l_values.lowbit(counter)`, the NOT operator (~) is used to invert the bit's value before it is stored to its bit array location in `l_values`. The same process is then repeated for `r_values`. After the fifth time through the `for...next` loop, the IR data bits have all been loaded into `l_values` and `r_values`.

```
   freqout left_pin,1,IR_freq
   l_values.lowbit(counter) = ~left_in

   freqout right_pin,1,IR_freq
   r_values.lowbit(counter) = ~right_in

next
```

The `display` subroutine uses a variety of formatters and text strings to display the `l_values` and `r_values` variables. The first row of the display is the text heading indicating which readings correspond the right IR detector and which readings correspond to the left IR detector. Remember that left and right are treated as though you are sitting in the Toddler's body.

```
display:
  debug home, cr, cr, "Left readings     Right Readings", cr
```

The second row displays `l_values` and `r_values` in binary format. This allows for observation of how the bit values in `l_values` and `r_values` change as the apparent distance of an object changes.

```
debug "  ",bin8 l_values, "          ", bin8 r_values, cr
```

The third row displays the `ncd` value of each variable. The `NCD` operator returns a value that corresponds to the location of the most significant bit in a variable. If the variable is all zeros, `ncd` returns a zero. If the least significant bit contains a "1," and all the rest of the digits are "0," `NCD` returns a "1." If bit-1 contains a "1," but all the numbers to the left of bit-1 are zeros, `ncd` returns a "2," and so on. The `NCD` operator is a handy way of indicating how many ones have been loaded into the lower bits of `l_values` and `r_values`. What's really handy is that `ncd` directly tells you in which zone the object has been detected.

```
debug "  ",dec5 ncd(l_values), "           ", dec5 ncd(r_values), cr, cr
```

When the display routine is finished sending data to the Debug Terminal, program control is returned to the `main` label.

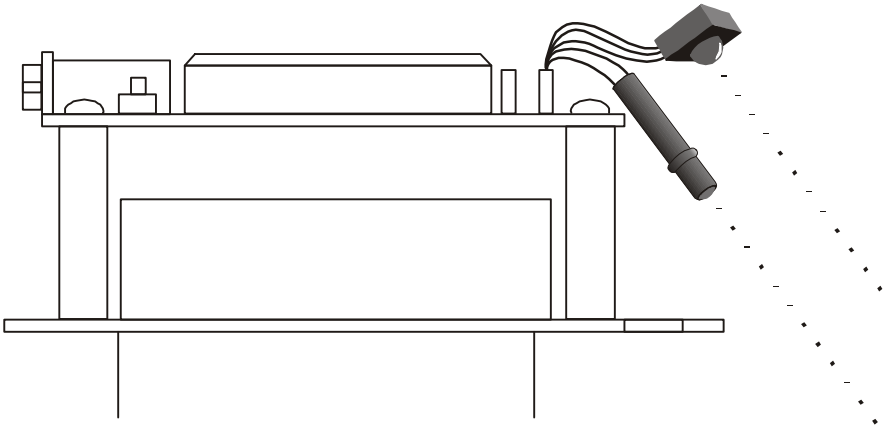
```
goto main
```


Your Turn

- With Program 7.1 running, place the Toddler facing the wall so that the IR LEDs are about 1.5 cm. from the wall. For best results, tape a white sheet of paper to the wall.
- Make a note of the left and right readings.
- Start pulling the Toddler away from the wall.
- Each time the value of one or the other sensors decreases, make a note of the distance. In this way you can determine the zones for each of your Toddler's IR pairs.
- If the readings on one side are consistently larger than the other, you can point the IR LED on the side reporting the larger readings outward a little further. For example, if the left IR pair continually reports higher readings than the right IR pair, try pointing the left IR LED and detector a little further to the left.

Activity #2: The Drop-off Detector

Figure 7.3: IR LED Adjustment for Edge Detection.



One application for distance detection is checking for a drop-off. For example, if the Toddler is navigating on a table, it can change direction if it sees the edge of the table. All you have to do is point the IR pairs downward so that they are both pointing at the table right in front of the Toddler. A distance detection program can then be used to detect that the table is close-up. When the Toddler nears the edge of a table, one or both of the distance detectors will start reporting that they no longer see something close-up. That means it's time to turn away from the abyss. This program works best on a light-colored table. Darker tables will absorb more light and be less useful at reflecting infrared.

- Point your IR pairs at the surface directly in front of the Toddler as shown in Figure 7.3. The IR pairs should be pointed downward at least 45° from horizontal and outward 45° from the Toddler's center line.
- Perform the tests below using Program 7.1 before trying Program 7.2.
- Record the IR pair outputs when the Toddler is looking straight at the table. If the values of the IR pairs when they are looking at your tabletop are "3" or more, it indicates your detectors are seeing what they are supposed to see.
- Record the IR pair outputs when the Toddler is looking off the edge of the table. If these values remain less than "3," the Toddler is ready to try Program Listing 7.2.

- If the Toddler does not give you steady and consistent readings of "3" or more when the Toddler is looking at the table, try first adjusting the direction the IR pairs are pointing. Also, if the Toddler does not consistently register less than "3" when it's looking off the edge of the table, some additional adjustment of the IR pairs also is in order.

- If the sensors report "3" or more while looking at the table and "2" or less when looking off the edge, the Toddler is ready for Program Listing 7.2.

Make sure to be the spotter for your Toddler when running Program Listing 7.2. Always be ready to pick your Toddler up as it approaches the edge of the table it's navigating. If the Toddler tries to drive off the edge, pick it up before it takes the plunge. Otherwise, your Toddler might become a Not-Bot!

When spotting your Toddler while it's avoiding drop-offs, be ready to pick it up from above. Otherwise, the Toddler will see your hands instead of the drop-off and not perform as expected..

Programming for Drop-Off Detection

Program Listing 7.2 uses modified versions of the forward, right_turn, left_turn and backward routines that have been used and reused in every chapter since Chapter #2. The number of pulses in each routine have been adjusted for better performance along a table edge. The `check_sensors` subroutine takes distance measurements by recycling code from Program Listing 7.1: IR Distance Gage.

- Run and test Program Listing 7.2. Remember, always be ready to pick your Toddler up if it tries to run off the table.

```
' Toddler Program 7.2: Drop-off Detection
' {$Stamp bs2}                      ' Stamp Directive.

' -----[ I/O Definitions ]-------------------------------------------------

TiltServo             CON           13            ' Tilt servo on P12
StrideServo           CON           12            ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------

MoveDelay             CON           25            ' in micrcoseconds

TiltStep              CON           10            ' TiltServo step size
```

```
RightTilt             CON           620           ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           880


StrideStep            CON           10            ' StrideServo step size

RightStride           CON           650           ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]----------------------------------------------------------

FigureLoop            VAR           Nib
MoveLoop              VAR           Byte          ' Loop for repeat movements
MoveLoopLimit         VAR           Byte

SubMoveLoop           VAR           Byte          ' Loop for repeat submovements
SubMoveLoopLimit      VAR           Byte

Pulses                VAR           Word          ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses

Mx                    VAR           Word
MxCurrent             VAR           Word

Sx                    VAR           Word
SxCurrent             VAR           Word


' -----[ Movement Support Codes ]----------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL            CON     0
TC            CON     1
TR            CON     2

SL            CON     3
SC            CON     4
SR            CON     5

xx            CON     255
```

```
' -----[ Movement Value Tables ]------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

LeftSemicircle        DATA          7, bLeftTurn,  bLeftTurn,  bForward, xx
RightSemicircle       DATA          7, bRightTurn, bRightTurn, bForward, xx

WalkForward3          DATA          3, bForward, xx
WalkForward8          DATA          8, bForward, xx


' -----[ Basic Movement Codes ]--------------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish               CON           0
bForward              CON           1
bBackward             CON           2
bLeftTurn             CON           3
bRightTurn            CON           4
bPivotLeft            CON           5
bPivotRight           CON           6


' -----[ Basic Movement Tables ]--------------------------------------------------------
'
' These tables can contain Movement Support Codes.

BasicMovements CON            Forward

Forward               DATA          1, TR, SL, TL, SR, xx
Backward              DATA          1, TR, SR, TL, SL, xx

LeftTurn              DATA          1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn             DATA          1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft             DATA          3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight            DATA          3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                DATA          1, TR, SC, TC, xx


'----- Local Declarations --------------

  counter      var     nib                  ' For...next loop index variable.
  l_values     var     Mx                   ' Store R sensor vals for processing.
  r_values     var     Sx                   ' Store L sensor vals for processing.
  l_IR_freq    var     MxCurrent            ' Stores L IR freqs from lookup table.
  r_IR_freq    var     SxCurrent            ' Stores R IR freqs from lookup table.
```

```
  left_pin     con     10
  right_pin    con     15

  left_in      var     in11
  right_in     var     in14


'----- Initialization ------------

  output left_pin                           ' Set all I/O lines sending freqout
  output right_pin                          ' signals to function as outputs.


'----- Initialization ------------

  output 2                                  ' Declare freqout lines to be outputs.
  freqout 2,500,3000                        ' Signal program is starting/restarting.

  GOSUB ResetCC

'----- Main Routine --------------

main:                                       ' Main routine

  ' The command "gosub check_sensors" sends the program to a subroutine that
  ' loads distance values into l_values and r_values.  So, when the program returns
  ' from the check_sensors subroutine, the values are updated and ready for
  ' distance based decisions.

  gosub check_sensors

  ' The distances are checked for four different inequalities.  Depending on the
  ' inequality that turns out to be true, the program either branches to the
  ' forward, left_turn, right_turn or backward navigation routine.
  ' The "3" value used below to test the boundary conditions may need to be
  ' changed depending upon the color of the walking surface and the angle of
  ' IR LEDs and detectors.

  boundary CON 2

  if l_values >= boundary and r_values >= boundary then go_forward
  if l_values >= boundary and r_values <  boundary then left_turn
  if l_values <  boundary and r_values >= boundary then right_turn
  if l_values <  boundary and r_values <  boundary then go_backward

goto main                                   ' Repeat the process.

'----- Navigation Routines -------

   go_forward:                              ' Deliver a single forward pulse, then
      Mx = Forward
      GOSUB Movement
```

```
      goto main                                ' go back to the main: label.

   left_turn:                                  ' Deliver eight left pulses, then
      Mx = PivotLeft
      GOSUB Movement
      goto main                                ' go back to the main: label.

   right_turn:                                 ' Deliver eight right pulses, then
      Mx = PivotRight
      GOSUB Movement
      goto main                                ' go back to the main: label.

   go_backward:                                ' Deliver eight backward pulses, then
      Mx = Backward
      GOSUB Movement
      goto main                                ' go back to the main: label.

'----- Subroutines ---------------

  ' The check sensors subroutine is a modified version of Program Listing 6.1
  ' without the debug Terminal display.  Instead of displaying l_values and
  ' r_values, the main routine uses these values to decide which way to go.

  check_sensors:

    l_values = 0                                    ' Reset l_values and r_values to 0.
    r_values = 0

    ' Load sensor outputs into l_values and r_values using a for...next loop,
    ' a lookup table, and bit addressing.

      for counter = 0 to 4

         check_left_sensors:
            lookup counter,[37500,38250,39500,40500,41500],l_IR_freq
            freqout left_pin, 1, l_IR_freq
            l_values.lowbit(counter) = ~ left_in

         check_right_sensors:
            lookup counter,[37500,38250,39500,40500,41500],r_IR_freq
            freqout right_pin, 1, r_IR_freq
            r_values.lowbit(counter) = ~ right_in

      next

    ' Convert l_values and r_values from binary to ncd format.

      l_values = ncd l_values
      r_values = ncd r_values

    ' Now l_values and r_values each store a number between 0 and 5 corresponding
    ' to the zone the object is detected in.  The program can now return to the
    ' part of the main routine that makes decisions based on these distance
```

```
    ' measurements.

   return


' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                            ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit                    ' read movement table repeat count
  MxCurrent = Mx + 1

StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                          ' Mx = start of movement table

 debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                            ' skip if movement table
    SxCurrent = Mx                          ' SxCurrent = submovement table index
      GOTO StartSubMovement                 ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent                    ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                            ' skip if end of list
 debug "  ", hex SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                            ' lookup submovement table index
StartSubMovement:                           ' start executing submovement table
      READ SxCurrent, SubMoveLoopLimit
                                            ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
        Sx = SxCurrent

 debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr
```

```
SubMovementLoop:
          READ Sx, Dx                         ' read next submovent action
          Sx = Sx + 1

          IF Dx = xx THEN SubMovementDone
                                              ' skip if end of list
          GOSUB DoMovement                    ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      IF Mx < BasicMovements THEN MovementLoop
                                              ' exit if submovement table
MovementDone:
  NEXT
  RETURN

DoMovement:
 debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                              ' will fall through if invalid index
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride
```

```
StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride

MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
RETURN
```

Aliased Variables

The Drop-off Detection program in Program 7.2 is the beginning of a rather large program in terms of DATA memory. In fact, without a little PBASIC programming trick, the program will not compile. The trick is PBASIC's ability to alias a variable so it uses the storage space of another variable. This allows the program to run with the 16 words of RAM space (actually 3 words are used for the BASIC Stamp's PBASIC and interface pin support).

The following code from Program Listing 7.2 shows how the aliasing is done.

```
       '----- Local Declarations --------------

         counter    var    nib
         l_values   var    Mx
         r_values   var    Sx
```

```
l_IR_freq   var    MxCurrent
r_IR_freq   var    SxCurrent
```

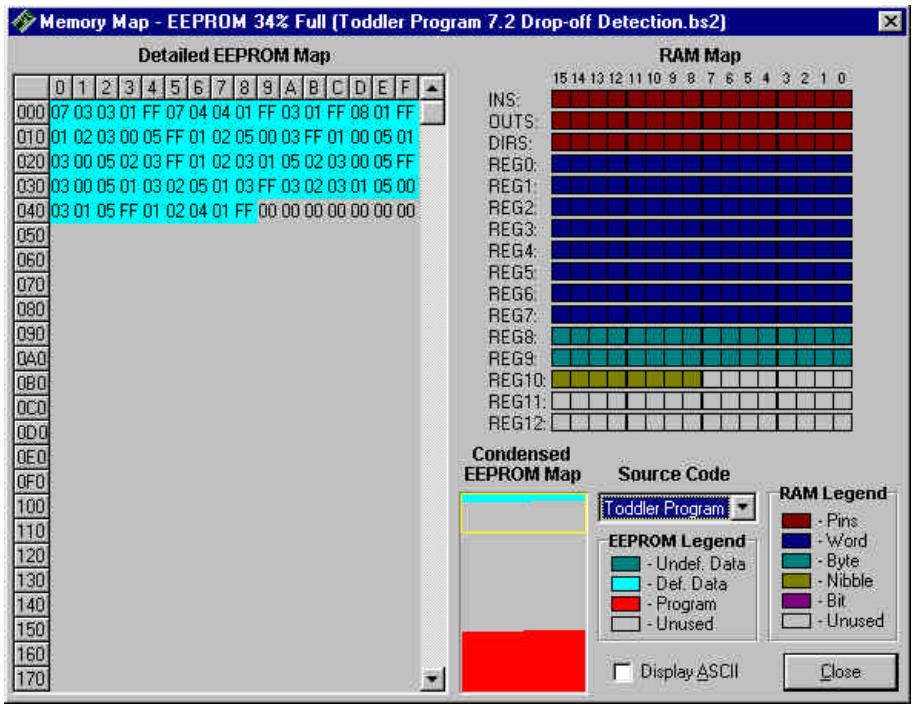
The first `var` definition is normal. It defines a nibble variable. The next four reuse different variables. They are the same size as the aliased variables. The main requirement to keep in mind when using aliased variables is that any variables sharing the same storage that these variables cannot be used at the same time. In other words, do not try the following.

```
l_values = 1
Mx = 2
```

Aliasing is normally used because the original variable names do not work well with a new part of the program or subroutine. PBasic has no concept of local variables so aliasing is required.

The BASIC Stamp's IDE can present the memory map of the current program. This provides RAM and EEPROM usage information. The memory map for the Toddler Program 7.2 is shown in Figure 7.4. It shows 5 bytes of free RAM. Not much but enough. This inlcudes the use of four word aliased variables. If these variables were not aliased then the program would need additional 8 bytes, 3 more than available.

Figure 7.4: EEPROM Memory Map for Toddler Program 7.2



Aliasing should be used with great care. It is a significant source of problems when debugging a program. The advantage of using this with the BASIC Stamp is that only a limited number of variables will be used in the program so it is readily apparent where problems occur.

In this case, the initial set of variables is including `Mx` are used in the movement part of the program. Only the `Mx` variable is used outside of the routine `Movement` routine and that is used to pass a parameter to the routine. The aliased variables including `l_IR_freq` variable is used in the range finding routine. Since these two routines do not call each other it is easy to isolate the two with respect to variables.

How the Drop-off Avoidance Program Works

Now that we have the aliasing issue out of the way we can move onto the main program. The first thing the `main` routine does is call the `check_sensors` subroutine. Note that `check_sensors` is simply Program 7.1 with no Debug Terminal display placed in a subroutine. Instead of debugging the `NCD` values of `l_detect` and `r_detect`, the values of these two variables are simply converted to `NCD` values using the statements:

```
l_values = ncd l_values
```

and

```
r_values = ncd r_values
```

After calling the `check_sensors` subroutine, `l_values` and `r_values` are numbers between "0" and "5." These values are used instead of the "1" and "0" values used in the whiskers program. After the program returns from the `check_sensors` subroutine, `l_values` and `r_values` are checked against the benchmarks distance indicating the edge of the table has been detected.

```
boundary CON 2

  if l_values >= boundary and r_values >= boundary then go_forward
  if l_values >= boundary and r_values <  boundary then left_turn
  if l_values <  boundary and r_values >= boundary then right_turn
  if l_values <  boundary and r_values <  boundary then go_backward
```

The routines then load the `Mx` variable with the index of the appropriate table. The `Movement` routine then uses the table to initiate the Toddler's leg movements. The `boundary` value is the distance boundary condition. This may need to be changed depending upon the color of the surface the Toddler is walking on. It must be set so that the Toddler reliably sees the table when moving forward.

The angle at which the IR LEDs and sensors can be tilted downward is limited so a low `boundary` value is typical. One alternative to having a value of 1 or 2 is to adjust the range finding frequencies so that the midrange values are sensing distances farther away. The other alternative is to mount the IR LEDs and sensors closer or on to the Toddler's feet.

The current configuration with the IR KEDs and sensors mounted on the Toddler's circuit board does lead to a long rang recognition of the edge of the table so the Toddler should not get much closer than a foot from the edge. This means the Toddler needs a relatively large table with a white or light colored surface to walk on.

The Toddler will also attempt to walk along the edge of the table although this is not explicitly built into the program. In theory, if the Toddler walks perpendicular to the edge it will walk to the edge, back up, walk forward and repeat this indefinitely. In practice, this does not occur for two reasons. The first is that the Toddler's movement is not perfectly repeatable. As it moves forward and backwards, the Toddler turns slightly to the one side or the other. Eventually the IR sensors will detect the difference and the Toddler will turn instead of backing up or moving forward.

The sensors themselves are another area that will cause the Toddler to turn parallel to the edge of the able. This will occur if one sensor is more sensitive than the other. Of course, this difference will work in one direction and may cause the Toddler to take an extra step forward if the detection is handled by the other side. This will not cause the Toddler to walk off the table though since it tries to stay so far away from the edge. An extra step or two will not cause a problem.

One area that can be a problem especially when the IR LEDs and sensors are pointed forward is that the Toddler will have limited peripheral vision. It is possible for the Toddler to turn parallel to an edge and drift towards the edge. In theory, the sensor on that side should detect the edge and the Toddler will turn away from the edge. This problem occurs more often when the edge of the table is irregular. Aiming the IR LEDs and sensors outward slightly can help eliminate the problem if it occurs.

Activity #3: Toddler Shadow Walker

For one Toddler to follow another, the Toddler that follows, a.k.a. the shadow walker, has to know how far the lead vehicle is ahead. If the shadow vehicle is lagging behind, it has to detect this and speed up. If the shadow vehicle is too close to the lead vehicle, it has to detect this as well and slow down. If it's the right distance, it can wait until the measurements indicate it's too far or too close again.

Unlike the Toddler's sibling the Boe-Bot, the Toddler moves in discrete steps, not small increments using wheels. Whereas the Boe-Bot uses calculated proportional control, the Toddler must be a bit more discrete. It is possible to take proportional steps but the accuracy of the Toddler's movements minimizes the effect of minor changes to movements. On the other hand, the Boe-Bot can move one or both of its wheels a fraction of an inch in subsecond times. A Toddler step can take as long as a second.

As it turns out, the Toddler's IR range finders work well for tracking another Toddler. The range results are in discrete values and the number is not large. If it were, then the values would have to be converted down to this level of gradation that is manageable. It is then simply a matter of choosing the appropriate step type and magnitude.

The Toddler is a difficult target for another Toddler to locate with its many facets. To improve the detection using the IR sensors, the target Toddler should have a white box placed around it. This can be made of paper or cardboard and it can be affixed to the Toddler's frame using tape or other means. The box should start about where the base of the Toddler's central box containing the servos and can extend to just above the circuit board. The IR sensors can be angled down slightly so they will detect the central portion of the box at a distance of about a foot. The box should not impede the foot movement or the servors and it can extend out from the Toddler by as much as a few inches. It should not be too heavy or large so as to significantly change the center of gravity forcing adjustments in walking behavior.

Although these changes are not absolutely required for one Toddler to follow another, they will improve the overall system performance. Also, the roaming area needs to be free of obstacles and walls otherwise the Toddler that will be following may detect these obstacles instead. There is no checking in the program to determine if the object detected is remaining stationary although it is possible to modify the program to do so.

Programming the Toddler Shadow Walker

Program Listing 7.3 uses additional `branch` and `lookup` statements to adjust the Toddler's position based on the range finder results. The movements are designed to aim the Toddler at the object it is following, usually another Toddler, and to keep that object at a discrete distance.

- Run Program Listing 7.3.
- Point the Toddler at an 8 ½ × 11" sheet of paper held in front of it as though it's a wall-obstacle. The Toddler should maintain a fixed distance between itself and the sheet of paper.
- Try moving the paper so it rotates about the Toddler. The Toddler should rotate with it.
- Try using the sheet of paper to lead the Toddler around. The Toddler should follow it.

```
' Toddler Program 7.3: Shadow Walker
' {$Stamp bs2}                            ' Stamp Directive.

' -----[ I/O Definitions ]-------------------------------------------------

TiltServo             CON           13             ' Tilt servo on P12
StrideServo           CON           12             ' Stride servo on P13


' -----[ Constants ]-------------------------------------------------------

MoveDelay             CON           25             ' in micrcoseconds

TiltStep              CON           20             ' TiltServo step size

RightTilt             CON           630            ' Tilt limits
CenterTilt            CON           750
LeftTilt              CON           870


StrideStep            CON           20             ' StrideServo step size

RightStride           CON           650            ' Stride limits
CenterStride          CON           750
LeftStride            CON           850


' -----[ Variables ]-------------------------------------------------------

FigureLoop            VAR           Nib
MoveLoop              VAR           Byte           ' Loop for repeat movements
MoveLoopLimit         VAR           Byte

SubMoveLoop           VAR           Byte           ' Loop for repeat submovements
SubMoveLoopLimit      VAR           Byte

Pulses                VAR           Word           ' Pulse variable

CurrentTilt           VAR           Word
CurrentStride         VAR           Word
NewValue              VAR           Word

Dx                    VAR           Pulses

Mx                    VAR           Word
MxCurrent             VAR           Word

Sx                    VAR           Word
SxCurrent             VAR           Word


' -----[ Movement Support Codes ]------------------------------------------
```

```
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.

TL              CON     0
TC              CON     1
TR              CON     2

SL              CON     3
SC              CON     4
SR              CON     5

xx              CON     255


' -----[ Movement Value Tables ]------------------------------------------------------------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section

TurnLeftForward         DATA            1, bLeftTurn,  bForward, xx
TurnRightForward        DATA            1, bRightTurn, bForward, xx

PivotLeftForward        DATA            1, bPivotLeft,  bForward, xx
PivotRightForward       DATA            1, bPivotRight, bForward, xx

BackwardPivotLeft       DATA            1, bBackward, bPivotLeft,  xx
BackwardPivotRight      DATA            1, bBackward, bPivotRight, xx

Forward2                DATA            2, bForward, xx
Backward2               DATA            2, bBackward, xx


' -----[ Basic Movement Codes ]-------------------------------------------------------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.

bFinish                 CON             0
bForward                CON             1
bBackward               CON             2
bLeftTurn               CON             3
bRightTurn              CON             4
bPivotLeft              CON             5
bPivotRight             CON             6


' -----[ Basic Movement Tables ]-------------------------------------------------------
'
' These tables can contain Movement Support Codes.
```

```
BasicMovements        CON           Forward

Nop                   DATA          1, xx

Forward               DATA          1, TR, SL, TL, SR, xx
Backward              DATA          1, TR, SR, TL, SL, xx

LeftTurn              DATA          1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn             DATA          1, TR, SL, TC, SR, TR, SL, TL, SR, xx

PivotLeft             DATA          3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight            DATA          3, TR, SL, TC, SR, TL, SL, TC, SR, xx

Finish                DATA          1, TR, SC, TC, xx

'----- Movement LOOKUP entries  --------------
'
' These constants should reference the appropriate movement table.
' The constant syntax is lxry where x and y indicate the range from the
' left and right sensor respectively. A zero value indicates nothing
' is within range while a 5 indicates an object is within inches.
' In general, a 3 will be the closest desirable distance.

l0r0    CON     Forward
l0r1    CON     TurnRightForward
l0r2    CON     PivotRightForward
l0r3    CON     PivotRight
l0r4    CON     RightTurn
l0r5    CON     BackwardPivotRight

l1r0    CON     PivotLeftForward
l1r1    CON     Forward
l1r2    CON     PivotRightForward
l1r3    CON     PivotRight
l1r4    CON     PivotRight
l1r5    CON     BackwardPivotRight

l2r0    CON     TurnLeftForward
l2r1    CON     TurnLeftForward
l2r2    CON     Forward
l2r3    CON     Nop
l2r4    CON     PivotRight
l2r5    CON     BackwardPivotRight

l3r0    CON     PivotLeft
l3r1    CON     PivotLeft
l3r2    CON     Nop
l3r3    CON     Nop
l3r4    CON     Nop
l3r5    CON     BackwardPivotRight

l4r0    CON     BackwardPivotLeft
```

```
l4r1    CON     BackwardPivotLeft
l4r2    CON     PivotLeft
l4r3    CON     Nop
l4r4    CON     Backward
l4r5    CON     Backward

l5r0    CON     BackwardPivotLeft
l5r1    CON     BackwardPivotLeft
l5r2    CON     BackwardPivotLeft
l5r3    CON     BackwardPivotLeft
l5r4    CON     Backward
l5r5    CON     Backward


'----- Local Declarations --------------

  counter      var     nib                   ' For...next loop index variable.
  l_values     var     Mx                    ' Store R sensor vals for processing.
  r_values     var     Sx                    ' Store L sensor vals for processing.
  l_IR_freq    var     MxCurrent             ' Stores L IR freqs from lookup table.
  r_IR_freq    var     SxCurrent             ' Stores R IR freqs from lookup table.

  left_pin     con     10
  right_pin    con     15

  left_in      var     in11
  right_in     var     in14


'----- Initialization ------------

   output 2                            ' Declare outputs.
   output left_pin
   output right_pin

   freqout 2,500,3000                  ' Beep at startup.

   GOSUB ResetCC

'----- Main Routine --------------

main:                                  ' Main routine

  gosub check_sensors                  ' Get distance values for each sensor

 'debug "l",dec l_values,"r", dec r_values,cr

  Branch l_values,[left0,left1,left2,left3,left4,left5]

  left0:
     LOOKUP r_values,[l0r0,l0r1,l0r2,l0r3,l0r4,l0r5],Mx
     GOTO main_movement
```

```
  left1:
     LOOKUP r_values,[l1r0,l1r1,l1r2,l1r3,l1r4,l1r5],Mx
     GOTO main_movement

  left2:
     LOOKUP r_values,[l2r0,l2r1,l2r2,l2r3,l2r4,l2r5],Mx
     GOTO main_movement

  left3:
     LOOKUP r_values,[l3r0,l3r1,l3r2,l3r3,l3r4,l3r5],Mx
     GOTO main_movement

  left4:
     LOOKUP r_values,[l4r0,l4r1,l4r2,l4r3,l4r4,l4r5],Mx
     GOTO main_movement

  left5:
    LOOKUP r_values,[l5r0,l5r1,l5r2,l5r3,l5r4,l5r5],Mx

main_movement:
  GOSUB Movement
  GOTO main                                  ' Infinite loop.


'----- Subroutine(s) -------------

check_sensors:

  l_values = 0                               ' Set distances to 0.
  r_values = 0
  ' Take 5 measurements for distance at each IR pair.  If you fine tuned your
  ' frequencies in Activity #2, insert them in the lookup tables.

  for counter = 0 to 4
    check_left_sensors:
      lookup counter,[37500,38250,39500,40500,41000],l_IR_freq
      freqout left_pin,1,l_IR_freq
      l_values.lowbit(counter) = ~left_in

    check_right_sensors:
      lookup counter,[37500,38250,39500,40500,41000],r_IR_freq
      freqout right_pin,1,r_IR_freq
      r_values.lowbit(counter) = ~right_in

  next

  l_values = ncd l_values            ' Value (0 to 5) for distance depending on MSB.
  r_values = ncd r_values

return

' ----- Movement: Move feet using DATA table referenced by Mx -----
'
```

```
' Input: Mx = movement table index, table ends in xx
'    or
'        Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.

Movement:
  IF Mx < BasicMovements THEN SetupMovement

  MxCurrent = Mx                    ' setup to use submovement table
  MoveLoopLimit = 1
  GOTO StartMovement

SetupMovement:
  READ Mx, MoveLoopLimit            ' read movement table repeat count
  MxCurrent = Mx + 1

StartMovement:
  FOR MoveLoop = 1 to MoveLoopLimit
    Mx = MxCurrent                  ' Mx = start of movement table

 'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr

    IF Mx < BasicMovements THEN MovementLoop
                                    ' skip if movement table
    SxCurrent = Mx                  ' SxCurrent = submovement table index
      GOTO StartSubMovement         ' enter middle of loop

MovementLoop:
      READ Mx, SxCurrent            ' read next submovment byte
      Mx = Mx + 1
      IF SxCurrent = xx THEN MovementDone
                                    ' skip if end of list
 'debug "  ", hex SxCurrent, " movement",cr
      LOOKUP
SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,PivotLeft,PivotRight],SxCurrent
                                    ' lookup submovement table index
StartSubMovement:                   ' start executing submovement table
      READ SxCurrent, SubMoveLoopLimit
                                    ' read submovement table repeat count
      SxCurrent = SxCurrent + 1

      FOR SubMoveLoop = 1 to SubMoveLoopLimit
        Sx = SxCurrent

 'debug "   ", hex Sx, " submovement ", dec SubMoveLoop, " of ", dec SubMoveLoopLimit,cr

SubMovementLoop:
         READ Sx, Dx                ' read next submovent action
         Sx = Sx + 1

         IF Dx = xx THEN SubMovementDone
                                    ' skip if end of list
```

```
          GOSUB DoMovement           ' execute movement
          GOTO SubMovementLoop

SubMovementDone:
      NEXT
      IF Mx < BasicMovements THEN MovementLoop
                                     ' exit if submovement table
MovementDone:
  NEXT
  RETURN

DoMovement:
 'debug "    ", dec Dx, " action",cr
  BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,StrideRight]
                                     ' will fall through if invalid index
  RETURN


' ----  Movement routines can be called directly ----

TiltLeft:
  NewValue = LeftTilt
  GOTO MovementTilt

TiltCenter:
  NewValue = CenterTilt
  GOTO MovementTilt

TiltRight:
  NewValue = RightTilt

MovementTilt:
  FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
    PULSOUT TiltServo,   Pulses
    PULSOUT StrideServo, CurrentStride
    PAUSE MoveDelay
  NEXT

  CurrentTilt = NewValue
  RETURN


StrideLeft:
  NewValue = LeftStride
  GOTO MovementStride

StrideCenter:
  NewValue = CenterStride
  GOTO MovementStride

StrideRight:
  NewValue = RightStride
```

```
MovementStride:
  FOR Pulses = CurrentStride TO NewValue STEP StrideStep
    PULSOUT TiltServo,   CurrentTilt
    PULSOUT StrideServo, Pulses
    PAUSE MoveDelay
  NEXT

  CurrentStride = NewValue
  RETURN


' ----- Move feet to initial center position -----

ResetCC:
  CurrentTilt   = CenterTilt
  CurrentStride = CenterStride

  FOR Pulses = 1 TO 100 STEP StrideStep
    PULSOUT TiltServo, CenterTilt
    PULSOUT StrideServo, CenterStride
    PAUSE MoveDelay
  NEXT

DoReturn:
RETURN
```


How the Shadow Walker Program Works

The first thing the `main` routine does is call the `check_sensors` subroutine. After the `check_sensors` subroutine is finished, `l_values` and `r_values` each contain a number corresponding to the zone in which an object was detected for both the left and right IR pairs.

```
main:

  gosub check_sensors
```

The next line of code jumps to one of many `LOOKUP` statements. The `BRANCH` statement uses the status of the left IR sensor while the `LOOKUP` statements use the status of the right IR sensor. These set the `Mx` variable with the table index for the movement to be performed by the `Movement` routine.

```
Branch l_values,[left0,left1,left2,left3,left4,left5]

  left0:
     LOOKUP r_values,[l0r0,l0r1,l0r2,l0r3,l0r4,l0r5],Mx
     GOTO main_movement

  left1:
```

```
         LOOKUP r_values,[l1r0,l1r1,l1r2,l1r3,l1r4,l1r5],Mx
         GOTO main_movement

      left2:
         LOOKUP r_values,[l2r0,l2r1,l2r2,l2r3,l2r4,l2r5],Mx
         GOTO main_movement

      left3:
         LOOKUP r_values,[l3r0,l3r1,l3r2,l3r3,l3r4,l3r5],Mx
         GOTO main_movement

      left4:
         LOOKUP r_values,[l4r0,l4r1,l4r2,l4r3,l4r4,l4r5],Mx
         GOTO main_movement

      left5:
        LOOKUP r_values,[l5r0,l5r1,l5r2,l5r3,l5r4,l5r5],Mx

   main_movement:
     GOSUB Movement
```

The values used in the LOOKUP Statement are defined near the start of the program using CON constant definitions. While it is possible to put these values in the LOOKUP statement, this makes the statements long. It also makes it difficult to see what action is performed in a particular state. The constant definitions provide a way to do this. It is now easy to correlate a particular state such as `l3r3` with a particular movement, in this case a `nop` or no movement. Likewise, `l5r5` indicates that the Toddler is immediately in front of an obstacle and `l0r0` indicates the Toddler has not located an obstacle within its range.

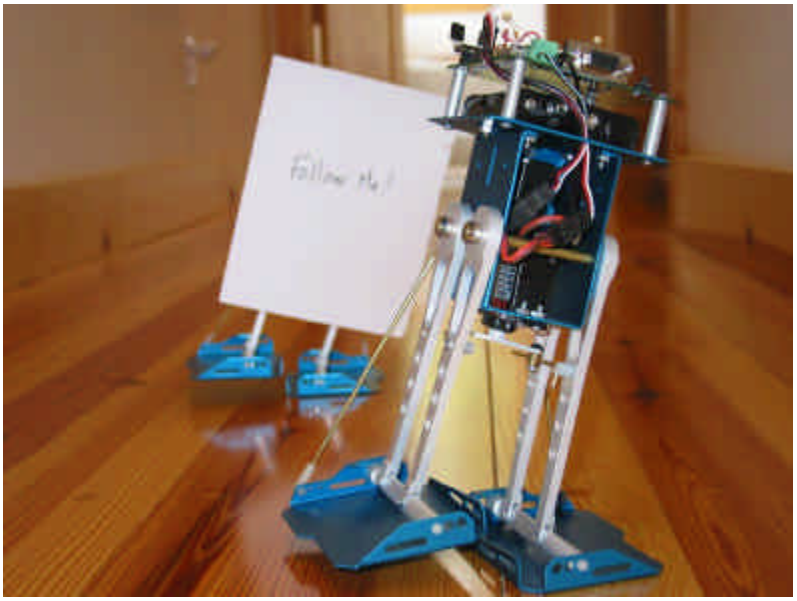
Program control is returned to the `main:` label after the movement has been performed, and the loop repeats itself.

```
      goto main
```

Challenges

Figure 7.5 shows a lead Toddler followed by a shadow Toddler. The lead Toddler could run any of the prior programs provided the speed is slower (increase the PAUSE values or decrease the Step values) and the shadow Toddler is running Program Listing 7.3: Shadow Vehicle. Proportional control makes the shadow Toddler a very faithful follower. One lead Toddler can string along a chain of 2 or 3 Toddlers. Just add a 4' x 4" paper to the lead Toddler's backside.

Figure 7.5: Lead Toddler and Shadow Toddler



- If you are part of a class, mount paper panel on the back of the lead Toddler as shown in Figure 7.5.
- If you are not part of a class (and only have one Toddler) the shadow vehicle will follow a piece of paper or your hand just as well as it follows a lead Toddler.

- The Shadow Toddler should be running Program Listing 7.3 without any modifications.

- With both Toddlers running their respective programs, place the shadow Toddler behind the lead Toddler. The shadow Toddler follows at a fixed distance, so long as it is not distracted by another object such as a hand or a nearby wall.

Appendix A: Parts Listing and Sources

All parts used in the Toddler kit are available individually from the Parallax Component Shop (www.parallaxinc.com/componentshop). If you can't readily find the component you are looking for in the Component Shop enter the name of it in the on-line search box using the stock code.

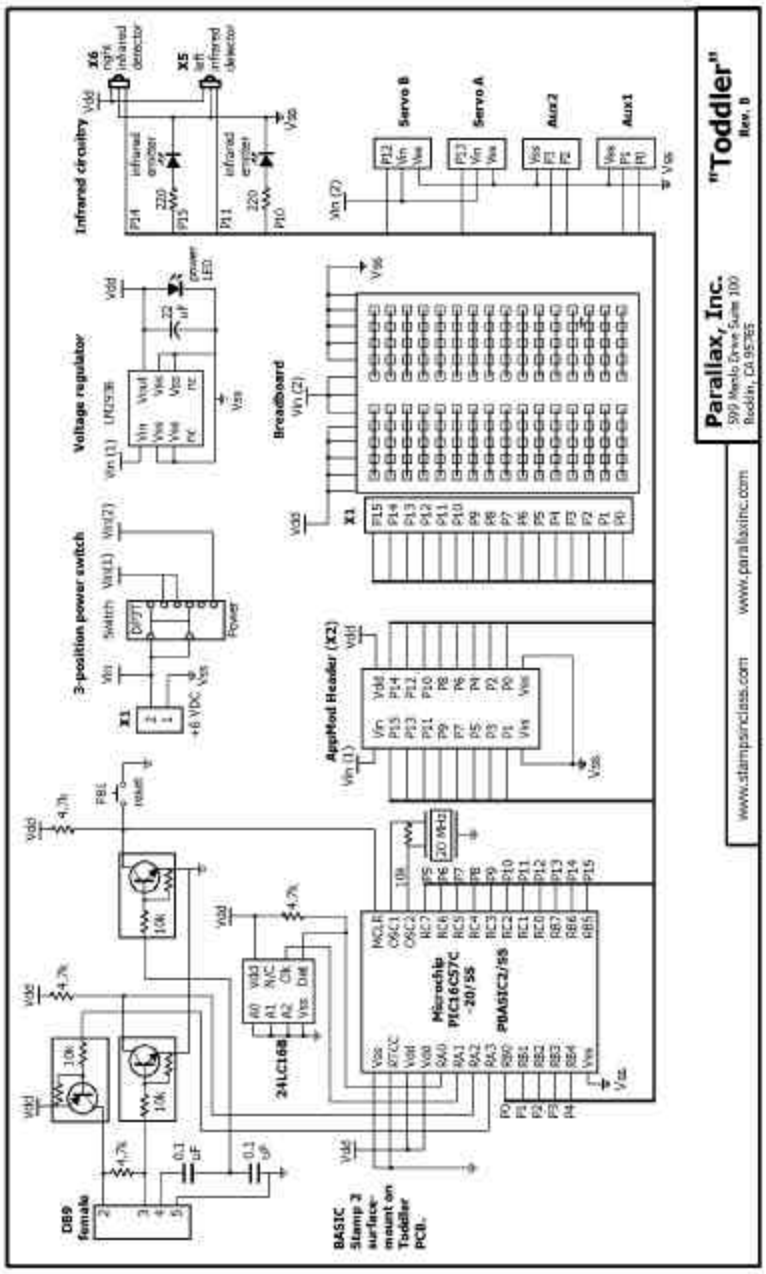
The Toddler Kit comes in two flavors:

- Gold Anodized Toddler Kit - 27310
- Blue Anodized Toddler Kit - 27311

Parallax Part #	Description	Qty/Kit
Electronic Components		
150-02210	220 Ohm resistors ¼ watt 5% tolerance	4
200-01040	0.1 uF capacitor	2
200-01031	0.01 uF capacitor	2
350-00001	Green LED	2
350-00007	Yellow LED	2
350-00009	Photoresistors	2
350-00014	Infrared detector	4
350-00017	Infrared LED w/ heat shrink tubing	4
550-00020	Toddler Printed Circuit Board with BASIC Stamp 2	1
753-00001	Battery Pack with Tinned Wires	1
800-00016	Jumper wires (bag of 10)	1
900-00001	Speaker	1
900-00010	Parallax Toddler Servo (Toddler Mini F BB)	2
Metal Parts		
720-00001	Toddler Top Plate - Gold Anodized	1
720-00002	Toddler Top Plate - Blue Anodized	1
720-00003	Toddler Body - Gold Anodized	1
720-00004	Toddler Body - Blue Anodized	1
720-00005	Toddler Foot - Left Gold Anodized	1
720-00006	Toddler Foot - Left Blue Anodized	1
720-00007	Toddler Foot - Right Gold Anodized	1
720-00008	Toddler Foot - Right Blue Anodized	1
720-00009	Toddler Ankle	2
720-00010	Toddler Legs	4

Hardware		
700-00002	3/8" 4/40 machine screw – panhead	10
700-00003	4/40 nut	14
700-00016	3/8" 4/40 machine screw – flathead	6
700-00028	1/4" 4/40 machine screw – panhead	12
700-00060	1" 4/40 aluminum standoffs female/female	4
710-00100	3/16" long 4/40 socket head cap screw – nylon	4
712-00001	1/2" outer diameter flat round plastic washer	4
725-00002	3" long 3/16" outer diamter brass rod	2
725-00003	1/16" ball joints with 2/56 thread (nut, ball joint, cup)	4
725-00004	5.4" long brass rod with 2/56 0.5" thread on each end	2
725-00005	3/32" hex L-key	1
725-00006	3/32" E/Z adjust plastic horn bracket for 4-40 screw	1
725-00007	.072" brass servo horn connector (brass fitting, rubber holder and small screw) – in a package	1
725-00008	.072" diameter "L" shaped 2" wire	2
726-00001	3/16" collars (4), set screw and wrench	1
Miscellaneous		
27218	BASIC Stamp Manual Version 2.0c	1
122-00001	Advanced Robotics with the Toddler Manual	1
123-00001	Toddler Printed Insert	1
800-00003	Serial cable	1
900-00007	The Plastic Box	1
700-00064	Parallax Screwdriver	1

Appendix B:
Toddler Revision B Printed Circuit Board Schematic


"Toddler"
Rev. B
Parallax, Inc.
www.stampsinclass.com
www.parallaxinc.com
Infrared circuitry
Voltage regulator
3-position power switch
Breadboard
AppMod Header (X2)
Servo B
Servo A
Aux2
Aux1
Microchip PIC16C57C -20/SS
PBASIC2/SS
24LC168
DB9 female
BASIC Stamp 2 surface-mount on Toddler PCB.