

Robotics!

Student Guide for Experiments #1 through #4

Version 1.2

Note regarding accuracy of this text:

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

PARALLAX 7

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 1999 by Parallax, Inc. BASIC Stamp is a registered trademark of Parallax, Inc. If you decided to use the name BASIC Stamp on your web page or in printed material, you must state that "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.

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
Ftp: [ftp.parallaxinc.com](ftp://ftp.parallaxinc.com) and [ftp.stampsinclass.com](ftp://ftp.stampsinclass.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. To subscribe to the BASIC Stamp list, send e-mail to majordomo@parallaxinc.com and write *subscribe stamps* in the body of the message. This list generates about 40 messages per day.

The Stamps in Class list is for students and educators who wish to share educational ideas. To subscribe to this list go to <http://www.stampsinclass.com> and look for the E-groups list. This list generates about 5 messages per day.

Table of Contents

Preface	3
Audience and Teacher's Guides	3
Copyright and Reproduction.....	4
Special Contributors	4
Experiment #1: Boe-Bot Construction	5
Parts Required.....	6
Step #1: Servo Modification and Testing	7
Step #2: Mounting Servos and Wheels on Boe-Bot Platform	12
Step #3: Install Tail Wheel and Battery Holder	14
Step #4: Mounting the Board of Education and Connecting the Servos	15
Step #5: Programming the Boe-Bot for Basic Movement Pattern	18
Challenge!.....	20
Experiment #2: Basic Movement Using Subroutines and Memory.....	22
Pulsout Command and Servo Control	23
Parts Required.....	24
Build It.....	25
Program It.....	26
Sound Feedback	26
LED Feedback.....	27
Distance Calibration	27
Making Turns.....	29
Goto Statement	29
Gosub: A Close Relative of Goto.....	30
Using the Data Command and EEPROM to Store Movements.....	32
All Together Now	33
Challenge	36
Experiment #3: Following Light	38
Parts Required.....	39
Build It.....	39
Program It.....	41
Using the Photoresistor	41
Challenge!.....	46

Contents

Experiment #4: Infrared Object Detection.....	47
Infrared Transmission	47
Infrared Reception	48
Build It.....	48
Program It.....	50
Infrared Transmit and Receive	50
Troubleshooting	55
Challenge!	56
Appendix A: Parts Listing and Sources.....	57
Appendix B: Resistor Color Code	63
Appendix C: Changing the Board of Education Voltage Regulator	65
Appendix D: Boe-Bot Competition Rules	67

Preface

The goal of the Robotics curriculum is to show students how easy it is to become interested and excited about the fields of science, information and engineering as they design, construct and program an autonomous robot. The Board of Education Robot (Boe-Bot) provides students with a project area to build and customize their own mechanical, electrical, and programming project. The use of a robot to introduce microcontroller circuits and interfacing is ideal since the outputs are almost entirely visible and easy to customize.

The Board of Education may be taken off of the Boe-Bot base and used for other Stamps in Class curriculum experiments. This portability saves the class from purchasing another set of hardware and reduces the cost of exploring robotics. The Board of Education was not originally designed for use on a robot (Boe-Bot was created in response to customer demand), so you will notice one or two work-arounds that wouldn't exist if a robot had been considered when the Board of Education was conceived. Specifically, the servos use the unregulated 6 V power supply from the Vin instead of the regulated Vdd. In conjunction with a 3300 uF capacitor between Vss and Vdd this prevents resets on the BASIC Stamp. Older boards may also need to have their voltage regulator swapped out for the new LM2940 low-dropout regulator. Details for this are shown in Appendix C, and the replacement parts are free from Parallax.

The Robotics curriculum will be revised and updated based on feedback from students and educators. If you would like to author an addition to this curriculum, or have ideas for improvements, please send them to stampsinclass@parallaxinc.com. We'll do our best to integrate your ideas and assist you with whatever technical support, sales support, or on-site training you need. If we accept your Boe-Bot project we'll send you a free Boe-Bot.

Audience and Teacher's Guide

The Robotics curriculum was created for ages 17+ as a subsequent text to the "What's a Microcontroller?" guide. Like all Stamps in Class curriculum, this teaches new techniques and circuits with minimal overlap between the other texts. Topics introduced in this series are EEPROM data storage, mechanics of basic robotics, servo control, infrared obstacle detection and communication, and the use of photoresistors.

The depth and availability of a Teacher's Guide varies between the Stamps in Class curriculum. Because experts in their field independently author each set of experiments, and they are provided leeway in terms of format. Please call if you have any questions.

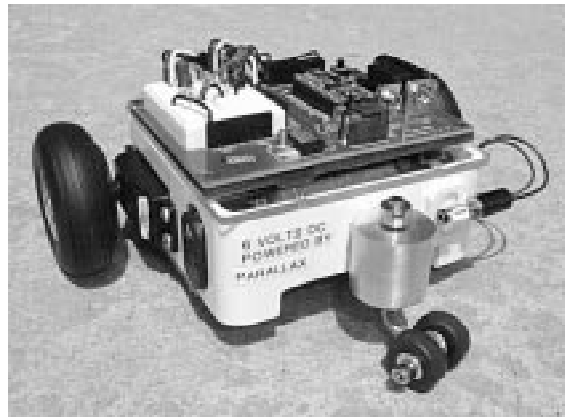
Copyright and Reproduction

Stamps in Class curriculum is copyright © Parallax 1999. 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* would profit from duplication of this text. Preferably, duplication would have no expense to the student. Any educational institution wishing to produce duplicates for their 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 to any other language with the permission of Parallax, Inc.

Special Contributors

Chuck Schoeffler Ph.D. at the University of Idaho authored portions of this curriculum in conjunction with Parallax, Inc. Chuck is a professor in UI's Industrial Technology Education department. He specializes with mechanical and electrical designs, and without question one of his hobbies is robotics and the BASIC Stamp. Chuck designed the original Board of Education Robot (Boe-Bot) shown below and many similar robot derivatives with unique functions. After several versions, Chuck's design was adopted as the basis of the Parallax Boe-Bot that is used in this Robotics curriculum. Chuck teaches robotics to all levels, ranging from high school students to his summer workshops with teachers. On free time he is known to be fixing automobiles for his family and riding his motorcycle in the hills around Moscow, Idaho. You can reach Chuck by e-mail anytime at chucks@uidaho.edu. Parallax is very appreciative of his involvement with the Stamps in Class program.

Special thanks also to the Parallax team who provided ideas and content for the program, particularly Russ Miller. Russ designed the Parallax Boe-Bot based on Dr. Schoeffler's input. The entire Parallax staff that designs, manufacturers, and accepts orders and packages the Stamps in Class products is a key part of the Stamps in Class program.





Experiment #1: Boe-Bot Construction

Robots are used in the auto industry, medical, manufacturing plants, and of course science fiction films. Building and programming a robot is a combination of mechanics, electronics, and problem solving. What you're about to experience with the Boe-Bot will be applicable to realistic applications using robotic control, the only

difference being the size and sophistication. The electronic control principles, source code, and circuits you will use are very similar (and sometimes identical) to industrial applications developed by electronic engineers.

The word "robot" first appeared in a Czechoslovakian satirical play *Rossum's Universal Robots* by Karel Capek in 1920. Robots in this play tended to be human-like. From this point it seemed that any good science fiction story would involve them revolting against human authority, which requires intelligence. This changed when General Motors installed the first robots in their manufacturing plant in 1961. For science fiction or manufacturing, intelligence is only installed in a robot through programmability.

This series of Robotics experiments will introduce you to basic robotic concepts and programming using the Board of Education Robot (hereafter the "Boe-Bot"). The experiments will begin with construction of the Boe-Bot. After that we'll be programming the Boe-Bot for basic maneuvers, and proceed to interface sensors that will allow the robot react to it's surroundings.

Experiment #1: Boe-Bot Construction



Parts Required

A Boe-Bot can take many different forms of construction, and it doesn't have to use the Parallax Boe-Bot chassis with a Board of Education and BASIC Stamp. One approach might be to build a rolling breadboard with servos attached using double-sided tape, and another might be to carve a robot base from a piece of plastic irrigation pipe. If you are using a BASIC Stamp II and the schematics

and source code included in this series of experiments your robot would work fine and be just as unique. It might look different than the Boe-Bot, but that's appropriate since the design has a lot to do with creativity.

If you're using the Boe-Bot from Parallax, you'll need the following parts for this experiment (all are in the Robotics Parts Kit):

- (1) Boe-Bot chassis
- (2) Two standard servos (Futaba S-148s or those supplied with the Robotics Parts kit)
- (2) plastic wheels
- (2) O-ring tires
- (1) polyethylene ball wheel
- (1) cotter pin to hold wheel on chassis
- (8) 4-40 3/8" machine screws
- (8) 4-40 1/4" machine screws
- (2) 4-40 3/8" flathead screws
- (10) 4-40 nylon insert locking nuts
- (4) 1/2" standoffs
- (1) grommet 13/32" inner diameter (fits 1/2" hole)
- (2) grommet 9/32" inner diameter (fits 3/8" hole)
- (1) AA battery pack holder with 9" of wire and 2.1 mm plug
- (2) 10K ohm resistors
- (2) 3-pin headers to plug the servos into the Board of Education
- (1) 3300 uF capacitor
- (1) Board of Education and BASIC Stamp II

Pictures of these parts are shown on the back cover of this text in color, along with quantities and Parallax stock code in case you need specific replacements.



Build It!

Boe-Bot construction requires a small Phillips screwdriver, 1/4" box-end wrench or small pliers, and small diagonal cutters. Constructing the Boe-Bot consists of five parts:

- Step #1:** Mechanical modification of the servos to provide continuous rotation and testing them for use as a robot chassis drivetrain;
- Step #2:** Mounting the servos on the robotics platform and attaching the wheels;
- Step #3:** Attaching the tail wheel and battery holder;
- Step #4:** Mounting the Board of Education and BASIC Stamp on the completed platform and connecting the servos to the BASIC Stamp; and
- Step #5:** Programming the Boe-Bot with some sample source code to verify that it works.

Step #1: Servo Modification and Testing

Servo Modification

The Boe-Bot uses two modified servos originally developed for use in radio controlled (R/C) model airplanes. The Parallax servos or the Futaba-S148 provide low cost, easy-to-modify gear motors that let the platform move continuously. Most hobby servos are usually designed to move about 90° to 180° total. The servos respond to a pulse width modulation signal (PWM) that you send to it using the BASIC Stamp. This is accomplished using the `pulsout` command.

Modifying the servos takes only a few minutes and requires a Phillips screwdriver, diagonal cutters, and a little careful disassembly. This modification can be reversed at a later date to return to the 90° to 120° angle if you save the small plastic drive plate (we're going to remove it), but you'll need to be careful not to damage the internal parts by turning the output shaft too far.

The servos have a control horn attached to the main output shaft that is secured in place with a Phillips screw. Turn the servo horn with your fingers until it stops so you have an idea of the physical limitation. We need to modify it so that the control horn will rotate continuously.

Remove the small Phillips screw holding the control horn on the servo. The shaft has splines on it, so you will have to apply upward pressure and then wiggle the control horn off. Look at the bottom of the servo and find the four Phillips head screws on the bottom of the case because you will need to remove those in addition to the one Phillips screw that was holding the control horn on the main output gear.

Experiment #1: Boe-Bot Construction

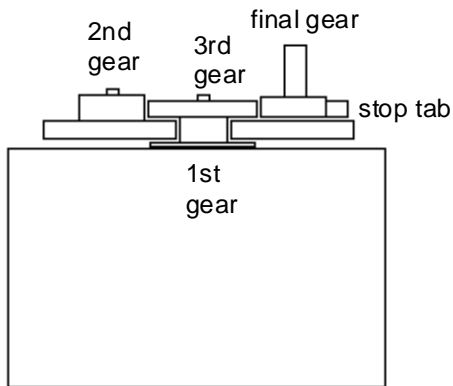


Figure 1.1: Servo Case Removal

Remove the top case by unscrewing the screw on the control horn and the four small screws holding the top case onto the bottom.

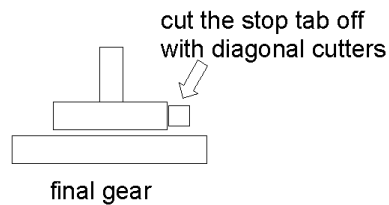


The bottom plate of the servo will come off at this point so look at the control circuitry. You won't need to do any soldering unless you break a wire off. Hold your finger on the output gear shaft and press down (the one that the control horn was on) and carefully pry and wiggle the top of the servo case up and remove it. Work slowly so the gears all stay in place on their shafts, and try to keep the bottom plate connected to the case. Once removed your servo should look like Figure 1.1.

The final gear is the one you are going to modify. It's also the one you held down with your finger. You'll need to remove the 3rd gear in order to access the final gear. Looking at the top of the final gear you'll see a plastic stop tab that we need to cut off to make the servo turn completely around when we issue commands from the BASIC Stamp. File, sand or cut the stop tab on the final gear until it is gone (don't sand into the gear teeth), making sure the area is flush once the tab is gone. The tab is shown in Figure 1.2.

Figure 1.2: Stop Tab Removal

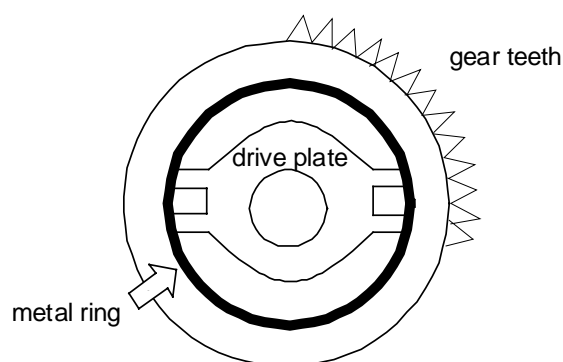
Once the final gear has been removed from the servo, cut or sand the stop tab off the part so that it may rotate freely. The area around the stop tab needs to be very smooth, and free of plastic debris.



Turn the final gear over and look at the bottom. You will see a metal ring pressed into the plastic, and you need to pry that out with a small screwdriver, your fingernail or a paper clip. Remove the potentiometer drive plate. You might want to save this plastic part if you ever want to convert the servo back to its normal mode of operation (returning the servo to its original state isn't covered in this text). Insert the metal ring back into the main gear. Figure 1.3 shows the bottom of the main gear. *Don't* reassemble your servo once you've finished this task.

Figure 1.3: Remove Drive Plate from Bottom of Final Gear

This is the bottom of the final gear. Pry out the metal ring and remove the potentiometer drive plate. Reinsert the metal ring without the drive plate.



Servo Testing and Calibration

At this point your servo is disassembled, and the final gear is removed to expose a metal post which is the end of a potentiometer. We will be rotating the position of the potentiometer to complete the modification for continuous rotation. This step is more understandable if you have a brief understanding of how the servo's internal electronics operate.

A servo is a classic example of a closed-loop feedback system. The servo has an internal potentiometer (the small metal shaft underneath the final gear) that is coupled to the output gear. Its resistance is proportional to the position of the servo's output shaft (0° to 120° before modification). This resistance is compared to the BASIC Stamp's command to generate an error signal when the desired position isn't the same as the current position. If you send a servo a command to place itself at 90° and the output shaft is actually at 45°, an error signal will cause the motor to move the output shaft (via the gears) until the error signal is 0 (when the output has reached 90°). If the output shaft had been at 180°, an error signal of opposite polarity would have been generated and the motor would have turned in the original direction to bring it 'back' to 90°. The current position is 'fed-back' to the servo's control system in a loop to maintain a zero error signal. The farther the potentiometer is from the desired position, the larger the error signal and, within the limits of the motor and electronics, the faster the motor turns to bring the error back to zero.

Experiment #1: Boe-Bot Construction

By removing the potentiometer drive plate we are opening the control loop. Now when we send a command to the servo an error signal will still be generated according to the difference between the command and potentiometer positions, but since the potentiometer isn't turned by the output shaft anymore, the motor will continue to run at the same speed as long as commands are sent. In summary, the modified servo will be functioning as a variable speed geared motor with a control signal that is very easy for the BASIC Stamp to generate.

The servo's internal potentiometer must be rotated to be "centered" according to a signal it expects to receive from the BASIC Stamp. This is done by connecting the servos to your BASIC Stamp and Board of Education as shown in Figure 1.4, which presents the servo case installed for convenience, though it should still be removed at this point. Servos have three wires leading from the base: power, ground and control. The control lead is used to send the positioning signal - the one connected to the BASIC Stamp I/O pin.

When preparing this circuit, note that the servos are connected to the Vin (unregulated 6.0 volts) that is jumpered from the AppMod Connector to the breadboard (see the cover photo). There's also a large 3300 uF capacitor placed between Vss and Vdd on the top of the breadboard. This is a polarized capacitor, so be sure it's placed correctly, with the long lead towards Vdd. See figure 1.10 for suggested parts placement.

Use the three-pin headers to plug the servos into the Board of Education. Before you plug the header into the board make sure the black plastic is centered. This will ensure that it makes contact with the breadboard nodes *and* the servo leads. Note that there is an optional resistor between the power and ground. This resistor prevents short servo movements during "timeouts" when the Boe-Bot is in a sitting stage and the servos are not being pulsed.

Figure 1.4: Servo Calibration Schematic

To test the servo connects the signal wires to the BASIC Stamp's P15 and P3 I/O pins on the Board of Education. Place the 3300 uF capacitor between Vdd and Vss above the breadboard, and jumper Vin from the AppMod connector to the breadboard for the servo's power supply.

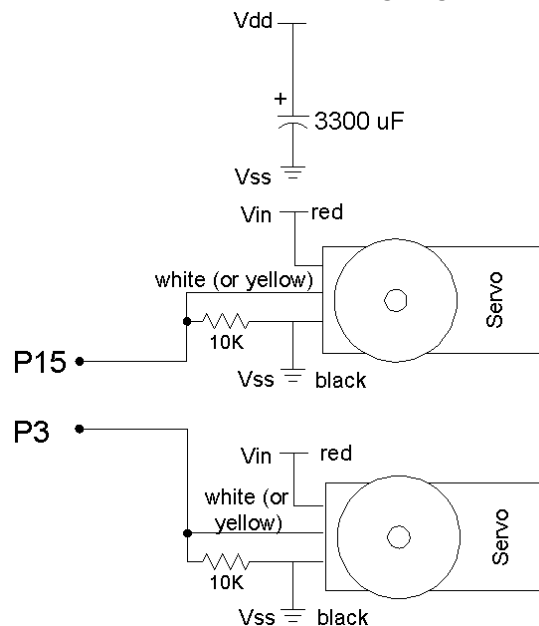
Servo wire color code:

Vin = red

Signal = white (sometimes yellow)

Vss = black

Use the 3-pin connector to plug the servos into the Board of Education. Slide the pins to center them in the plastic part like this:



Before we load some source code into the BASIC Stamp, take a look at the voltage regulator on your Board of Education. If the voltage regulator is labeled 7805 then you'll need to replace it (details are in Appendix C). If the voltage regulator is labeled 2940 then continue with this section. Parallax used the 7805 regulators before the idea for Boe-Bot was created, but switched to the 2940 in order to provide enough current to drive the servos. (The 2940 has a lower voltage drop and is able to supply plenty of current to drive the servo motors). If you have a 7805 regulator Parallax will provide a free upgrade kit.

To adjust the potentiometer load the following code into your BASIC Stamp:

```
'Program Listing 1.1 v1.2
'Program for calibrating servo to its center using BS-2

center:                'establishes a name for this calibration routine
  pulsout 15, 750      'sends a pulse of 1.5 milliseconds to the servo
  pause 20             'delay between pulses is 10 ms to 20 ms
goto center
```

When the code is running, turn the potentiometer shaft until the motor stops. Change the `pulsout 15,750` to read `pulsout 3, 750` and adjust potentiometer on the second servo.

So what have you done?

Servos are controlled by sending pulses of specific widths which the servo electronics interpret as positions using a system called Pulse Width Modulation (PWM). In order to understand this, you need to understand the terms "milliseconds" (ms) and "microseconds" (μ s). 1 ms is 1/1000th of a second; or put another way, there are 1000 ms in every second. 1 μ s is 1/1,000,000th of a second, or, there are 1 million μ s in each second. Servo manufacturers usually specify pulse-widths in μ s, so it's handy to be able to convert between μ s and ms.

The servo electronics are designed to interpret a positive-going pulse with period (width in ms) between about 1 and 2 ms as a position between 0 and 90 degrees for the output shaft. When you rotated the shaft back and forth the servo's potentiometer was comparing the pulse width value of 750 (this is 1.5 milliseconds because the `pulsout` command works in units of 2 μ s) to what it expected to receive for the potentiometer's center position after modification. You rotated the shaft until the two values were equal. At this point the motor stopped turning even though it was still receiving pulses.

The pulses are repeated about every 20 ms so that a new comparison can be made. The 20 ms repetition rate is not critical because all of the position information is carried by the pulse length itself, but the servo was designed to work well with a new pulse every 10 to 40 ms (25 to 100 times/second, or Hz).

Experiment #1: Boe-Bot Construction

Check the servo again before putting all the screws back in to see if the servo is indeed stopped when you sent it the test program. The servo case parts should go back together smoothly and when put together you should just barely be able to see where the case part joins together.

Step #2: Mounting Servos and Wheels on Boe-Bot Platform

When you're done modifying and testing the servos you can install them in the Boe-Bot base. This is done by placing the servo spline closest to the center of the chassis. This will let it turn corners in shorter spaces, and cause less friction on the tail wheel. For maximum stability, you could consider mounting the wheels with the servo spline closest to the front of the Boe-Bot. Each servo requires four screws and nuts for mounting.

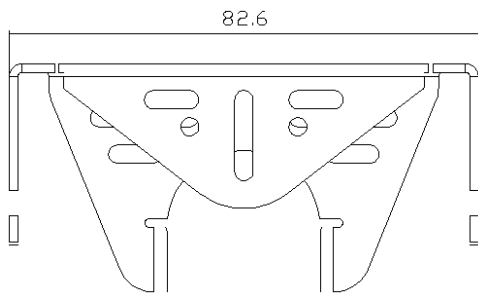


Figure 1.6: Boe-Bot Chassis Front View

The front of the Boe-Bot chassis has additional mounting holes for sensors, scrap printed circuit boards, or whatever you want to build into your Boe-Bot.

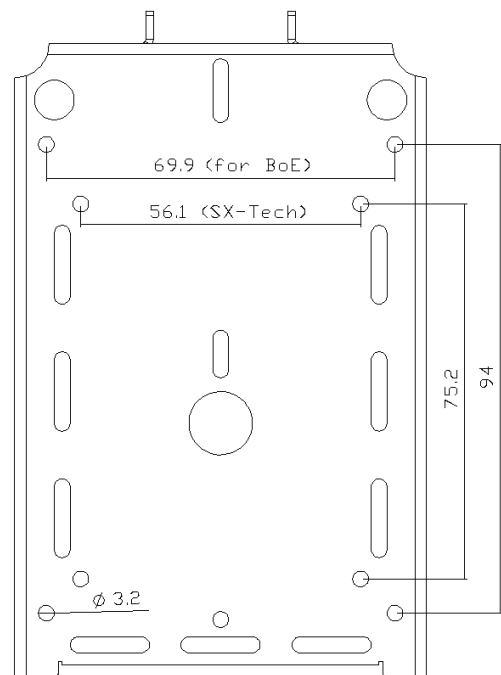


Figure 1.5: Boe-Bot Chassis Top View

The top of the Boe-Bot has mounting holes for the Board of Education, and places to attach other mechanical robot needs.

Figure 1.7: Boe-Bot Chassis Side View

Mount the servos on each side of the Boe-Bot chassis, with the spline closest towards the middle of the chassis.

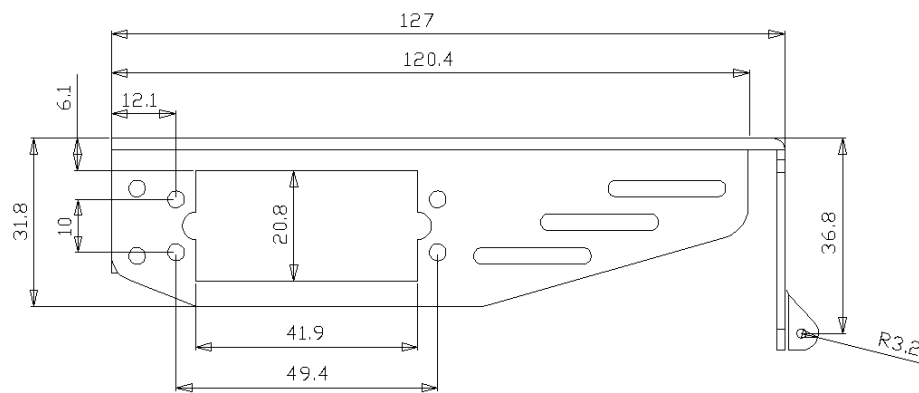
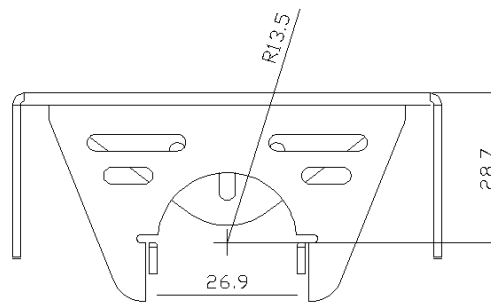


Figure 1.8: Boe-Bot Back View

The tail wheel is held between the rear gap with a cotter pin.



Experiment #1: Boe-Bot Construction

The pictures in Figures 1.5 through 1.8 are included with millimeter dimensions in case you would like to build your own Boe-Bot base, or you can also download dimensioned drawings in Adobe PDF, AutoCad DXF and AutoCad DWG files from <http://www.stampsinclass.com>.

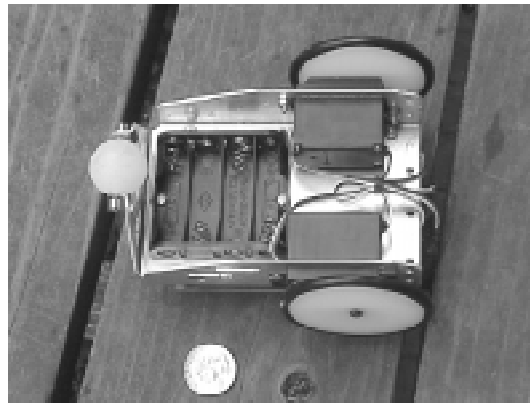
The Boe-Bot wheels have been machined with a spline that fits snugly on Parallax and Futaba style servos. Attach both wheels and replace the small black screw to hold them onto the servos. Once this is finished you can slide the rubber tires over the wheels.

Step #3: Install Tail Wheel and Battery Holder

Mount the tail wheel using the supplied cotter pin. The battery holder is attached using two flathead screws and nut. Install the three grommets: two fit in the rear holes and serve as a battery plug holder when it's not plugged into the Board of Education, and one mounts in the middle top part of the chassis where wires are threaded through the body. The 9" 2.1 mm coaxial DC power plug should be threaded through the middle hole of the Boe-Bot so you can connect it to the Board of Education. Bring all wires through the middle grommet hole. Figure 1.9 shows the completed undercarriage (this picture does not show the middle hole).

Figure 1.9: Boe-Bot Bottom Side View

Servos, battery pack, and tail wheel should be mounted at this point. Bring the wires towards the top of the Boe-Bot. This Boe-Bot has the wheels closest to the front, but the robot turns tighter if they are mounted in the reverse fashion.



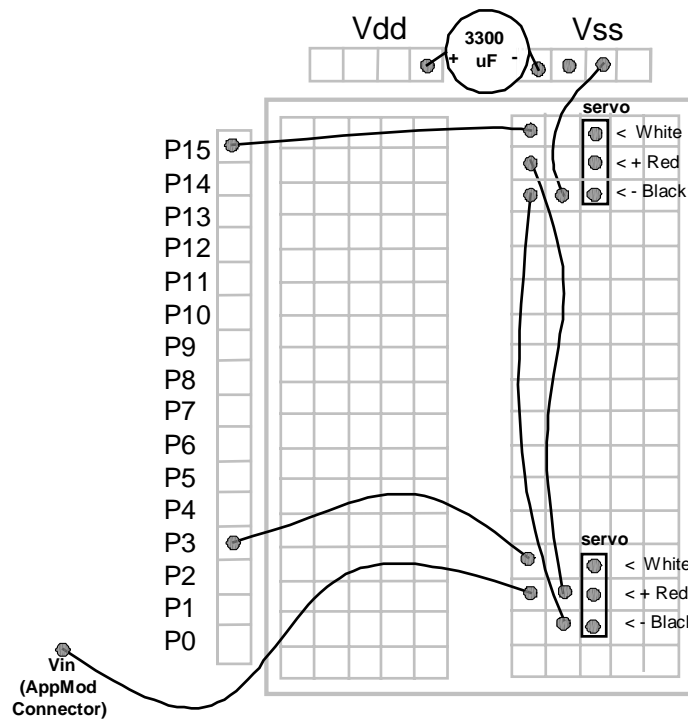
Step #4: Mounting the Board of Education and Connecting the Servos

Mount the Board of Education ("Boe" part of Boe-Bot) on the robot chassis base with eight 4/40 machine screws and standoffs. The breadboard should be closest to the front of the chassis, centered directly in between the two wheels. This will allow sensors to be mounted in a position where they can respond to the physical environment in front of the Boe-Bot.

Once this is done, re-connect the servos as shown in Figure 1.4. You'll need to use the two three-pin posts to plug the servos into the breadboard, and jumper them over to Vss and Vin. The finished Boe-Bot should roughly resemble the one shown in Figures 1.11 to 1.14.

Figure 1.10: Boe-Bot Suggested Wiring Placement

Once this circuit is built on your Boe-Bot it will be ready for simple rolling patterns.



Experiment #1: Boe-Bot Construction

Figure 1.11: Boe-Bot Completely Assembled

The basic Boe-Bot is assembled and servos are connected to the Board of Education (doesn't show 10K resistors and servo connections to I/O pins).

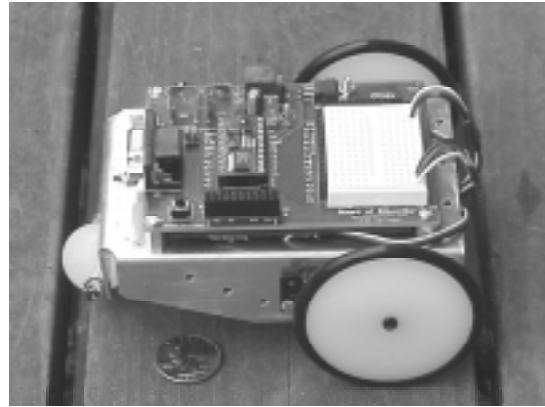


Figure 1.12: Boe-Bot Front Drawing

The front of your Boe-Bot has a few holes and slots where you will be able to mount sensors or other mechanical devices.

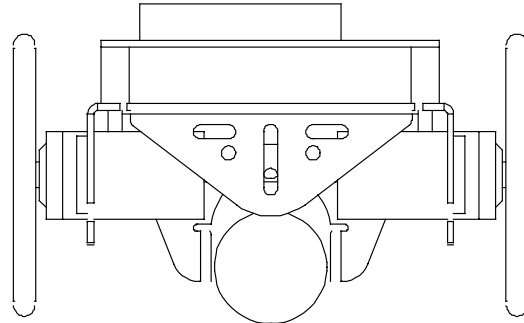


Figure 1.13: Boe-Bot Top Drawing

Note that the servos are placed with the wheels closest to the center of the robot. This makes turns very tight and reduces friction on the rear wheel.

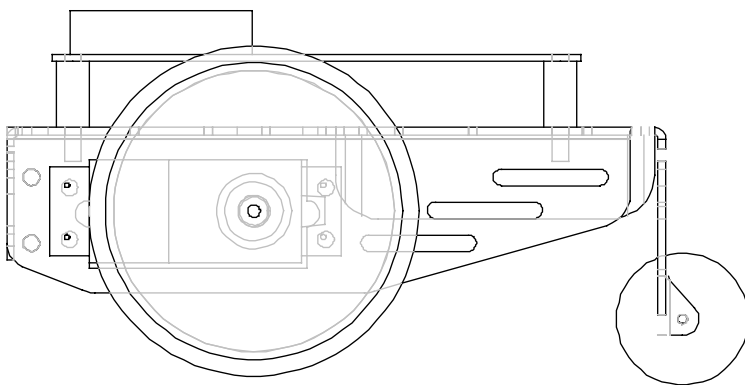
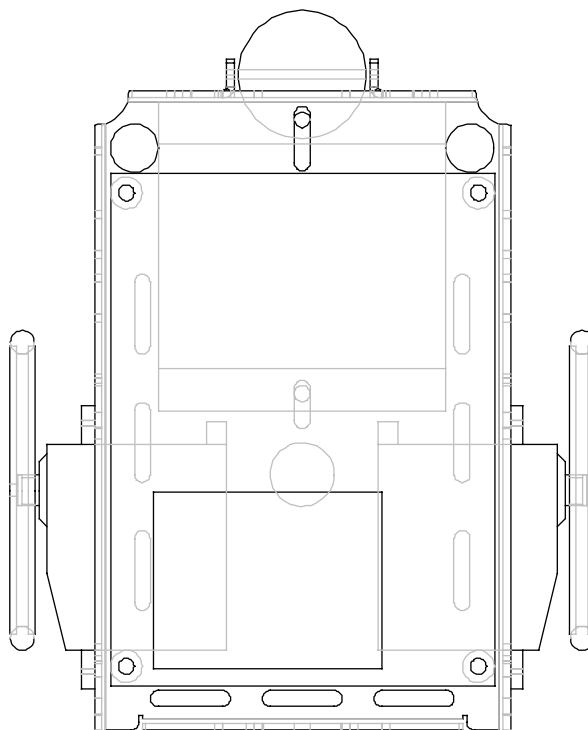


Figure 1.14: Boe-Bot Side Drawing

The breadboard on the Board of Education is mounted towards the front of the Boe-Bot. We'll be building our projects on this breadboard.

Experiment #1: Boe-Bot Construction

Step #5: Programming the Boe-Bot for Basic Movement Pattern

If construction went okay, the Boe-Bot is now ready to be programmed for a basic movement pattern. Load the Program Listing 1.2 code in your BASIC Stamp.

```
'Program Listing 1.2  v1.2
x      var      word

right_servo  con    3
left_servo   con   15

backwards:
for x=1 to 100
    pulsout left_servo,850
    pulsout right_servo,650
    pause 20
next

pause 250

forward:
for x=1 to 100
    pulsout left_servo,650
    pulsout right_servo,850
    pause 20
next
```

If everything worked properly the Boe-Bot should have moved forward and backward, finishing in the same location where it started. If it quivered and didn't move, then you'll need to verify that it's working from a full 6.0 volts and that the servos were properly modified. If there's no response then check the schematic to see that the correct BASIC Stamp I/O pins (3 and 15) are connected to the servos, and that the Vin and Vss wires are correct.

If the Boe-Bot moves backward first and then forward the servo wires have probably been swapped right for left.

In looking over the program, note the `pulsout` values of 650 and 850. Since we set the servo to stop moving with a command of 750, any shorter pulse than this will cause the servo to rotate clockwise, and longer values will rotate counter-clockwise. Because one servo is reversed when they are used as a drive train, one is pulsed the opposite direction to make the robot move forward or backwards. The value of the `for . . next` loop is proportional to the distance the Boe-Bot travels the specified direction.

Experiment #1: Boe-Bot Construction

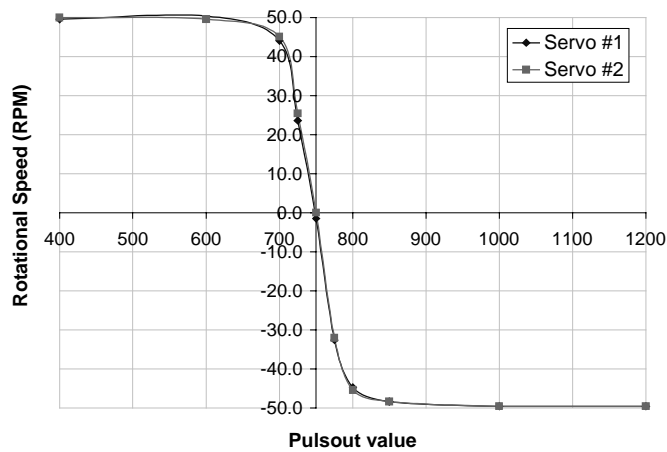


Challenge!

Program the Boe-Bot to move in several different patterns. Try the following:

- To move straight forward or backward we used different pulse lengths for each servo. What happens if both servos are sent the same pulse length?
- Identify `pulsout` values that make the Boe-Bot move forward very slowly, then backwards very slowly in a straight line.
- Make a graph of wheel speed as a function of pulse length for each servo.
Hints:
 - Use several pulse lengths between 1 and 2 ms (pulsout values between 500 and 1000).
 - Either count how many revolutions the wheel completes in a specified time (20 sec or a minute), or see how much time it takes to complete 10 revolutions

Your graph might look something like this:



Experiment #2: Basic Movements Using Subroutines and Memory



Experiment #2: Basic Movement Using Subroutines and Memory

Movement is one of the most distinctive features of robots. Moving around is necessary to find a way through a maze, to follow light, to locate a fire, to open and detonate a bomb, or to retrieve a parcel of hazardous waste. It can also be used to express emotion and intention. Robot movement

isn't limited to forward, backward, left and right, either. Depending on the style of a robot it could move up or down, and even sideways underwater.

Autonomous mobile robots decide to move based on the data they receive from their input devices - in the case of the Boe-Bot this includes infrared sensors, photoresistors and any other sensors that you decide to add.

This experiment is all about BASIC programming as it pertains to simple movement without sensor input. Structuring your program so the Boe-Bot moves as you intend requires an understanding of how to jump between subroutine labels, read movement patterns from an EEPROM, determine how far to travel, and how to get back to your starting point (physically, and within your source code). Experimenting and making all of these decisions will be made without the help of an input sensor.

In many experiments we used the `debug` statement to view the value of variables calculated by the BASIC Stamp. The `debug` statement sends the data to your PC screen, primarily to determine if we're getting the results we expected. With the Boe-Bot using `debug` becomes a bit more complex since your robot is usually ready to roll once a program is downloaded. There are two tricks we'll use to overcome this problem and verify that our code is working properly:

1. Stand the Boe-Bot on it's Front Chassis so that it doesn't move The front of the Boe-Bot is flat, allowing you to stand it on it's up vertically to prevent the wheels from touching the surface of your desk. The serial cable can remain connected to the Board of Education's DB-9 connector so that you can use the `debug` command.
2. LEDs and Piezospeaker LEDs can provide visual feedback to help evaluate the functionality of your source code. A Piezospeaker will be used to generate unique sounds that correspond to particular movements.

The LEDs and piezospeaker will add some interesting output devices to your Boe-Bot. Use these feedback mechanisms to give information about what your robot is doing. Keep in mind that a robot doesn't need to mimic human behavior. Deep space probes, automobile assembly line robots that paint and weld, and automated office mail delivery systems are good examples of real world robot applications. They accept commands and react to their surroundings, but neither look nor act anything like people.

Pulsout Command and Servo Control

Let's have a quick refresher course on servo control. Servos are closed loop devices, and they are constantly comparing their commanded position (from the BASIC Stamp's `pulsout` command) to their actual position (proportional to the resistance of a potentiometer mechanically linked to the shaft). If there is more than a small difference between the two, the servo's electronics will turn the motor to eliminate the error.

The servos we are using with the Boe-Bot were modified to rotate continuously. The potentiometer shaft was rotated until the gears stopped moving when sent 1500 us pulse with this program:

```
'Robotics Program Listing 2.1 v1.2
left_servo  con  15
right_servo con   3

start:
  pulsout left_servo,750      'begin of routine
  pulsout right_servo,750    'pulse width of 1500 us
  pause 20                   'pulse width of 1500 us
                             'pause for 20 ms
goto start                   'do it again
```

The `pulsout` value of 750 is equal to 1500 μ s since the command operates in units of two microseconds. If you send a value larger than 750 the servo will turn clockwise, and a value less than 750 will cause it to turn counter-clockwise. A value very close to 750, like 760, will cause the servo to turn very slowly (this is a way to add variable speed control to your Boe-Bot), but a value similar to 650 or 850 will command it to turn full speed. Experiment with different values while the Boe-Bot is standing on it's front. Try to send values that make the servos turn very slowly, not at all, or very quickly. Program Listing 2.2 demonstrates how a `for . . . next` loop could be used to vary speed.

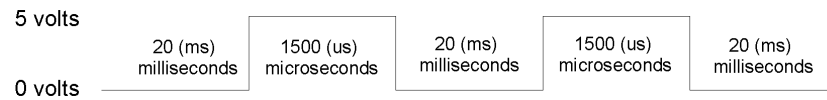
```
'Robotics Program Listing 2.2 v1.2
left_servo  con  15
right_servo con   3
x          var word

pause 2000
for x = 850 to 650
  pulsout left_servo,x      'begin of routine
  pulsout right_servo,1500-x 'pulse width of 1500 us
  pause 20                  'pulse width of 1500 us
                             'pause for 20 ms
next
```

Experiment #2: Basic Movements Using Subroutines and Memory

Pictorially, the `pulsout 750` command looks like Figure 2.1 The command sends pulses of 1500 μ s, with pauses of 20 ms in between. The pulse is a positive 5 V signal.

Figure 2.1: Pulsout Command



Parts Required

You'll need the following parts for these experiments:

- (1) fully constructed Boe-Bot with servos
- (1) 3300 μ F capacitor
- (1) piezo speaker
- (2) 10k Ohm resistors
- (2) 470 Ohm resistors
- (2) LEDs
- (misc.) jumper wires



Build It!

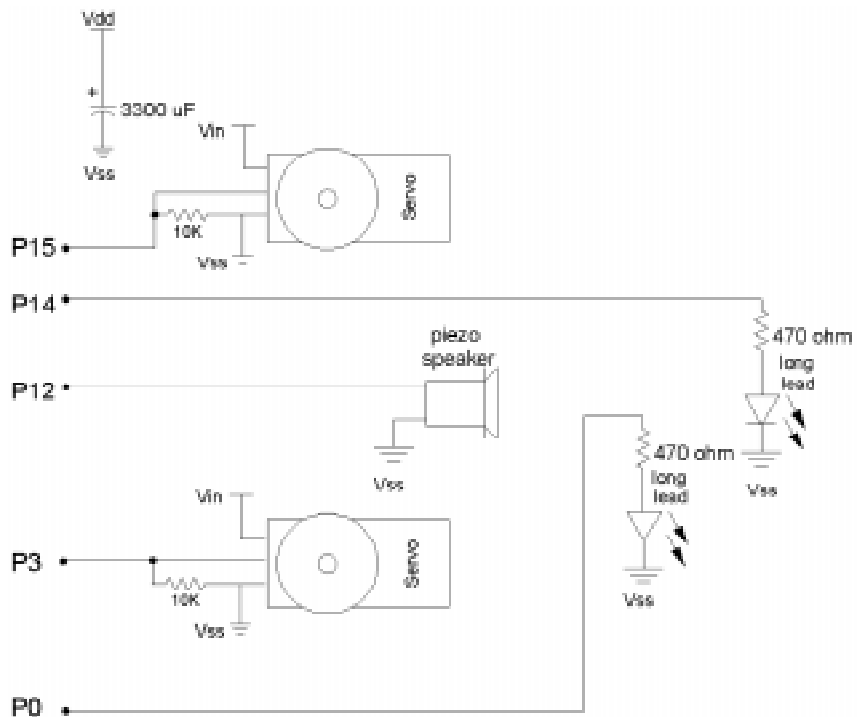
Build the project shown below in Figure 2.2. Place the 3300 uF capacitor between the Vdd and Vss connections on the top of the breadboard. Note that the servos are connected to the Vin (from the Boe's AppMod header) instead of Vdd adjacent to the breadboard.

Figure 2.2: Operational Schematic

This is a complete operational schematic using servos, peizo speaker and LEDs for feedback while the Boe-Bot rolls around.

Note that the servos are connected to the Vin power supply. This is an unregulated 6.0 V power supply from the battery pack.

There's also a 3300 uF capacitor between power (Vdd) and ground (Vss) on the top of the breadboard.



Experiment #2: Basic Movements Using Subroutines and Memory



Program It

This experiment will be developed in individual stages. The project is cumulative, starting with sound and light, and proceeding to use all pieces by the end.

Sound Feedback

The BASIC Stamp has a unique command for generating sound, appropriately named `freqout`. The `freqout` command will be used to drive the piezo speaker, and let us know which commands are being executed in our program. Like all PBASIC commands, it has a particular syntax that must be followed to make it work.

To hear the speaker download the following code to your BASIC Stamp:

```
freqout 12,750,2000      'generate 750 ms 2000 Hz tone on P12
```

The `freqout` command generates one or two sine waves using fast pulse-width modulation (PWM). The syntax for `freqout` is shown below:

```
freqout 12,750,2000 'generate 750 ms 2000 Hz tone on P12
^===== BASIC Stamp I/O pin 12
^===== duration is 750 ms
^=== frequency is 2000 Hz
```

The command also lets you add a second frequency that can be mixed with the first. This will let you create a sound that's far more "robotic" than a simple beep. For example try this:

```
'Robotics Program Listing 2.2a v1.2
Hz      var      word

for Hz = 1 to 4000 step 1000
  freqout 12,70,HZ,4000-HZ 'generate two 70 ms tones on P12
next
```

This routine begins by declaring `HZ` as a word variable, a number between 0 and 65,536. The loop executes a total of four times $((4000-1)/1000)$, generating two frequencies at once on P12. The first frequency is increasing from 1 to 4000 Hz while the second frequency is decreasing from 4000 to 1 Hz. The mixing of the two frequencies also creates a lower frequency tone making the combination sound very "electronic". Sounds like this could be added throughout your program.

LED Feedback

The circuit you built on the Boe-Bot has two LEDs. These LEDs are turned on and off using the BASIC Stamp's `high` and `low` commands. This command causes the BASIC Stamp to actively output a 5 V on one of its pins. This is a single, fast instruction. The command makes the corresponding bit high. For example, run this program on your BASIC Stamp:

```
'Robotics Program Listing 2.3 v1.2
led:      'led flashing routine
high 14   '5V to P14
high 0    '5V to P0
pause 1000 'pause 1 second
low 14    '0V to P14
low 0     '0V to P0
pause 1000 'pause 1 second
goto led  'do it again
```

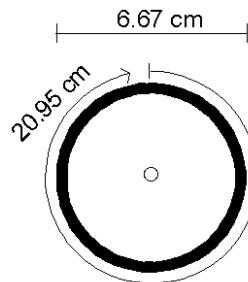
We'll actively use the LEDs when we program the Boe-Bot for turning movements.

Distance Calibration

When programming the Boe-Bot, you'll find that we're interested in making it move a specific distance or executing a particular turn (to the right or left). It is helpful to know how to figure out how far the robot will travel or turn when given a specific command. It's easy to get a rough idea. We know that circumference is equal to pi (π) multiplied by the wheel diameter:

$$\begin{aligned}\text{circumference} &= \pi \times \text{wheel diameter} \\ \text{circumference} &= 3.14159 \times 6.67 \text{ cm} \cong 21 \text{ cm}\end{aligned}$$

Figure 2.3: Wheel Diameter and Circumference



Experiment #2: Basic Movements Using Subroutines and Memory

With one complete turn of the wheels the Boe-Bot would travel about 21 cm. If we send pulses to the servo for the correct amount of time the Boe-bot can be made to travel a specific distance. For example with a pulsout command of 850, the servo will turn at about 50 revolutions per minute (RPM) or 0.83 revolutions/sec. So the speed of the robot will be about:

$$21 \text{ cm/revolution} \times .83 \text{ revolutions/sec} = 17.5 \text{ cm/s}$$

If we want it to go 100 cm, it would have to travel for:

$$100 \text{ cm} / 17.5 \text{ cm/s} = \text{about } 5.7 \text{ seconds}$$

By adjusting the number of times a FOR..NEXT loop is run we can set the distance the robot will travel. We want 5.7 seconds in this example, so since each servo pulse takes about 1.5 ms and there is a 20 pause, each loop will take about 23 ms (1.5+1.5+20), or .023 seconds per loop.

$$5.7 \text{ sec} / .023 \text{ sec/loop} = 247 \text{ loops}$$

Measure how far your Boe-bot goes with our estimate:

```
'Robotics Program Listing 2.4 v1.2
x      var      word
right_servo  con   3
left_servo   con  15

for x=1 to 247
  pulsout right_servo,850
  pulsout left_servo,650
  pause 20
next
```

How far did it go? Several factors can affect how far the robot moves, including differences between servos and battery voltage, but hopefully our estimate has given a good starting place to adjust the loop limit.

Modify the for. . . next loop value in the `forward` routine so your Boe-Bot travels 100 cm. This will give you an idea of the number of loops you need to execute to travel a particular distance. Suppose that you only wanted the Boe-bot to travel 10 cm – then the proper loop value would be about one tenth of the 100cm value. Remember this number because we'll be using it in the future.

Making Turns

Two parameters are modified to make the ideal turn at whatever angle you select. The first is that the `pulsout` values are the same for both sets of servos (they are either both 650 or they are both 850). The second modification is the number of loops. For example, enter the following program to make the robot move forward and then turn around:

```
'Robotics Program Listing 2.5 v1.2
right_servo con 3
left_servo con 15
x var word

'forward
for x=1 to 24
  pulsout left_servo,850
  pulsout right_servo,650
  pause 20
next

'turn_180:
for x=1 to 30 '←modify this value for turn angle
  pulsout left_servo,850 '←note the value of 850
  pulsout right_servo,850 '←note the value of 850
  pause 20
next
end
```

Substitute the `turn_180:` routine's `pulsout 850` for a value of 650, download the code and the Boe-Bot will turn the other way. The number of loops determines the angle of the turn. To make a 90 degree turn left or right simply decrease the number of loops.

Goto Statement

Normally, the program will execute one instruction and proceed to the next instruction. The GOTO command causes the BASIC Stamp to jump to a named place somewhere else in the program. It can be either forward or backward in the program.

```
goto forward 'goto the place on the program named with the
              'label forward:

^===== This label "forward" gives a name to a particular point
           in the program
```

Experiment #2: Basic Movements Using Subroutines and Memory

To test it out, add the following statement to the end of Program Listing 2.6

```
goto forward
```

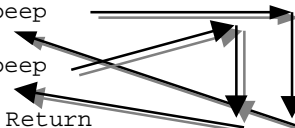
The purpose of the statement is probably *too* clear by now since the Boe-Bot won't stop moving forward, around, and back! To make the Boe-Bot stop, tilt it forward so it doesn't roll away and reprogram without the `goto` statement at the end of the code.

Gosub: A Close Relative of Goto

The `gosub` statement also causes the program execution to jump somewhere else, but there is a very important difference. The line after the `gosub` is remembered so that the program can automatically go back and continue where it left off. This lets us easily reuse sections of the program called subroutines. A subroutine always starts with a label and ends with a `return` statement:

```
Gosub blinkbeep
Pause 1000
Gosub blinkbeep
END
Return
```

```
blinkbeep:  `subroutine label
high 0
freqout 12,100,2000
Low 0
```



What's

Nested:

The term "nested" refers to the placement of subroutines or commands within a loop or branch command. For example:

```
x var byte
y var byte
for x = 1 to 5
  debug ? x
  for y = 1 to 5
    debug ? y
  next
next
```

The statements within the `y for... next` loop are nested within the `x` loop. The `y` loop counts from 1 to 5 for every single increment in `x`.

This little example shows the `blinkbeep` instructions being executed twice.

`Gosubs` may also be **nested**, so that each `return` takes the program back to the instruction after the most recent `gosub`.

To learn how to use the `gosub` statement download Program Listing 2.7 to your BASIC Stamp (the code is shown in two columns to conserve space). Trace out the jumps and returns of the program execution.

```
'Robotics Program Listing 2.6 v1.2
x      var      word
right_servo  con   3
left_servo   con  15

gosub forward
gosub left
gosub forward
gosub left
gosub forward
gosub left
gosub forward
gosub turn_around
gosub backward
end

'Subroutines

forward:
for x=1 to 60
pulsout left_servo,850
pulsout right_servo,650
pause 20
next
return

backward:
for x=1 to 60
pulsout left_servo,650
pulsout right_servo,850
pause 20
next
return

right:
for x=1 to 18
pulsout left_servo,850
pulsout right_servo,850
pause 20
next
return

left:
for x=1 to 18
pulsout left_servo,650
pulsout right_servo,650
pause 20
next
return

turn_around:
for x=1 to 30
pulsout left_servo,850
pulsout right_servo,850
pause 20
next
return
```

This program moves the Boe-Bot in a square. You will probably have to adjust the number of loops executed in each subroutine to make 90 degree turns. Try all of the subroutines. Could `turn_around` be written in a different way?

By making a long list of `gosubs` we could build up a complex movement pattern, but it is very tedious to make changes. The next section will use letters stored in the BASIC Stamp EEPROM memory to represent the movements we wish to make.

Experiment #2: Basic Movements Using Subroutines and Memory

Using the Data Command and EEPROM to Store Movements

The BASIC Stamp has a 2K EEPROM that is used for program storage (which builds downward from address 2047) and data storage (stores in the opposite direction – from address 0 to 2047). If the data collides with your program the PBASIC program won't execute properly. Figure 2.4 graphically shows how the EEPROM is filled with your program and data.

EEPROM is different from RAM variable storage in several aspects:

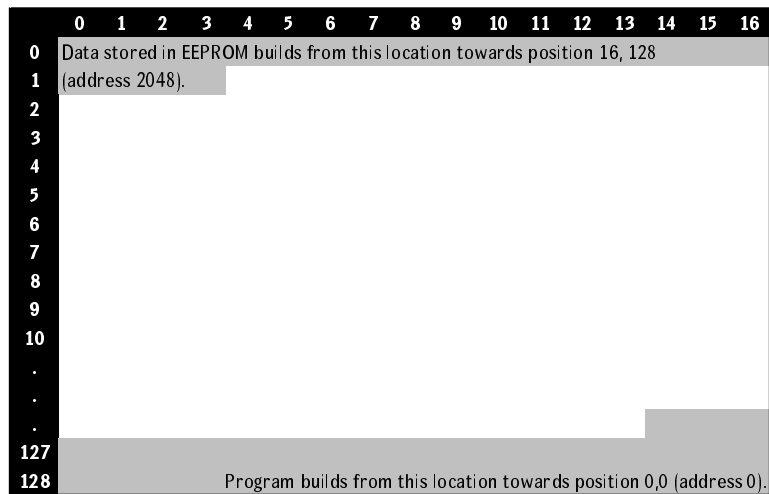
- EEPROM takes more time to store a value, sometimes up to several milliseconds.
- EEPROM can accept a finite number of write cycles, around 10 million writes to be exact (RAM has unlimited read/write capabilities).
- Primary function of the EEPROM is to store programs; data is stored in leftover space.

Three commands are used to access the EEPROM: `data`, `read`, and `write`. We'll be using the `data` and `read` statements.

Physically, the EEPROM is the small black chip on the BASIC Stamp II module labeled "24LC16B".

Figure 2.4: EEPROM Memory Map

By accessing the run / memory map icon on the BASIC Stamp editor you will be able to see how much space is used by your PBASIC program. Data stored in EEPROM is not visible on the memory map.



To demonstrate the EEPROM, we'll combine everything we learned from Experiment #2 into one big program!

All Together Now

Program Listing 2.7 brings all of these concepts together: light, sound, movement, and speed. In order to make effective use of the speed, you will need to identify the exact center position of your servo. Chances are although we started with 750 (1500 us) the servo may have wandered to a slightly different value. Program Listing 2.1 may be run to identify these values. At higher speeds (*speed* constant around 40) this is not as visible, but at slower speeds the Boe-Bot will slowly move sideways if the center position is not exactly 750. Most importantly: experiment! Change values, movement patterns, and sounds. This program is also available from the downloads section in <http://www.stampsinclass.com>. Code is not included for the LEDs, but you can add those if you desire.

```
'Robotics Program Listing 2.7 v1.2
'Boe-Bot Program for Roaming and Sound
'Define Variables and Constants
'-----
x          var    word    'loop counter for pulsout
position  var    word    'EEPROM address counter
direction var    word    'value stored in EEPROM
Hz        var    word    'frequency variable
right_servo con  3      'right servo on P3
left_servo con  15     'left servo on P15
speaker   con  12     'speaker on P12
right_led con  0      'right LED on P0
left_led  con  14     'left LED on P14
speed     con  50     'added or subtracted value
'-----

'Programmed Movement Patterns
'-----
data "FRFRFRBBTFE"      'store movements
'-----

'Main Program
'-----
position=0              'start at EEPROM cell 0
move:                  'main loop
read position,direction 'read direction command
position=position+1    'increment to next cell
  if direction="E" then quit 'Decide which action to take
  if direction="F" then forward 'by matching command letter
  if direction="R" then right
  if direction="L" then left
  if direction="B" then backward
```

Experiment #2: Basic Movements Using Subroutines and Memory

```
    if direction="T" then turn_around
goto move                                'repeat until E is seen
'-----

'Sound Routines
'-----
forward_sound:
for Hz = 1 to 4000 step 1000
freqout speaker,70,Hz,4000-Hz
next
return

back_sound:
for Hz = 4000 to 6000 step 1000
freqout speaker,70,Hz,Hz-400
next
return

right_sound:
freqout speaker,200,2500
return

left_sound:
freqout speaker,200,4500
return
'-----

'Movements
'-----
forward:
gosub forward_sound
for x=1 to 60
pulsout left_servo,750+speed
pulsout right_servo,750-speed
pause 20
next
goto move

backward:
gosub back_sound
for x=1 to 60
pulsout left_servo,750-speed
pulsout right_servo,750+speed
pause 20
next
goto move
```

```
right:
high right_led
gosub right_sound
for x=1 to 18
pulsout left_servo,750+speed
pulsout right_servo,750+speed
pause 20
next
low right_led
goto move
```

```
left:
high left_led
gosub left_sound
for x=1 to 18
pulsout left_servo,750-speed
pulsout right_servo,750-speed
pause 20
next
low left_led
goto move
```

```
turn_around:
for x=1 to 40
pulsout left_servo,850
pulsout right_servo,850
pause 20
next
goto move
```

```
quit:
end
```

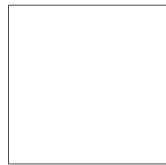
Experiment #2: Basic Movements Using Subroutines and Memory



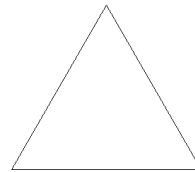
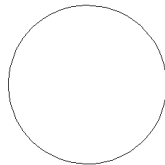
Challenge!

1. The Boe-Bot is being used to transport reactive material, in particular solid sodium and water. If the two react they explode, leaving your Boe-Bot as a pile of components. In order to carefully transport the chemicals you will need to start movements with increasing velocity. Create a program that drives the Boe-Bot in a 1 meter square with smooth turning transitions.
2. Create a simple movement pattern with several directions, for example F,B,R,R,F,F,L, and lastly F. These patterns would be stored and read from the EEPROM. When you are done executing the pattern, trace the same pattern in a backwards fashion.
3. Create source code for the following movement patterns:

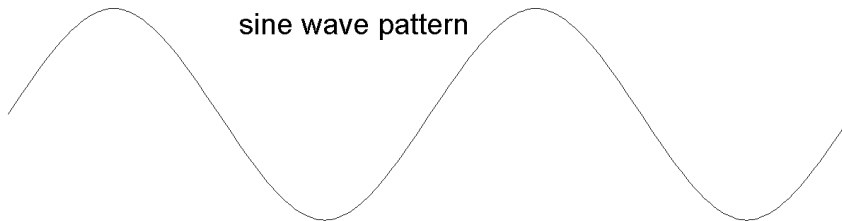
40 cm/side



20 cm radius



sine wave pattern



Experiment #3: Following Light



Experiment #3: Following Light

Light has many applications in robotics and industrial control. Some examples include sensing the edge of a roll of fabric in the textile industry, determining when to activate streetlights at different times of the year, when to take a picture, or when to deliver water to a crop of plants.

To sense the presence and intensity of light we'll build a couple of photoresistor circuits on our Boe-Bot. 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 proportionate 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 Boe-Bot kit are EG&G Vactec (#VT935G). If you need additional photoresistors they are available from Parallax and electronic component suppliers (parts listing is included in Appendix A). The specifications of these photoresistors are shown in Figure 3.1:

Figure 3.1: EG&G Vactec
Photoresistor Specifications

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

Illuminance is a scientific name for the measurement of incident light. The unit of measurement of illuminance 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 illuminance is higher or lower in certain directions. Based on this data the Boe-Bot will turn towards the light. For more information about light measurement with a microcontroller, take a look at Earth Measurements Experiment #4, Light on Earth and Data Logging.

The topics we'll explore in this experiment also relate to program structure. Some of the variables that will need to be customized for your Boe-Bot are how far to travel before checking the photoresistors and how much to turn when a photoresistor detects light.



Parts Required

You'll need the following parts for these experiments:

- (1) fully constructed Boe-Bot with servos
- (1) 3300 uF capacitor
- (1) piezo speaker
- (2) 0.01 uF capacitor
- (2) 10K Ohm resistors
- (2) 220 Ohm resistors
- (2) 470 Ohm resistors
- (2) red LEDs
- (2) photoresistors
- (misc.) jumper wires

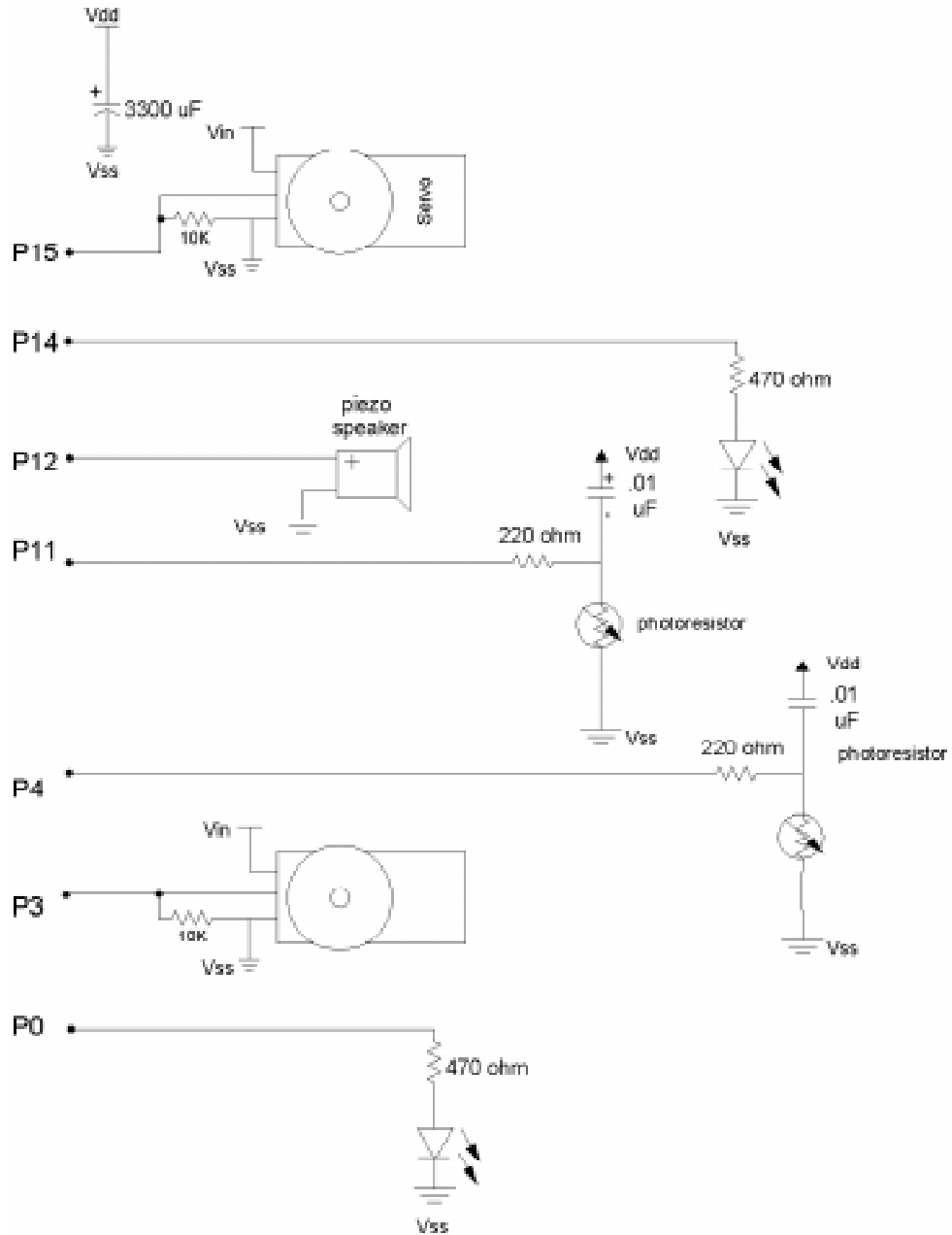


Build It!

While building this circuit it would be practical to move all of the servo connections and speakers towards the part of the breadboard closest to the BASIC Stamp. The schematic for this project is shown in Figure 3.2.

Experiment #3: Following Light

Figure 3.2: Light Sensing Boe-Bot Schematic
 The capacitor sizes for the photoresistors are not critical. These may also be 0.1 or 1.0 uF.



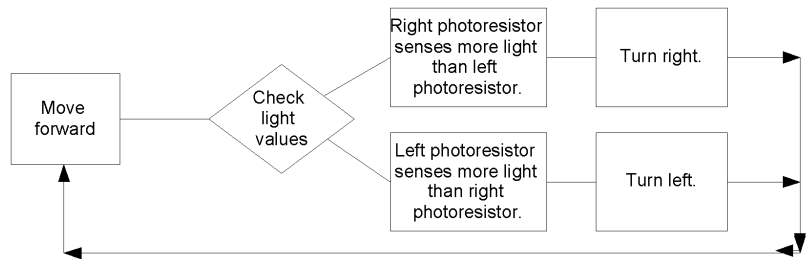


Program It

The Boe-Bot must be programmed to follow light, making turns towards more light. Consider how we want the Boe-Bot to execute the series of commands. One approach is to build a flowchart of the steps the Boe-Bot should execute to find the light and move in that direction. An example is shown in Figure 3.3.

Figure 3.3: Basic Operational Flow Chart

The main decision that needs to be made is in response to the question "where's the light?". Other decisions relating to "how often" to check the photoresistors and "how far" to travel are also important.



There are some decisions not shown in this flowchart. For example, how far to travel before checking the photoresistors, and how much to turn in order to track the light without missing it entirely. And how do we know if the two photoresistors are measuring light properly? We'll verify the light values by generating sound with a piezospeaker.

Using the Photoresistor

The photoresistors have a non-linear response to light. To measure their output we'll use a resistor/capacitor circuit. The `RCTime` command measures the charge (or discharge) time of a resistor/capacitor circuit. When `RCTime` executes, it starts a counter that increments ever $2 \mu\text{s}$. It stops this counter as soon as the pin is no longer in the starting state. The starting state of our pin will be 1 (5 volts). When the circuit discharges to 1.4 V (and the pin is considered "low") the command will stop and store the time value in a variable. The command works as shown below, and an example is shown in Program Listing 3.1:

Experiment #3: Following Light

```
RCTime 4, 0, rightLDR
    ^===== I/O pin with photoresistor
      ^===== starting state (0 or 1)
        ^===== variable location to store the charge time

'Program Listing 3.1 v1.2
scale          con    100    'adjust value to make audible frequency
rightLDR       var    word
leftLDR        var    word
left_LDRpin    con    11
right_LDRpin   con    4
speaker        con    12

left_light:
high left_LDRpin
pause 100
rctime left_LDRpin,1,leftLDR
debug cls,home,? leftLDR
freqout speaker,100,leftLDR*scale+100
pause 10

right_light:
high right_LDRpin          'discharge the capacitor
pause 100
rctime right_LDRpin,1,rightLDR    'measure the time to charge
debug ? rightLDR              'display the charge time
freqout speaker,100,rightLDR*scale+100 'play a tone, more light = lower
pitch
pause 200

goto left_light
```

With more light present on the photoresistors the charge time of the resistor / capacitor circuit is shorter, and the lower frequency of the tone. In order to generate an audible range of tones adjust the `scale` constant while exposing the photoresistors to varying light levels.

You may have noticed that when you point the Boe-Bot's photoresistors towards an even light (or dark) area of the room the `rightLDR` and `leftLDR` values are not equal. This is a result of variations between capacitors and tolerance of resistors. You can "calibrate" the photoresistor returning the lower `RCTime` value by adding the difference between the two photoresistors. For example, if the `rightLDR` returns `RCTime` values that are lower than the `leftLDR` by a value of 5, you could make the following modification to your code:

```
.  
.  
right_light:  
high right_LDRpin  
pause 100  
rctime right_LDRpin,1,rightLDR  
rightLDR=rightLDR + 5  
debug home, dec ? rightLDR  
freqout speaker,100,rightLDR*scale+100  
pause 10  
.  
.
```

This calibration will only be effective over a narrow range of `RCTime` values since the photoresistors are non-linear.

Experiment #3: Following Light

Source Code Example

Program Listing 3.2 is an example application of light following. Readings are taken on each photoresistor and the Boe-bot turns toward the brighter reading.

```
'Boe-Bot Program for Light Following with Sound Feedback
'Program Listing 3.2  v1.2

'Define Variables and Constants
'-----
left_servo      con      15
right_servo     con      3
turnvalue       con      6
scale           con     100
speed           con     100
speaker         con      12
right_LED       con      0
left_LED        con      14
rightLDR        var      word
leftLDR         var      word
right_LDRpin    con      4
left_LDRpin     con     11
x               var      word
'-----

'Main Program
'-----
forward:
for x=1 to 10
pulsout left_servo,750+speed
pulsout right_servo,750-speed
pause 20
next

left_light:
high left_LDRpin
pause 10
rctime left_LDRpin,1,leftLDR
debug "L:",dec leftLDR
freqout speaker,100,leftLDR*scale+100

right_light:
high right_LDRpin
pause 10
```

```
rctime right_LDRpin,1,rightLDR
debug "  R:",dec rightLDR,cr
freqout speaker,100,rightLDR*scale+100

if leftLDR > rightLDR then right `brighter on the right
if leftLDR < rightLDR then left  `brighter on the left

right:
high right_LED
for x=1 to turnvalue
  pulsout left_servo,750+speed
  pulsout right_servo,750+speed
  pause 20
next
low right_LED
goto forward

left:
high left_LED
for x=1 to turnvalue
  pulsout left_servo,750-speed
  pulsout right_servo,750-speed
  pause 20
next
low left_LED
goto forward
'-----
```

What would happen if the light readings were the same on both sides?

Experiment with the source code by changing different software and hardware parameters. You'll find that the Boe-Bot is not aware of objects since it has no feedback for sensing objects, and can easily run into walls and chairs. It may also be necessary to shield the edges of the photoresistors with a straw or heat-shrink tubing to reduce light reflection from the side.

In Experiment #4 we'll introduce infrared communication for proximity sensing.

Experiment #3: Following Light



Challenge!

1. Program the Boe-Bot to hide from light. The Boe-Bot should move towards the darkest corner of the room, and when a specific level of luminance is reached it should stop in place and generate a siren signal.
2. Restructure Program Listing 3.2 to use the `gosub` statement to execute a series of tasks.
3. Can the Boe-Bot follow a light source with just one photoresistor? Instead of comparing readings from two sensors, write a program that takes two readings (in slightly different directions) with one sensor and then goes toward the brighter reading.



Experiment #4: Infrared Object Detection

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 use an infrared LED and receiver to detect objects that exist to the front and side of your traveling Boe-Bot.

4

The major limitation to IR technology is the need for a clear line of sight between the transmitter and receiver. If an object were to interrupt the beam of light, the connection would be disrupted. The Boe-Bot will be bouncing IR off of objects and receiving them. The success of the application depends on the height of the object you are detecting, the location within the IR LED's visible range, and the reflectivity of the surface.

Obstacle Alley

If you drive an automobile, you know the practical application of the Pauli exclusion principle: Two objects can't occupy the same space at the same time. What's true for automobiles is even truer for robots. An autonomous robot has to keep itself from colliding with obstacles. Obstacles might take the form of a wall or post, or they may be mobile like a dog, a person, or another robot.

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 sensor work at 980 nm (nanometers) which is considered near infra-red. Night Vision goggles and IR Temperature sensing use far infra-red wavelengths of 2000- 10,000 nm depending on the application.

<u>Color</u>	<u>Approximate Wavelength</u>
Violet	400 nm
Blue	470
Green	565
Yellow	590
Orange	630
Red	780 nm
Near infra-red	800-1000 nm

Since the robot can't know the positions of moving objects in advance, it must have some way of detecting obstacles in real time. Humans, of course, use vision. While a robot that can see would be very desirable, it is also quite expensive and difficult to make a vision system appropriate for robotics.

Luckily, 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.

Infrared Transmission

As the name implies, IR remote controls transmit instructions over a beam of light. To avoid interference from sunlight and other household sources of infrared, primarily incandescent lights, the detector that we're using has two important features that help reject unwanted signals.

Experiment #4: Infrared Object Detection

First, it has an optical filter which allows very little light except the 980 nm infra-red that we want to detect onto its internal photodiode sensor. Then, there is an electronic filter that will only allow signals around 38 kHz to pass through. Since there are very few sources of 38 kHz infra-red naturally occurring we're not likely to get interference. The brightness of incandescent and fluorescent lights does vary at 120 Hz, twice the power mains frequency of 60 Hz (or 100 Hz if your mains frequency is 50 Hz), but this is so much lower than 38kHz that the detector ignores it completely.

A 555 timer circuit is used to produce an output square wave that runs at about 38kHz. This circuit should be familiar from "What's a Microcontroller?" experiment 5. The output pin is connected to an IR LED, and the reset pin is used to turn the 555 on and off. The duty cycle, or percentage of LED on-time, is not critical to reliable operation of the IR detector module, but if the duty cycle gets too far above or below 50%, the detection range will be reduced.

We will need to tune the 555 circuit with a potentiometer to account for normal component variations to produce the desired 38 kHz frequency. The BASIC Stamp count function will be used to measure the frequency during the tuning process.

Infrared Reception

The IR LED and receiver may be used to send and receive serial data, but initially all we'll be doing is detecting the presence of an IR signal being reflected from an object. We'll be receiving the IR signal when the BASIC Stamp I/O pins connected to the receiver are "low".

Many types of IR LED transmitters could be used for this experiment. The transmitter we're using requires a frequency carrier of 38 kHz. This distance is shortened dramatically when we're trying to detect the presence of an object, and expecting the wave to be bounced back from the object's surface. Variables such as texture, surface color, and reflectivity affect reliability.



Build It!

Building the IR circuit requires two steps: oscillator tuning and implementing a receiver. Placement of the components will be critical due to the large number of parts that will be present on the breadboard. Share Vss and Vdd connections where possible, and leave space on the front of the breadboard for the transmitter and receiver. Refer to Figure 4.3 for parts placement.

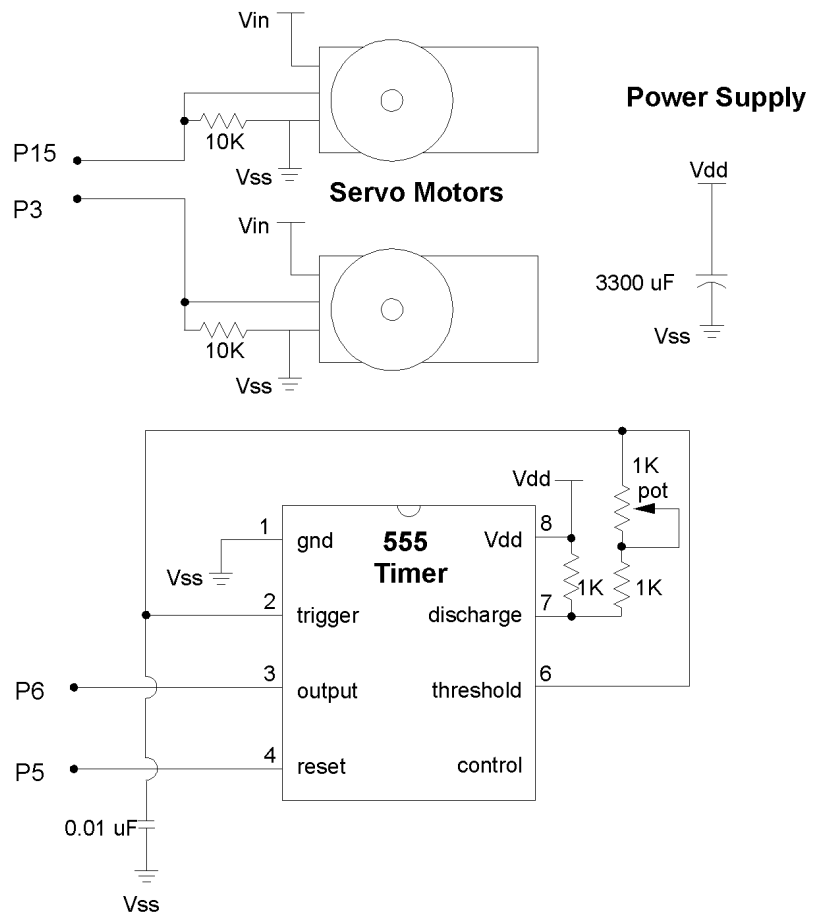
Oscillator Tuning

The first step requires tuning the oscillator circuit to 38 kHz. The projects are cumulative. You'll need the following components for this circuit:

- (1) operational Boe-Bot with 3300 uF capacitor between Vss and Vdd
- (1) 555 timer
- (1) 1K potentiometer
- (1) 0.01 uF capacitor
- (2) 1K resistor
- (misc.) jumper wires

Schematic is shown in Figure 4.1.

Figure 4.1: Oscillator Tuning Schematic



Experiment #4: Infrared Object Detection



Program It

Run Program Listing 4.1. In order to tune the circuit we'll adjust the potentiometer with a small screwdriver.

```
'Program Listing 4.1 v1.2
frequency var word
high 5          'turn on oscillator
start:
  count 6,100,frequency
  debug dec5 frequency*10,cr
goto start
```

The 555's oscillation frequency will be shown on the debug screen. Turn the potentiometer with a small screwdriver until the frequency is equal to approximately 38000. Next we'll add the infrared LEDs and receivers.

Infrared Transmit and Receive

Complete the project by adding infrared receivers shown in Figures 4.2, 4.3, and 4.4. Infrared receivers and LEDs need to be placed near the front of the breadboard. You'll need the additional following parts to add the receivers:

- (2) infrared receivers
- (2) infrared LEDs
- (2) 470 ohm resistors
- (2) 0.1 uF capacitors
- (misc.) jumper wires

Figure 4.2: Infrared Receiver and LED Placement on Breadboard

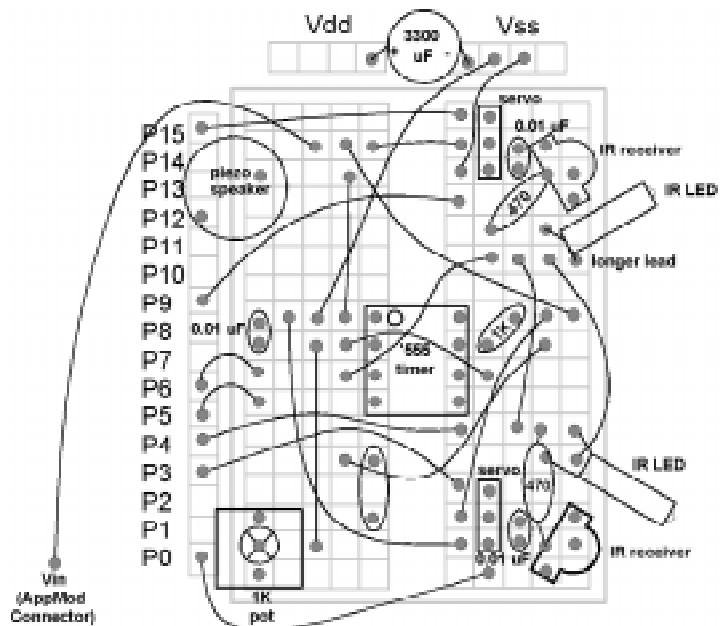
The infrared LEDs will focus forwards at a 45° angle. The angle between the two LEDs is 90°. The receivers should be positioned on each side of the breadboard facing the same direction. These positions will need to be tuned for optimal performance.



Figure 4.4 shows a diagram of how the parts could be placed on the breadboard to achieve this result.

Figure 4.3: Infrared Object Detection Parts Placement Diagram

There are many ways to wire this circuit, and this is only one option. The key features of this placement are that servo connection Vss (grounds) are shared with the IR receivers, and Vin connections are jumpered between servos.



Experiment #4: Infrared Object Detection

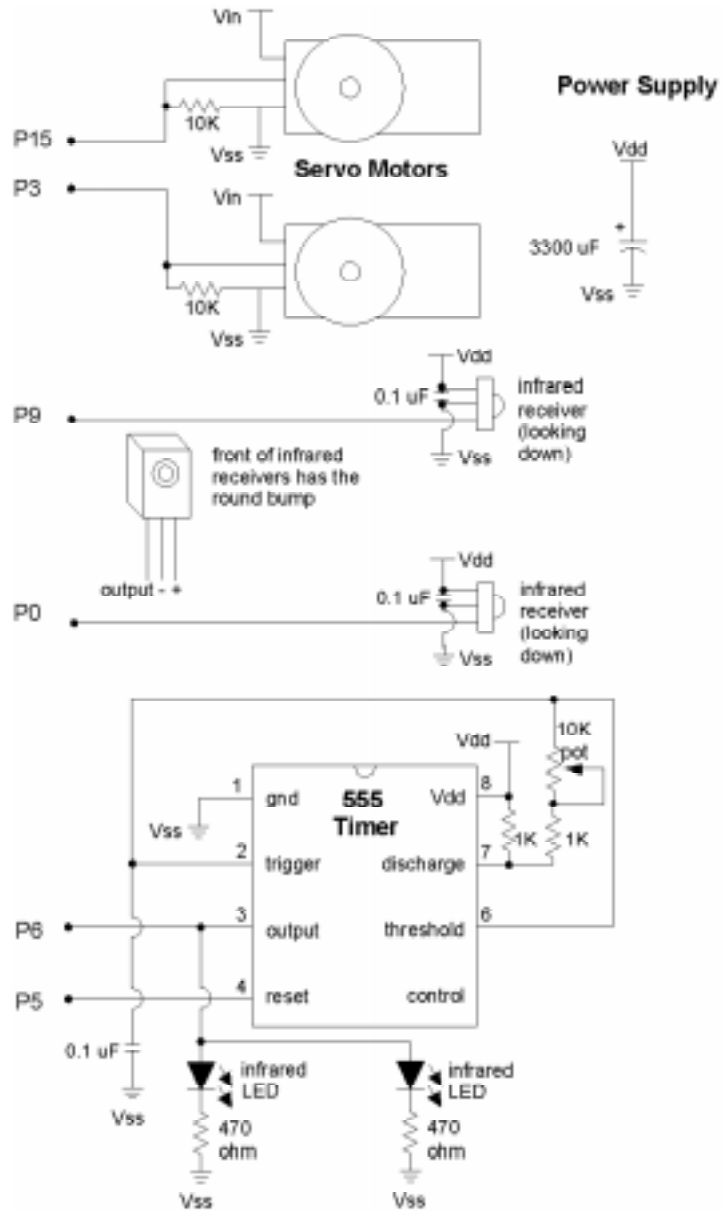


Figure 4.4: Infrared Receiver and Transmit Schematic

Enter and download Program Listing 4.2. The infrared receivers are connected to P0 and P9. When the I/O pins are "low" the infrared signal has been received from bouncing off of an object. This source code may be debugged while the Boe-Bot is connected to your PC. The infrared transmitters and receivers will need to be tuned through experimentation.

```
'Program Listing 4.2  v1.2
'Boe-Bot Program for Avoidance Using IR

'Define Variables and Constants
'-----
Hz          var    word    'frequency variable
x           var    word    'loop counter
right_IR    var    in0     'right IR on P0
left_IR     var    in9     'left IR on P9
right_servo con    3       'right servo on P3
left_servo  con    15      'left servo on P15
IR_out      con    5       'Enable IR transmitter
delay       con    10      'distance constant
speaker     con    12      'speaker on P12
speed       con    50      'speed constant
'-----

'Define Variables and Constants
'-----
high IR_out                'set IR high

alarm:                     'reset routine
forward_sound:             'occurs if BASIC Stamp
for Hz = 1 to 4000 step 1000 'is reset
  freqout speaker,70,Hz,4000-Hz 'sound on P12
next                        'repeat Hz times

sense:                     'check IR routine
if (left_IR=0) and (right_IR=0) then back 'if both IR detect then back
if left_IR=0 then right     'left IR detect then right
if right_IR=0 then left    'right IR detect then left

forward:                   'otherwise move forward
debug "forward",cr         'debug direction to screen
for x = 1 to delay*2       'distance
  pulsout left_servo,750+speed 'pulse left servo
  pulsout right_servo,750-speed 'pulse right servo
  pause 20                 'pause 20 ms
next                       'repeat x times
goto sense                 'jump to sense routine
```

Experiment #4: Infrared Object Detection

```
back:                                     'move backward
debug "backward",cr                       'debug direction to screen
for x=1 to delay*3                        'distance
  pulsout left_servo,750-speed            'pulse left servo
  pulsout right_servo,750+speed           'pulse right servo
  pause 20                                'pause 20 ms
next                                       'repeat x times
goto right                                'turn right when done

left:                                      'move left
debug "left",cr                           'debug direction to screen
for x = 1 to delay*1                      'distance
  pulsout left_servo,750-speed            'pulse left servo
  pulsout right_servo,750-speed           'pulse right servo
  pause 20                                'pause 20 ms
next                                       'repeat x times
goto sense                                'jump to sense routine

right:                                    'move right
debug "right",cr                          'debug direction to screen
for x = 1 to delay *1                    'distance
  pulsout left_servo,750+speed            'pulse left servo
  pulsout right_servo,750+speed           'pulse right servo
  pause 20                                'pause 20 ms
next                                       'repeat x times
goto sense                                'jump to sense routine
'-----
```


Troubleshooting

- During movement or object encounter the alarm routine begins. The `alarm` routine should execute only once when the Boe-Bot is programmed or reset by pressing the "reset" button on the Board of Education. If you hear the alarm more than once then your BASIC Stamp is resetting. It is resetting because the servos are drawing the power supply low. To fix this, be sure you have installed the 3300 uF capacitor between `Vdd` and `Vss`, and that the servos are connected to the `Vin` supply. If you continue to encounter resets, replace the 4AA batteries. This is a result of the Board of Education being designed before the Boe-Bot, and having only one power supply for logic and motor control.
- Far-away objects are encountered early. The Boe-Bot could find himself barely making his way down a narrow hallway due to object detection of walls on each side. This can be fixed by doing three things: 1) focusing the IR LEDs and receiver in the same direction, and by narrowing their angle by using straws or tape wrapped around the perimeter of each device. IR can be reflected from many objects around the Boe-Bot, making it difficult to determine what object was actually detected; and 2) detuning the potentiometer by slightly turning it away from 38 kHz; and 3) use bigger resistors (2K ohm) for the IR LEDs to reduce their output.
- Boe-Bot turns too much after an object is detected. The amount of turning is determined by the number of `for . . next` loops in the `left:` and `right:` routines. By decreasing the number loops in the `x` variable the turn will become shorter.
- Boe-Bot travels in a wide arc when it should go straight. Particularly at slow speeds, you may notice it is difficult for the Boe-Bot to travel straight. To fix this, set the `speed con 50` to `speed con 0`. Then, if your servos are still centered at a pulse width of 750, the Boe-Bot should sit idle. If it does not then adjust the values for `right_servo` and `left_servo` so your Boe-Bot will travel straight.
- Boe-Bot travels in circles. The `sense:` routine checks the IR receivers in the following order: 1) both left and right are checked together; 2) left is checked alone; and 3) right is checked alone. When an object is detected on both IR receivers the Boe-Bot backs up and turns right. If objects are detected by both sensors then it will continue to move in a circle. Try using the BASIC Stamp's `random` command to decide which direction the Boe-Bot should go after both IR receivers detect a signal.

4

Experiment #4: Infrared Object Detection



Challenge!

1. Use speaker to signal which of the infrared receivers detected an object.
2. Use the BASIC Stamp's EEPROM to log the detection of objects on the right and left receivers. This requires using the `read` and `write` statements.
3. Add two photoresistors to your Boe-Bot. Follow light *and* avoid objects.



Parts Listing

All components (next page) used in the Robotics experiments are readily available from common electronic suppliers. Customers who would like to purchase a complete kit may also do so through Parallax. To use this Robotics curriculum you need three items: 1) a BASIC Stamp II module (available alone, or in the Board of Education

- Full Kit); 2) a Board of Education (available alone or in a Board of Education Full Kit); and 3) the Robotics Parts Kit. The best setup consists of the Board of Education Full Kit and the Robotics Parts Kit.

Board of Education Kits

The BASIC Stamp II (BS2-IC) is available separately or in the Board of Education Full Kit. If you already have a BS2-IC module, then purchase the Board of Education Kit. Individual pieces may also be ordered using the Parallax stock codes shown below.

Board of Education – Full Kit (#28102)

Parallax Code#	Description	Quantity
28150	Board of Education	1
800-00016	jumper wires (bag of 10)	1
BS2-IC	BASIC Stamp II module	1
750-00008	300 mA 9 VDC power supply	1
800-00003	Serial cable	1

Board of Education Kit (#28150)

Parallax Code#	Description	Quantity
28102	Board of Education	1
800-00016	jumper wires (bag of 10)	1

This printed documentation is very useful for additional background information:

BASIC Stamp Documentation

Parallax Code#	Description	Internet Availability?
27919	BASIC Stamp Manual Version 1.9	http://www.stampsinclass.com
28125	Robotics Text	http://www.stampsinclass.com
27951	"Programming and Customizing the BASIC Stamp Computer"	Table of Contents only from http://www.stampsinclass.com

Appendix A: Parts Listing and Sources

The Robotics experiments require the Robotics Parts Kit (#28124)

Similar to all Stamps in Class curriculum, you need a Board of Education with BASIC Stamp and the Parts Kit. The contents of the Robotics Parts Kit is listed below. If you want to build your own metal chassis then download the drawings from <http://www.stampsinclass>. These replacement parts are available from Parallax but may also be sourced from common electronic suppliers.

Robotics Parts Kit (#28124)

Parallax Code#	Description	Quantity
700-00002	4-40 x 3/8" machine screws	8
700-00009	1" polyethylene ball, pre-drilled	1
700-00011	o-ring tires	2
700-00013	plastic machined wheels	2
700-00016	4-40 x 3/8" flathead machine screws	2
700-00022	aluminum chassis	1
700-00023	1/16" x 1.5" long cotter pin	1
700-00023	4-40 nylon locknuts	10
700-00025	13/32" rubber grommet (fits 1/2" hole)	1
700-00026	9/32" rubber grommet (fits 3/8" hole)	2
700-00027	1/2" double-female standoffs	4
700-00028	4-40 x 1/4" machine screw	8
700-00029	battery holder	1
700-00030	DC power cord with 2.1 mm jack	1
900-00003	Servos (Futaba s-148 or Parallax)	2
350-00006	red LEDs	2
451-00301	3-pin headers	2
604-00009	555 timer IC (LMC555 or equivalent)	1
350-00009	Photoresistors (EG&G Vactec VT935G group B)	2
150-02210	220 ohm resistors	3
150-04710	470 ohm resistors	2
150-01020	1K ohm resistors	5
150-02020	2K ohm resistors	2
150-01030	10K ohm resistors	5
350-00013	infrared receiver (Panasonic PNA4602M or eq.)	2
350-00014	infrared LEDs covered with heat shrink tubing (QT QEC113)	2
900-00001	Piezospaker	1
800-00016	jumper wires (bag of 10)	1

Appendix A: Parts Listing and Sources

201-03080	3300 uF capacitor	1
152-01010	1K ohm potentiometer (Bourns 3352-102 or equiv.)	1
200-01040	0.1 uF capacitors	4
200-01031	0.01 uF capacitors	2

Appendix A: Parts Listing and Sources



Sources

The Parallax distributor network serves approximately 40 countries world-wide. A portion of these distributors are also Parallax-authorized "Stamps in Class" distributors – qualified educational suppliers. Stamps in Class distributors normally stock the BASIC Stamp and Board of Education (#28102 and #28150). Several electronic component companies are also listed for customers who wish to assemble their own Robotics Parts Kit.

Country	Company	Notes
United States	Parallax, Inc. 3805 Atherton Road, Suite 102 Rocklin, CA 95765 USA (916) 624-8333, fax (916) 624-8003 http://www.stampsinclass.com http://www.parallaxinc.com	Parallax and Stamps in Class source. Manufacturer of the BASIC Stamp.
United States	Digi-Key Corporation 701 Brooks Avenue South Thief River Falls, MN 66701 (800) 344-4539, fax (218) 681-3380 http://www.digi-key.com	Source for electronic components. Parallax distributor. May stock Board of Education. Excellent source for components.
Australia	Microzed Computers PO Box 634 Armidale 2350 Australia Phone +612-67-722-777, fax +61-67-728-987 http://www.microzed.com.au	Parallax distributor. Stamps in Class distributor. Excellent technical support.
Australia	RTN 35 Woolart Street Strathmore 3041 Australia phone / fax +613 9338-3306 http://people.enternet.com.au/~nollet	Parallax and Stamps in Class distributor.
Canada	Aerosystems 3538 Ashby St-Laurent, QUE H4R 2C1 Canada (514) 336-9426, fax (514) 336-4383	Parallax distributor and Stamps in Class distributor.

Appendix A: Parts Listing and Sources

Canada	HVW Technologies 300-8120 Beddington Blvd NW, #473 Calgary, AB T3K 2A8 Canada (403) 730-8603, fax (403) 730-8903 http://www.hvwtech.com	Parallax distributor and Stamps in Class distributor.
Germany	Elektronikladen W. Mellies Str. 88 32758 Detmold Germany 49-5232-8171, fax 49-5232-86197 http://www.elektronikladen.de	Parallax distributor and Stamps in Class distributor.
New Zealand	Trade Tech Auckland Head Office, P.O. Box 31-041 Milford, Auckland 9 New Zealand +64-9-4782323, fax 64-9-4784811 http://www.tradetech.com	Parallax distributor and Stamps in Class distributor.
United Kingdom	Milford Instruments Milford House 120 High St., S. Milford Leeds YKS LS25 5AQ United Kingdom +44-1-977-683-665 fax +44-1-977-681-465 http://www.milinst.demon.co.uk	Parallax distributor and Stamps in Class distributor.



Robotic Internet Links

Parallax, Inc.	www.parallaxinc.com
Stamps in Class	www.stampsinclass.com
Digi-key	www.digikey.com (electronics parts and PIC chips)
Jameco	www.jameco.com (electronics parts, motors and BASIC Stamps)
National Semiconductor	www.national.com (they make the LM2940-5)
Tower Hobbies	www.towerhobbies.com (low cost servos)
Scott Edwards Elect.	www.seetron.com (serial LCD modules)
P.A.R.T.S.	www.rdrop.com/users/marvin/ (look at Explorer Bot)
R/C servo circuits	www.turbine.kuee.kyoto-u.ac.jp/~staff/onat/servobasics.html
Seattle Robotics	www.seattlerobotics.org/guide/servos.html
Resistor Color Codes	webhome.idirect.com/~jadams/electronics/resistor_codes.htm
Stepper motors	www.doc.ic.ac.uk/~ih/doc/stepper/
Motor info	mot-sps.com/motor/mtrtutorial/prin/index.html
Simple motor to build	fly.hiwaay.net/~palmer/motor.html (great activity-try it)
BEAM Robotics	nis-www.lanl.gov/robot/
Robots and resources	www.robotics.com/resource.html
Australian Robotics	werple.net.au/~tonymerc/
Honda Humanoid Robot	www.honda.co.jp/english/technology/robot/index.html
Robot behavior levels	ai.eecs.umich.edu/cogarch0/subsump/
Robotics Industry Assoc.	www.robotics.org
Robot Books	www.robotbooks.com
The Robot Store	www.robotstore.com
"AA" Battery Info.	www.eveready.com
Isaac Asimov FAQ	www.clark.net/pub/edseiler/WWW/asimov_FAQ.html



Resistor Color Code

Most common types of resistors have colored bands that indicate their value. The resistors that we're using in this series of experiments are typically "1/4 watt, carbon film, with a 5% tolerance". If you look closely at the sequence of bands you'll notice that one of the bands (on an end) is gold. This is band #4, &

the gold color designates that it has a 5% tolerance.

The resistor color code is an industry standard in recognizing the value of resistance of a resistor. Each color band represents a number and the order of the color band will represent a number value. The first two color bands indicate a number. The third color band indicates the multiplier or in other words the number of zeros. The fourth band indicates the tolerance of the resistor +/- 5, 10 or 20 %.

Color	1 st Digit	2 nd Digit	Multiplier	Tolerance
black	0	0	1	
brown	1	1	10	
red	2	2	100	
orange	3	3	1,000	
yellow	4	4	10,000	
green	5	5	100,000	
blue	6	6	1,000,000	
violet	7	7	10,000,000	
gray	8	8	100,000,000	
white	9	9	1,000,000,000	
gold				5%
silver				10%
no color				20%

Appendix B: Resistor Color Code

A resistor has the following color bands:

- Band #1. = Red
- Band #2. = Violet
- Band #3. = Yellow
- Band #4. = Gold

Looking at our chart above, we see that Red has a value of 2.

So we write: "2".

Violet has a value of 7.

So we write: "27"

Yellow has a value of 4.

So we write: "27 and four zeros" or "270000".

This resistor has a value of 270,000 ohms (or 270k) & a tolerance of 5%.



Appendix C: Changing the Board of Education Voltage Regulator

Prior to June 1999, Parallax produced the Board of Education with a LM7805CV voltage regulator. This voltage regulator is fine for general use, but will cause the BASIC Stamp to reset when used with 6 Volt power supplies and higher current devices like

servos or stepper motors. If your Board of Education has the LM7805CV voltage regulator, contact Parallax and ask for a free BOE Voltage Regulator Upgrade Kit (stock code #28151). This appendix includes instructions to remove the LM7805CV voltage regulator and replace it with the low-drop LM2940. Parallax will send it out for free by U.S. Mail.

The voltage regulator is the only component that has a machine screw and nut sticking up on the circuit board located right next to the "Stamps in Class" logo. To put in the new LM2940 regulator all you need is a small soldering iron, wire cutters, screwdriver and a little solder. Replacing the 7805CV with an LM2940 only takes a few minutes and is well worth the time. Here's how to do it.

Step 1: Remove the 7805 Voltage Regulator

Using your small wire cutters cut the leads of the existing 7805 regulator off flush with the case of the regulator as shown in Figure C.1. When you're done with this step you should have leads still hanging off your circuit board. You don't need to unsolder anything.

Figure C.1: Remove the 7805 regulator by cutting the leads close to the regulator case and unscrewing the small nut.

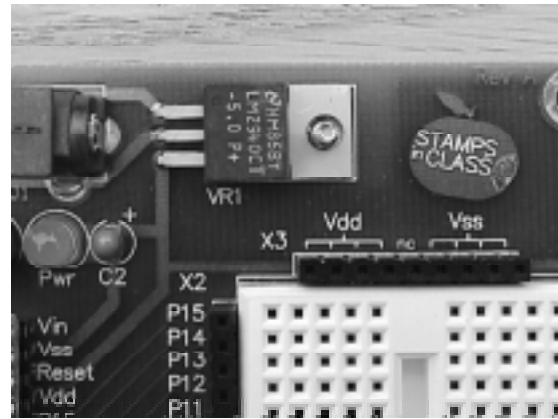


Appendix C: Changing the Board of Education Voltage Regulator

Step 2: Install LM2940 Regulator

Put the LM2940 regulator where the old one was, trim the leads off a little and put the screw back in and tighten the nut. Line it up just like the old one was so the leads from the new regulator are laying on top of the ones you left when you cut out the old 7805 regulator. Figure C.2 shows what things should look like at this point.

Figure C.2: Put the LM2940 in place of the 7805. The leads will line up with the old leads of the 7805,



Step 3: Solder Regulator on Board of Education

Heat up the soldering iron if you haven't done so yet and get ready to solder the new leads from the LM 2940 to the leads they are laying on top of. Before you solder trim the leads of the LM 2940-5 so they don't overhang the existing leads. Solder the leads together and you are finished. Be sure you tighten the 4-40 screw and nut that mounts the regulator to the Board of Education.



Appendix D: Boe-Bot Competition Rules

If you're planning a competition for autonomous robots, these rules are provided courtesy of Seattle Robotics Society (SRS). <http://www.seattlerobotics.org/>

(NOTE: The standard SRS rules have been reproduced here unmodified to encourage contests between different groups, however a group might choose to build smaller courses to save money and simplify transportation if all of their robots are similar in size to the Boe-Bot.)

Contest #1: Robot Floor Exercise

Purpose

The floor exercise competition is intended to give robot inventors an opportunity to show off their robots or other technical contraptions.

Rules

The rules for this competition are quite simple. A 10 foot by 10 foot flat area is identified, preferably with some physical boundary. Each contestant will be given a maximum of 5 minutes in this area to show off what it can do. The robot's controller can talk through the various capabilities and features of the robot. As always, any robot that could damage the area or pose a danger to the public will not be allowed. Robots need not be autonomous, but it is encouraged. Judging will be determined by the audience, either indicated by clapping (the loudest determined by the judge), or some other voting mechanism.

Contest #2: Line Following Rules

Objective

To build an autonomous robot that begins in Area "A" (at position "S"), travels to Area "B" (completely via the line), then travels to the Area "C" (completely via the line), then returns to the Area "A" (at position "F"). The robot that does this in the least amount of time (including bonuses) wins. The robot must enter areas "B" and "C" to qualify. The exact layout of the course will not be known until contest day, but it will have the three areas previously described.

Skills Tested

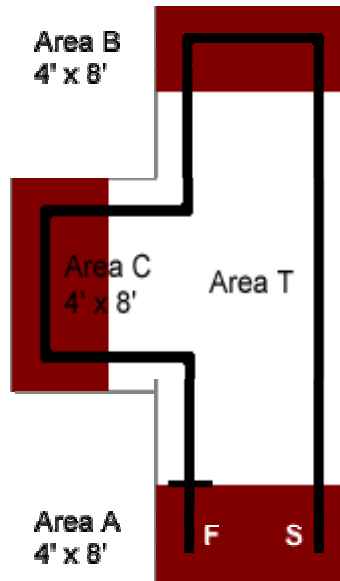
Appendix D: Boe-Bot Competition Rules

The ability to recognize a navigational aid (the line) and use it to reach the goal.

Maximum Time to Complete Course

Four minutes.

Example Course



All measurements in the example course are approximate. There is a solid line dividing Area "A" from Area "T" at position "F". This indicates where the course ends. The line is black, approximately 3/4 inches wide and spaced approximately two feet from the walls. All curves have a radius of at least one foot and at most three feet. The walls are 3 1/2 inches high and surround the course. The floor is white and made of either paper or Tyvec. Tyvec is a strong plastic used in mailing envelopes and house construction.

Positions "S" and "F" are merely for illustration and are not precise locations. A Competitor may place the robot anywhere in Area "A", facing in any direction when starting. The robot must be completely within Area "A". Areas "A", "B" and "C" are not (a different color) on the actual course.

Appendix D: Boe-Bot Competition Rules

Scoring

Each contestant's score is calculated by taking the time needed to complete the course (in seconds) minus 10% for each "accomplishment." The contestant with the lowest score wins.

Accomplishments	Reduction
Stops in area A after reaching B and C	10%
Does not touch any walls	10%
Starts on command	10%

("Starts on command" means the robot starts with an external, non-tactile command. This could, for example, be a sound or light command.)

Contest #3: Maze Following

Purpose

The grand maze is intended to present a test of navigational skills by an autonomous robot. The scoring is done in such a way as to favor robots which are either brutally fast or which can learn the maze after one pass. The object is for a robot which is set down at the entrance of the maze to find its way through the maze and reach the exit in the least amount of time.

Physical Characteristics

The maze is constructed of 3/4" shop plywood. The walls are approximately 24 inches high, and are painted in primary colors with glossy paint. The walls are set on a grid with 24-inch spacing. Due to the thickness of the plywood and limitations in accuracy, the hallways may be as narrow as 22-inches. The maze can be up to 20-foot square, but may be smaller depending on the space available for the event.

The maze will be set up on either industrial-type carpet or hard floor (depending on where the event is held). The maze will be under cover, so your robot does not have to be rain proof; however, it may be exposed to various temperatures, wind, and lighting conditions. The maze is a classical two-dimensional proper maze: there is a single path from the start to the finish and there are no islands in the maze. Both the entrance and exit are located on outside walls. Proper mazes can be solved by following either the left wall or the right wall. The maze is carefully designed so that there is no advantage if you follow the left wall or the right wall.

Robot Limitations

The main limit on the robot is that it be autonomous: once started by the owner or handler, no interaction is allowed until the robot emerges from the exit, or it becomes hopelessly stuck. Obviously the robot needs to be small enough to fit within the walls of the maze. It may touch the walls, but may not move the walls to its advantage -no bulldozers. The judges may disqualify a robot which appears to be moving the walls excessively. The robot must not damage either the walls of the maze, nor the floor. Any form of power is allowed as long as local laws do not require hearing protection in its presence or place any other limitations on it.

Scoring

Each robot is to be run through the maze three times. The robot with the lowest single time is the winner. The maximum time allowed per run is 10 minutes. If a robot cannot finish in that amount of time, the run is stopped and the robot receives a time of 10 minutes. If no robot succeeds in finding the exit of the maze, the one that made it the farthest will be declared the winner, as determined by the contest's judge.

Logistics

Each robot will make one run, proceeding until all robots have attempted the maze. Each robot then does a second run through the maze, then the robots all do the third run. The judge will allow some discretion if a contestant must delay their run due to technical difficulties. A robot may remember what it found on a previous run to try to improve its time (mapping the maze on the first run), and can use this information in subsequent runs-as long as the robot does this itself. It is not allowed to manually "configure" the robot through hardware or software as to the layout of the maze.