SumoBot : Mini-Sumo Robotics

Version 1.1

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

PARALLAX INC

Warranty

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

14-Day Money Back Guarantee

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

Copyrights and Trademarks

Disclaimer of Liability

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

Internet Access

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

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

Internet BASIC Stamp Discussion List

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

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

Table of Contents

Introduction

Like its human counterpart, robot Sumo was born and thrives in Japan. It was introduced to the United States in the early 1990's by Dr. Mato Hattori. One of the early American adopters of robot Sumo was noted Seattle Robotics Society member, Bill Harrison, who organized some of the first U.S. robot Sumo tournaments.

While things started out very slowly, robot Sumo eventually caught on. Bill created a "lightweight" class that matched the Japanese physical dimensions of 20 cm by 20 cm, but reduced the mass from three kilograms (6.6 pounds) to one kilogram (2.2 pounds). The intention was to reduce the sophistication of the components required to construct a working Sumo robot. Those early contests didn't have much in the way of corporate support with prizes, so Bill resorted to offering 30 hours of his own machine-shop services to the winner.

As luck would have it, Bill's friend Robert Jorgensen won that first contest prize. Since Robert already had a winning Sumo robot, he suggested that they build a smaller version, about half the size and weight of the lightweight class to be used as a robot Sumo demonstrator. The result of their work was a very small Sumo robot that measured just 8 cm by 8 cm and mass about 240 grams. Bill took that first small Sumo to a contest in San Francisco and actually won the lightweight competition – against bigger and heavier robots. The Mini-Sumo robot class was born.

The Mini-Sumo dimensions (10 cm x 10 cm) and mass (500 grams) were formalized and Bill published adapted Japanese robot Sumo rules on his Sine Robotics web site (and mirrored on many other sites). Through Bill's tireless efforts and nearly ten years of travel – often toting more than 20 robots in his bags – Mini-Sumo robotics has grown to a favorite activity among robot clubs all across the United States.

Recognitions

Many Mini-Sumo designs – especially the dual-wheel-and-scoop concept – can be traced back to Bill Harrison's early efforts to promote Mini-Sumo robotics competition. Parallax also recognizes Bill Boyer of the Dallas Personal Robotics Group for his version of the dual-wheel-and-scoop design that was refined and developed into the Parallax SumoBot described in this text.

This curriculum was authored by Parallax and contains material by several contributors, including Parallax engineers Jon Williams, Ken Gracey and Andy Lindsay, as well as Bill Wong of Pennsylvania. Bill is an editor with Electronic Design magazine and a serious BASIC Stamp robotics enthusiast. Bill enjoys creating BASIC Stamp powered robots with his daughter who has gone on to win several county and state awards with her maze solving robotics projects.

Audience and Teacher's Guide

The SumoBot curriculum was created for ages 12+ as a complimentary text to the "Robotics" and "Advanced Robotics" guides. Like all Stamps in Class curriculum, this series of experiments teaches new techniques and circuits with minimal overlap between the other texts. The general topics introduced in this series are: basic SumoBot locomotion under program control, edge avoidance and opponent detection based on a variety of sensor inputs, as well as navigation opponent hunting using programmed artificial intelligence. Each topic is addressed in an introductory format designed to impart a conceptual understanding along with some hands-on experience. Those who intend to delve further into industrial technology, electronics or robotics are likely to benefit significantly from initial experiences with these topics.

If your experience with the SumoBot differs from our expectations, please let us know at stampsinclass@parallaxinc.com. The SumoBot text presently has no teacher's guide. Based on demand we may elect to produce the answers to challenge questions posed in this text.

Educational Concepts from the SumoBot

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

- Interaction between mechanical and electrical systems and the ability to tune hardware or adjust software to obtain desired results.
- Intermediate programming skills with the BASIC Stamp 2. An efficient SumoBot program makes use of efficient BASIC Stamp programming techniques with **BRANCH** and **LOOKDOWN**, variable aliasing, general sound programming practices (constant/variable definitions that allow for program customization in just a few places rather than throughout an entire program).
- A step-wise process which starts with the basics and builds to something more complex and ultimately more useful.

Copyright and Reproduction

Experiment #1: Assembling the SumoBot

There's an old axiom among robot enthusiasts that states, "It's harder than it looks...." The purpose of this statement is not to alarm or dissuade the new robot builder, but simply to remind him or her that robotics, even on a small scale, is a serious endeavor and shouldn't be taken lightly.

Follow the construction steps carefully and you'll have your SumoBot running and ready to compete in a couple of hours.

The SumoBot is capable of doing any of the things other rolling robots can do. As you learn to program the SumoBot for competition, you'll become a more proficient – and *efficient* – programmer as you learn to exploit the BASIC Stamp's capabilities. The SumoBot demonstrates the importance of a PBASIC program that uses constants and variables, as well state-oriented design. A well-designed program means you can easily tune the software for the right mechanical control in just a few places rather than rewriting your entire program.

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

The SumoBot is a purpose-built rolling robot, much like its general-purpose cousin the Parallax BOE-Bot. While they share the same differential drive mechanism and the use of sensors, the SumoBot design meets the specific criteria defined by Mini-Sumo competition rules:

- Maximum [width and depth] dimensions of 10 cm by 10 cm
- Maximum mass of 500 grams

The standard SumoBot comes with two sets of sensors: two QTI line sensors to keep the SumoBot on the playing surface and two sets of infrared emitters/detectors used to locate its opponent. Advanced users may expand on the standard SumoBot design by adding ultrasonic or IR distance measuring sensors, tilt sensing and motor current sensing.

Let's Build the SumoBot

The SumoBot chassis design leaves little room for mechanical alteration; a requirement to stay within standard Mini-Sumo competition rules. Where the student is encouraged to explore changes is in the types of sensors used to detect the Sumo ring border and the opponent and the software algorithms used to control the SumoBot's behaviors. The demonstration code provided with this text will focus on the standard sensors provided in the SumoBot kit. Future supplements may be published that deal with advanced sensors and techniques for incorporating them into the SumoBot control logic.

Tools Required

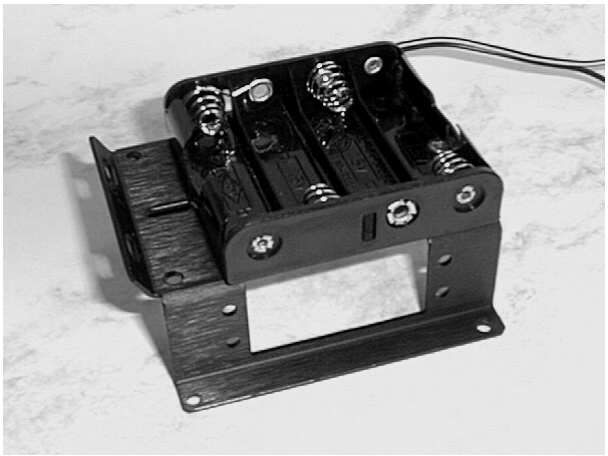
You'll need a screwdriver and a pair of needle-nose pliers and to build the SumoBot.

A Note About Parts in the SumoBot Kit

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

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

Step #1: Install the Battery Box



Parts Required:

- Battery Box
- (2) 4/40 3/8" long flat-head countersunk machine screws
- (2) 4/40 nuts
- SumoBot chassis

Stand the SumoBot chassis on its PCB mounting ears (upside down).

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

Step #2: Install the Servo Motors

Parts Required:

- (2) Modified servos
- (8) 4/40 3/8" long pan-head machine screws
- (8) 4/40 nuts
- SumoBot chassis

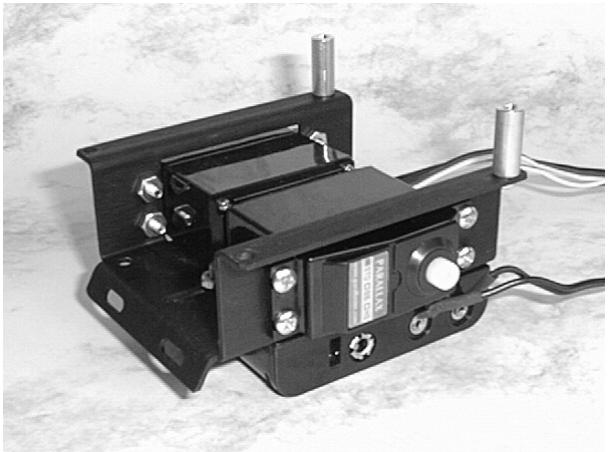
Using four (4) 4/40 3/8" pan-head machine screws and (4) 4/40 nuts, attach each servo motor to the chassis. The easiest way to do this is to hold the nut with one finger while turning the screwdriver with the other hand. Remove the horn from the servos, saving the black screws.

Step #3: Install the Rear SumoBoard Stand-offs

Parts Required:

- (2) 5/8" round standoffs
- (2) 4/40 3/8" long pan-head machine screws
- SumoBot chassis

Using a 4/40 3/8" pan-head machine screw, attach each stand-off to the rear of the SumoBot chassis.

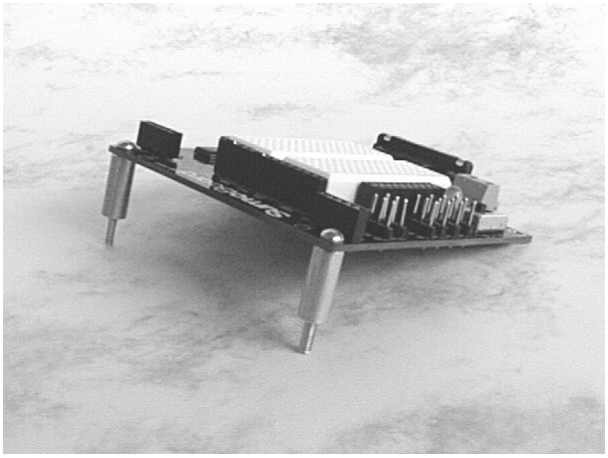


Step #4: Install the Front SumoBoard Stand-offs

Parts Required:

- (2) 5/8" round standoffs
- (2) 4/40 1" long pan-head machine screws
- SumoBoard PCB

Using a 4/40 1" pan-head machine screw, attach each standoff to the front mounting holes of the SumoBoard PCB.



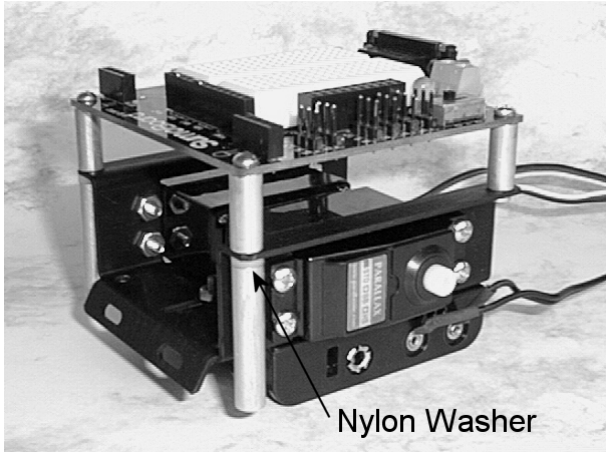
Step #5: Mount the SumoBoard

Parts Required:

- SumoBoard PCB
- (2) 4/40 3/8" long pan-head machine screws
- (2) 1-1/4" round stand-offs
- (2) Nylon washers
- SumoBot chassis

Feed the ends of the 1" long pan-head machine screws which you installed in Step #4 through the front mounting holes on the SumoBot chassis. Secure the rear side of the SumoBoard PCB to the 5/8" standoffs with two (2) 3/8" pan-head machine screws.

Holding the chassis upside-down, place a nylon washer onto the end of each 1" long pan-head machine screws, then secure by threading on the 1-1/4" round standoff.



Step #6: Prepare the Wheels

Parts Required:

- (2) SumoBot wheels
- (2) SumoBot rubber tires

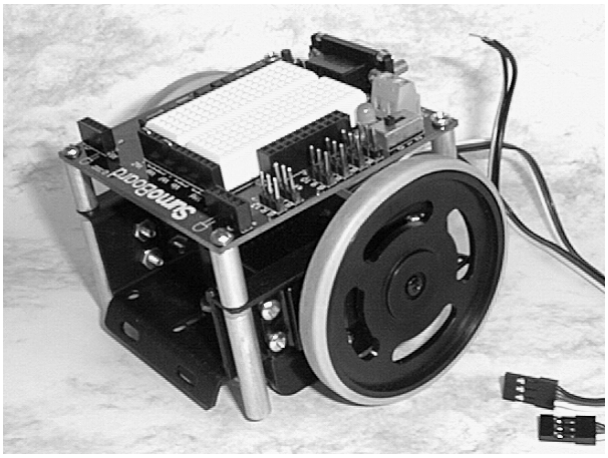
Stretch a "tire" of each wheel and adjust so that the "tire" is centered across the wheel.

Step #7: Mount the Wheels

Parts Required:

- (2) Prepared wheels/tires
- (2) Black servo-horn screws
- SumoBot chassis

Carefully press each prepared wheel onto the servo splines. Secure each wheel with the small black Phillips head screw.

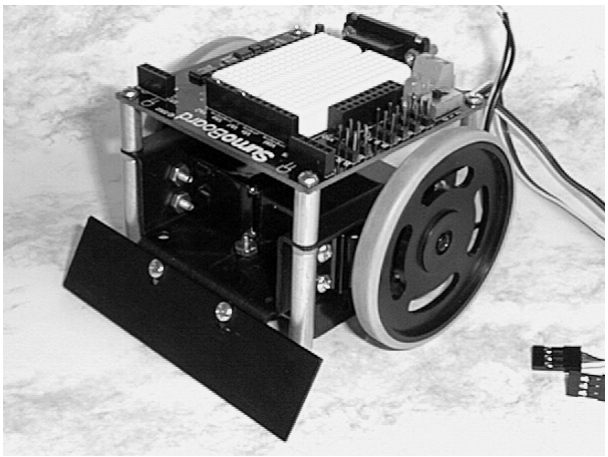


Step #8: Mount the Scoop

Parts Required:

- SumoBot scoop
- (2) 4/40 1/4" long pan-head machine screws
- (2) 4/40 nuts
- SumoBot chassis

Using two (2) 4/40 1/4" pan-head machine screws and (2) 4/40 nuts, attach the scoop to the SumoBot chassis. Carefully center the scoop before tightening the screws and nuts.



Step #9: Install Line Sensor Wires

Parts Required:

- (2) 10" 3-pin extension cables
- SumoBot chassis

Carefully feed each 10" 3-pin extension cable through the center chassis slot.

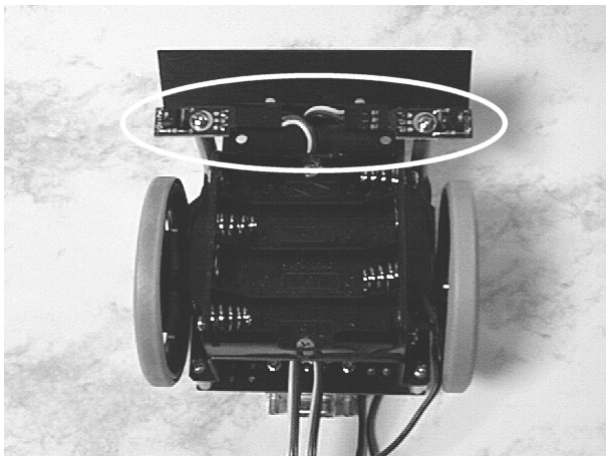
Step #10: Install the QTI Line Sensors

Parts Required:

- (2) QTI line sensors
- (2) 4/40 1/4" long pan-head machine screws
- SumoBot chassis

Using two (2) 4/40 1/4" pan-head machine screws, attach the QTI line sensors to the 1-1/4" round stand-offs.

Connect the ends of the 10" 3-pin extension cables to the QTI line sensors, noting the polarity markings (B-R-W) on the QTI sensors.



Step #11: Make the Connections

Plug the servo motors and QTI sensors into the SumoBoard connectors as indicated below. Note that the "B" pin on each connector is for the black wire.

X7 = Left Servo Motor
X6 = Right Servo Motor
X5 = Left QTI Line Sensor
X4 = Right QTI Line Sensor

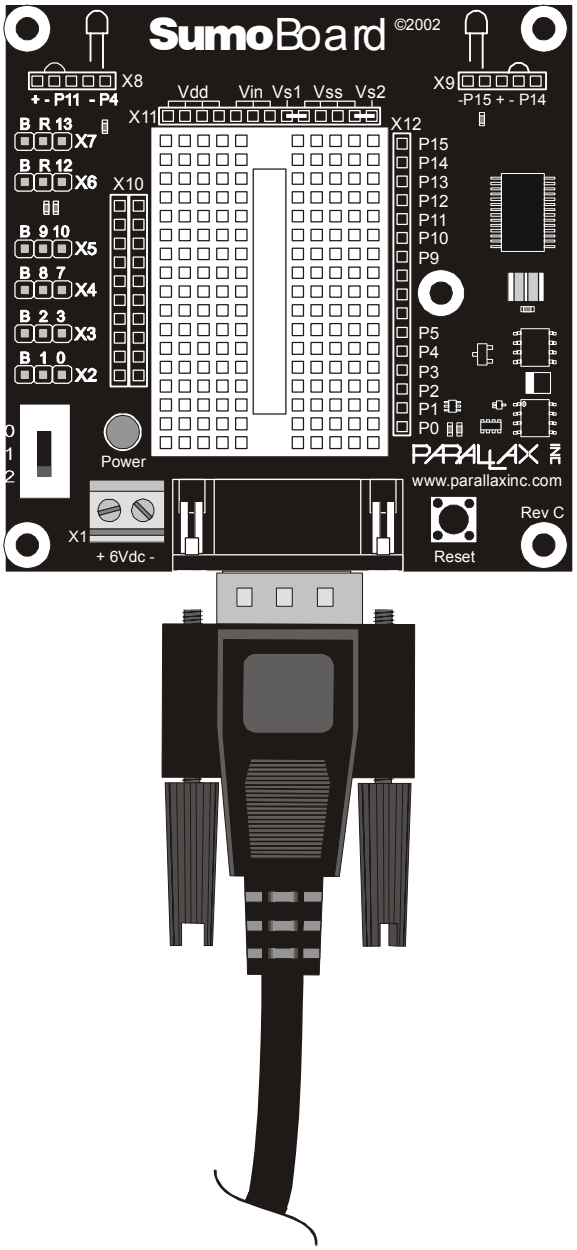
Hint: The servo and sensor cables can be secured to the left-front SumoBoard stand-off with a wire twist-tie.

Using solid 22-guage wire, connect Vs1 to Vss and Vs2 to Vss as shown in the illustration to the right. These jumpers provide the ground connections (Vss) to the servos motors.

Connect the SumoBot's battery pack wires to connector X1 by loosening the screws in the top. The battery pack's white-striped lead connects to the SumoBoard's **+** terminal. The wires slip into the side of the terminal block.

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

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



Experiment #2: SumoBot Locomotion

The first task of any Mini-Sumo robot is to move – most competition rules do not allow the robot to stop (without competitor contact) for more than a few seconds. In this experiment you will learn how to get the SumoBot moving and learn to take control over its motion.

How a Servo Works

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

The SumoBot's motion is controlled using two modified servo motors using a process called differential drive. The modification "tricks" the feedback circuitry so that the servo will stop only when it receives a centering command; it also allows the servo to continuously rotate in either direction. When both motors are turning in the same direction, the SumoBot will move in that direction. When the SumoBot's motors turn in different directions, the chassis will rotate. The rate of movement or rotation is determined by motor speeds.

Time Measurements and Voltage Levels

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

Table 2.1: Milliseconds and Microseconds and SumoBoard PCB Voltage Labels

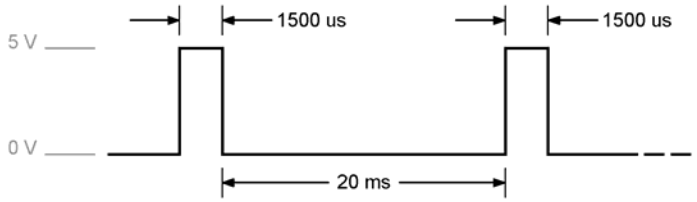
Milliseconds and Microseconds
$1\,\text{ms} = \frac{1}{1000}\,\text{s} = 1\times10^{-3}\,s$ $1\,\mu\text{s} = \frac{1}{1{,}000{,}000}\,\text{s} = 1\times10^{-6}\,s$
Voltages and SumoBoard PCB Labels
Vss = 0 V (ground)   Vdd = 5 V (regulated)   Vin = 6 V (unregulated)

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

Only use the Vdd sockets above the SumoBoard's breadboard for the Activities in this workbook. Do not use the Vdd on the 20-pin App-Mod header.

The control signal the BASIC Stamp sends to the servo's control line is called a "pulse train," and an example of one is shown in Figure 2.1. The BASIC Stamp can be programmed to produce this waveform using any of its I/O pins. In this example, the BASIC Stamp sends a 1500 us pulse to P13 (left servo) and P12 (right servo). When the pulse is done being executed the signal pin is low. Then, the BASIC Stamp creates a 20 ms pause.

Figure 2.1: Servo Pulse Train



This pulse train has a 1500 us high time and a 20 ms low time. The high time is the main ingredient for controlling a servo's motion, and it is most commonly referred to as the pulse width. Since these pulses go from low to high (0 V to 5 V) for a certain amount of time, they are called positive pulses. Negative pulses would involve a resting state that's high with pulses that drop low.

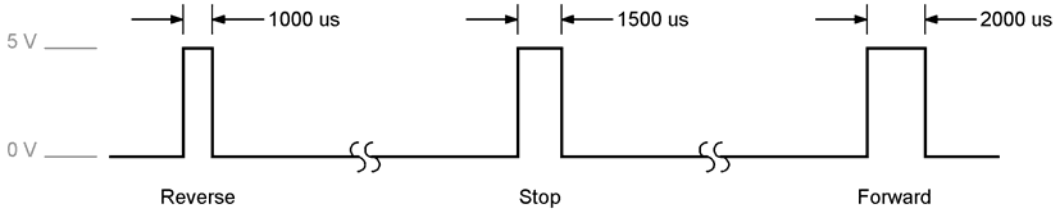
The ideal pause between servo pulses can be about 10 to 40 ms without adversely affecting the servo's performance.

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

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

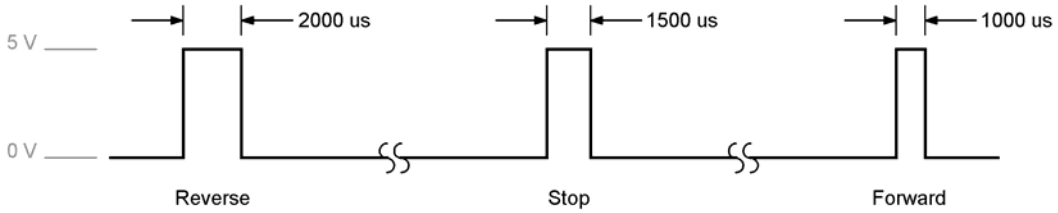
A pulse width of 1500 us (normally, the centering command) will cause the modified servo to stop. To make the servo turn we must give change the pulse width toward either end of the standard control range of 1000 us to 2000 us. For the left servo, the control signaling works as shown in Figure 2.2.

Figure 2.2: SumoBot Left Servo Control Pulses



Since the right side servo motor is physically mirrored from the left, its control signals are as well. Figure 2.3 shows the control signaling for SumoBot's right servo.

Figure 2.3: SumoBot Right Servo Control Pulses



For pulses between the 1500 us stop point and the extremes on either end of the control range, there is a degree of speed control. This range is not linear, however, and at pulse widths just outside the stop band, servo current increases dramatically. At some points in the control range, the servo current can go high enough to cause an excessive load on the BASIC Stamp's regulator circuitry, causing it to reset or behave erratically.

For Mini-Sumo competition, precise speed control is not a requirement. The goal is to find the opponent and move quickly toward him.

Open the BASIC Stamp Windows editor.[1] Load the following program that will test the SumoBot's motors.[2]

```
' -----[ Title ]-----------------------------------------------------------------
' Mini-Sumo 2.1 : SumoBot Motor Test
' {$STAMP BS2}

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

LMotor          CON     13                      ' left servo motor
RMotor          CON     12                      ' right servo motor

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

LFwdFast        CON     1000                    ' left motor forward; fast
LFwdSlow        CON      800                    ' left motor forward; slow
```

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

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

```
LStop           CON      750                    ' left motor stop
LRevSlow        CON      700                    ' left motor reverse; slow
LRevFast        CON      500                    ' left motor reverse; fast

RFwdFast        CON      500                    ' right motor forward; fast
RFwdSlow        CON      700                    ' right motor forward; slow
RStop           CON      750                    ' right motor stop
RRevSlow        CON      800                    ' right motor reverse; slow
RRevFast        CON     1000                    ' right motor reverse; fast

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

pulses          VAR     Byte                    ' counter for motor control

' -----[ Initialization ]--------------------------------------------------------

Start_Delay:
  PAUSE 2000                                    ' time to disconnect cable

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

Main:
  FOR pulses = 1 TO 125                         ' forward slow - about 12 inches
    PULSOUT LMotor, LFwdSlow
    PULSOUT RMotor, RFwdSlow
    PAUSE 20
  NEXT

  FOR pulses = 1 TO 110                         ' pivot turn - left wheel (ccw)
    PULSOUT LMotor, LStop
    PULSOUT RMotor, RFwdSlow
    PAUSE 20
  NEXT

  FOR pulses = 1 TO 50                          ' forward fast
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RFwdFast
    PAUSE 20
  NEXT

  FOR pulses = 1 TO 30                          ' spin turn - clockwise
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
```

```
Hold_Position:
  PULSOUT LMotor, LStop
  PULSOUT RMotor, RStop
  PAUSE 20
  GOTO Hold_Position
```

Move the SumoBot power switch to Position 2, then download the code using the Run I Run menu or by pressing the ►button on the toolbar. As soon as the program is downloaded, remove the programming cable from the SumoBoard. This program runs through key motion tests, then stops the SumoBot. When the servos stop moving (this will happen almost instantaneously) move the power switch to Position 0 (off).

How It Works

This program starts – as well-coded programs do – by defining connection and value constants used in the program. This methodology creates programs that are easier to read, maintain and debug. In the case of the SumoBot, the motor connections are on pins 13 (left) and 12 (right). For robot Sumo, we don't need finite speed control; simply stop, slow and fast. Speed constants for each motor are defined and can be tuned for motor variances and when changing motor types.

The Initialization section introduces a small delay to allow the programming cable to be removed from the SumoBot. It's best to untether the SumoBot while doing any movement testing.

The core of the program, at the label **`Main`**, is broken down into five sections:

1. Move forward slowly
2. Pivot turn on left wheel
3. Move forward quickly
4. Spin turn (rotates SumoBot around its own center)
5. Hold position

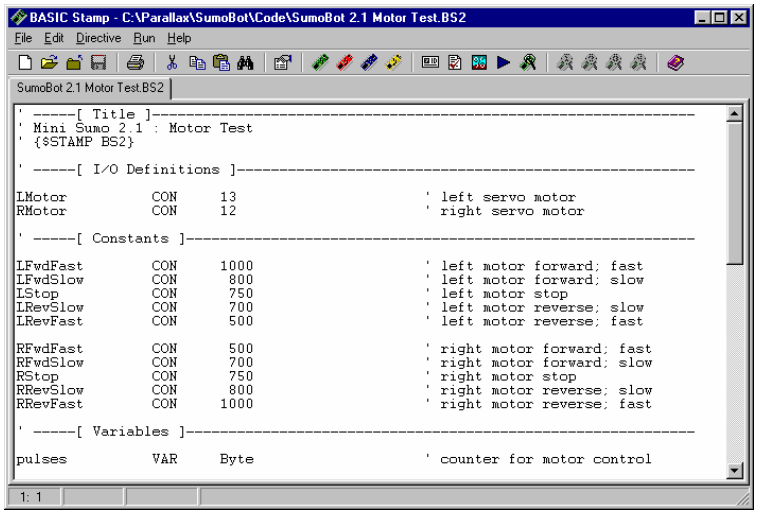
Note: If the SumoBot starts by backing up, the motor connections are reversed. Move the power switch to Position 0 (off), change the connections, then move the power switch back to Position 2 to retest. You could change the firmware as well, but then you'll need to change all the SumoBot programs included in this kit.

Each section is constructed using a **`FOR-NEXT`** loop to give the motors enough time to perform the actual movements. You may notice that the SumoBot does not move exactly as the code dictates that it should. Do not be alarmed. Small variations in motors may cause the SumoBot to veer to one side or the other when it should be moving straight.

This condition can be "corrected" (more precisely, tuned) by modifying the speed constants for the motors. If, for example, the SumoBot veers to the left when it should be moving straight, you may want to reduce the right motor speed a bit to correct the path of travel.

The same holds true for the turns. In this case, the motor speed is not the culprit. To adjust turns, modify the loop counter end-point and retest. If the SumoBot turns too much, reduce the loop count. If it doesn't turn quite enough, increase the loop count.

Figure 2.3: Windows Editor with SumoBot Program 2.1 Motor Test



Challenges

1. Modify the motor speed constants so that your SumoBot travels straight at low and high speeds.

2. Determine the proper loop count to cause the SumoBot to turn 30 degrees, 45 degrees and 90 degrees.

3. Using the information in Challenge 2, program the SumoBot to travel in the following patterns:
   - Square
   - Triangle
   - Figure-8

Experiment #3: SumoBot Sensors – Border Detection

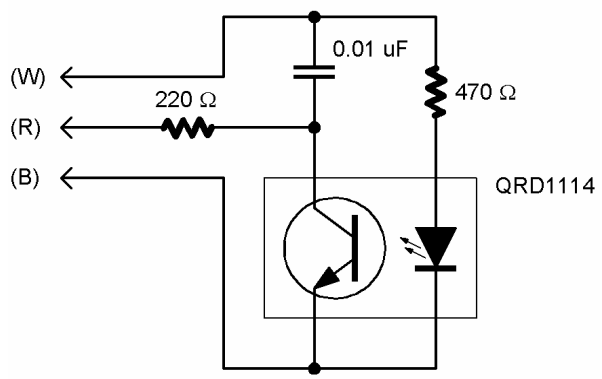
Once the SumoBot is moving, the next task is to scan the playing surface to make sure that it doesn't drive itself out of the ring. The task is accomplished by two specialized line detection sensors called QTIs. The QTI uses a reflective infrared sensor to allow the SumoBot to "look" for the ring's border.

Line Sensor Theory

The Parallax QTI uses a QRD1114 infrared (IR) reflective sensor to determine the reflectivity of the surface below it. When the SumoBot is over the black playing field or start lines (Shikiri), the reflectivity is very low; when the QTI is over the white border (Tawara), the reflectivity is very high and will cause a different reading from the sensor.

Figure 3.1 shows the schematic for the Parallax QTI line sensor.

Figure 3.1: QTI Line Sensor Schematic



The QTI sensor is activated by placing 5 V (Vdd) on the W pin. This will cause current to flow through the 470 ohm resistor to the LED side of the QRD1114. IR light reflecting of the surface below will cause a change in the ability for current to flow through the phototransistor side of the QRD1114. The transistor, in effect, behaves like an IR controlled resistance.

The BASIC Stamp has a specific command designed to read a variable resistance called **RCTIME**. When coupled with a capacitor, the BASIC Stamp can measure a variable resistance by timing the charge or discharge rate of the connected capacitor.
After the QTI is activated, the capacitor is discharged by bringing the R line high and holding it for about one millisecond. **RCTIME** is then used to measure the time required to charge the capacitor to a specified level. This timing will be controlled by the current flow through the phototransistor side of the QRD1114. When over the black playing field, the phototransistor current flow will be very low so the capacitor will take a long time to charge, hence **RCTIME** will return a large value. When the QTI is positioned over the white border line, the current flow through the phototransistor is high, so the capacitor charge time is fast and **RCTIME** returns a small value.

Load and run program 3.1 to test and evaluate the QTI sensors.

```
' -----[ Title ]-----------------------------------------------------------
' Mini-Sumo 3.1 : Line Sensor Test
' {$STAMP BS2}

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

LLineSnsrPwr    CON     10                      ' left line sensor power
LLineSnsrIn     CON      9                      ' left line sensor input
RLineSnsrPwr    CON      7                      ' right line sensor power
RLineSnsrIn     CON      8                      ' right line sensor input

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

ClrEOL          CON     11                      ' clear to end of line (DEBUG)

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

leftSense       VAR     Word                    ' left sensor raw reading
rightSense      VAR     Word                    ' right sensor raw reading

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

Read_Left:
  HIGH LLineSnsrPwr                             ' activate sensor
  HIGH LLineSnsrIn                              ' discharge QTI cap
  PAUSE 1
  RCTIME LLineSnsrIn, 1, leftSense              ' read sensor value
  LOW LLineSnsrPwr                              ' deactivate sensor

Read_Right:
```

```
  HIGH RLineSnsrPwr                              ' activate sensor
  HIGH RLineSnsrIn                               ' discharge QTI cap
  PAUSE 1
  RCTIME RLineSnsrIn, 1, rightSense              ' read sensor value
  LOW RLineSnsrPwr                               ' deactivate sensor

Display:
  DEBUG Home
  DEBUG "Left ", TAB, "Right", CR
  DEBUG "-----", TAB, "-----", CR
  DEBUG DEC leftSense, ClrEOL, TAB, DEC rightSense, ClrEOL

  PAUSE 100
  GOTO Read_Left
```


How It Works

This program starts by activating the left QTI line sensor, then bringing the R line high to discharge the onboard capacitor. A one millisecond **PAUSE** gives the capacitor plenty of time to discharge through the 220 ohm resistor.

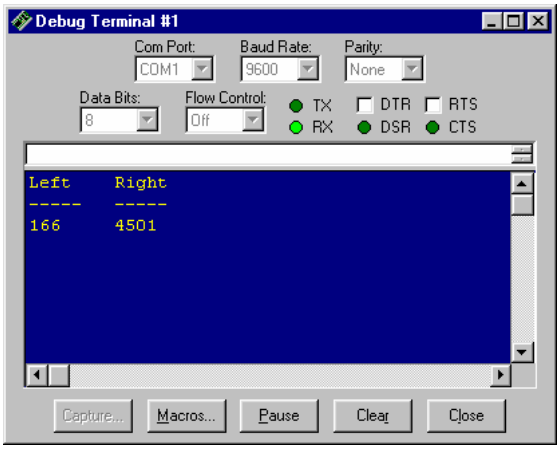
RCTIME makes the R line an input and allows the capacitor to begin to charge. While the capacitor is charging, **RCTIME** increments an internal counter. When the BASIC Stamp sees approximately 1.5 volts on the R line, the counter value is placed in the output variable called *leftSense*. At this point the process is complete and the QTI is deactivated.

The process is repeated for the right-side QTI and **DEBUG** is used to display the values. Figure 3.2 shows the output of the program with the left QTI over the Tawara (white border) line and the right QTI over the black playing field. Notice the dramatic difference between the two values.

Run the program (using Position 1 of the power switch) and record the values from your SumoBot QTI sensors in the space below:

	Black	White
Left QTI		
Right QTI		

Figure 3.2: QTI Test Program Output



Don't worry about small discrepancies between the QTI values over the same color – this is due to minor variations in components and won't adversely affect the SumoBot's performance.

For convenience in a competition program, the QTI code should be moved into a subroutine that can be called from any point in the program. The routine should also be developed to return either a True (1) or False (0) value if the QTI is over the Sumo ring border. To see this in action, load and run program 3.2.

```
' -----[ Title ]-----------------------------------------------------------
' Mini-Sumo 3.2 : Line Sensors Routine
' {$STAMP BS2}

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

LLineSnsrPwr    CON     10                      ' left line sensor power
LLineSnsrIn     CON      9                      ' left line sensor input
RLineSnsrPwr    CON      7                      ' right line sensor power
RLineSnsrIn     CON      8                      ' right line sensor input

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

ClrEOL          CON     11                      ' clear to end of line (DEBUG)

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

```
leftSense       VAR     Word                    ' left sensor raw reading
rightSense      VAR     Word                    ' right sensor raw reading
lineBits        VAR     Nib                     ' decoded sensors value
lineLeft        VAR     lineBits.Bit1
lineRight       VAR     lineBits.Bit0

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

Main:
  GOSUB Read_Line_Sensors
  DEBUG Home, "LR", CR                          ' show sensor readings
  DEBUG BIN2 lineBits, CR, CR

  ' Sumo movement
  BRANCH lineBits, [Go_Fwd, Spin_Left, Spin_Right, About_Face]

Go_Fwd:
  DEBUG "Continue forward", ClrEOL
  GOTO Main

Spin_Left:
  DEBUG "Spin Left", ClrEOL
  GOTO Main

Spin_Right:
  DEBUG "Spin Right", ClrEOL
  GOTO Main

About_Face:
  DEBUG "Turn around", ClrEOL
  GOTO Main

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

Read_Line_Sensors:
  HIGH LLineSnsrPwr                             ' activate sensors
  HIGH RLineSnsrPwr
  HIGH LLineSnsrIn                              ' discharge caps
  HIGH RLineSnsrIn
  PAUSE 1
  RCTIME LLineSnsrIn, 1, leftSense              ' read left sensor
  RCTIME RLineSnsrIn, 1, rightSense             ' read right sensor
  LOW LLineSnsrPwr                              ' deactivate sensors
  LOW RLineSnsrPwr
```

```
' convert readings to bits
LOOKDOWN leftSense, >=[1000, 0], lineLeft     ' 0 = on black, 1 = on line
LOOKDOWN rightSense, >=[1000, 0], lineRight
RETURN
```

How It Works

This program takes the working QTI code and incorporates into a subroutine. The end of the subroutine converts the analog values from the QTI sensors to a single nibble value that contains the status of both sensors. Incorporating both sensor readings into a single value streamlines the SumoBot's border avoidance logic.

The technique for converting the raw sensor reading to a bit value takes advantage of a seldom-used {optional} comparison parameter with the **LOOKDOWN** function. Without the comparison parameter, **LOOKDOWN** uses equality to scan its table for a value match. By using the comparison parameter, we can test a range of values with a single table entry.

This line of code:

```
LOOKDOWN leftSense, >=[1000, 0], lineLeft
```

will put 0 into *lineLeft* if *leftSense* is greater than or equal to 1000, otherwise it will but 1 in *lineLeft* (*leftSense* is between 0 and 999). This works because the comparison parameter is >= and the first table entry [index 0] is 1000. **LOOKDOWN** will transfer the index to the output variable as soon as a match is found. If the sensor value is less than 1000 (as when on the border), the index value of 1 is moved into *lineLeft*.

We can use an **IF-THEN** coding technique to accomplish the same thing:

```
Left_Conv:
  lineLeft = 0
  IF (leftSense >= 1000) THEN Right_Conv
  lineLeft = 1

Right_Conv:
```

As you can see, using **LOOKDOWN** with the comparison parameter is the more elegant approach. You may be wondering why the value 1000 was used a the black level threshold. It was selected to allow the QTI to be affected by external light and still return an accurate reading. Extra light falling on the playing surface will

reduce the QTI output values. By using a value about one fourth the normal black reading, the program has plenty of margin for lighting variability. Now let's examine what happens with the output values.

The bit variables `lineLeft` and `lineRight` are aliased into `lineBits` -- this means a change in either will cause a change in `lineBits`. We will use this variable with the **BRANCH** command to determine which action to take based on the QTI sensor values.

The movement logic is controlled by a **BRANCH** table. The purpose of **BRANCH** is to replace several **IF-THEN** commands that would use the same control variable. So, this single line of code:


```
BRANCH lineBits, [Go_Fwd, Spin_Left, Spin_Right, About_Face]
```

replaces these four lines:

```
IF (lineBits = %00) THEN Go_Fwd
IF (lineBits = %01) THEN Spin_Left
IF (lineBits = %10) THEN Spin_Right
IF (lineBits = %11) THEN About_Face
```

and accomplishes the same objective. As with **LOOKDOWN** in the example above, **BRANCH** offers a cleaner solution to multiple **IF-THEN** commands. Keep in mind that program 3.2 doesn't actually move the SumoBot. It simply displays the action that should be taken given the QTI inputs.

Our First Operational Sumo Program

With the ability to move and to see the border on the Sumo ring, we have enough to create a working Mini-Sumo program. Add an LED to the breadboard as shown in Figures 3.3 and 3.4 then load and run program 3.3.

Figure 3.3: Start LED Schematic

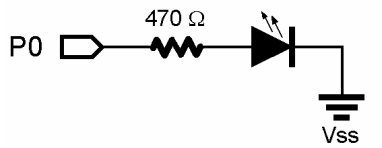
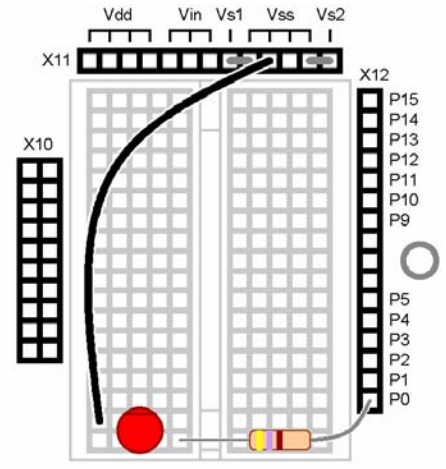


Figure 3.4: Start LED Connections on the SumoBoard




```
' -----[ Title ]-----------------------------------------------------------
' Mini-Sumo 3.3 : Simple Mini-Sumo (blind)
' {$STAMP BS2}

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

LMotor          CON     13                      ' left servo motor
RMotor          CON     12                      ' right servo motor

LLineSnsrPwr    CON     10                      ' left line sensor power
LLineSnsrIn     CON      9                      ' left line sensor input
RLineSnsrPwr    CON      7                      ' right line sensor power
RLineSnsrIn     CON      8                      ' right line sensor input

StartLED        CON      0                      ' display start delay

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

LFwdFast        CON     1000                    ' left motor forward; fast
LFwdSlow        CON      800                    ' left motor forward; slow
LStop           CON      750                    ' left motor stop
LRevSlow        CON      700                    ' left motor reverse; slow
LRevFast        CON      500                    ' left motor reverse; fast

RFwdFast        CON      500                    ' right motor forward; fast
```

```
RFwdSlow        CON      700                    ' right motor forward; slow
RStop           CON      750                    ' right motor stop
RRevSlow        CON      800                    ' right motor reverse; slow
RRevFast        CON     1000                    ' right motor reverse; fast

ClrEOL          CON     11                      ' clear to end of line (DEBUG)

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

leftSense       VAR     Word                    ' left sensor raw reading
rightSense      VAR     Word                    ' right sensor raw reading
lineBits        VAR     Nib                     ' decoded sensors value
lineLeft        VAR     lineBits.Bit1
lineRight       VAR     lineBits.Bit0

pulses          VAR     Byte                    ' counter for motor control
temp            VAR     Byte

' -----[ EEPROM Data ]-----------------------------------------------------

RunStatus       DATA    $00                     ' run status

' -----[ Initialization ]--------------------------------------------------

Run_Check:                                      ' user Reset button as On-Off
  READ RunStatus, temp                          ' read current status
  temp = ~temp                                  ' invert status
  WRITE RunStatus, temp                         ' save status for next reset
  IF (temp = 0) THEN Start_Delay                ' run now?
  END                                           '   -- no ... next time

Start_Delay:                                    ' mandatory five second delay
  HIGH StartLED                                 ' show active
  PAUSE 5000
  INPUT StartLED                                ' LED off

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

Main:
  GOSUB Read_Line_Sensors

  ' Sumo movement
  BRANCH lineBits, [Go_Fwd, Spin_Left, Spin_Right, About_Face]

Go_Fwd:
```

```
  PULSOUT LMotor, LFwdFast
  PULSOUT RMotor, RFwdFast
  GOTO Main

Spin_Left:
  FOR pulses = 1 TO 20
    PULSOUT LMotor, LRevFast
    PULSOUT RMotor, RFwdFast
    PAUSE 20
  NEXT
  GOTO Main

Spin_Right:
  FOR pulses = 1 TO 20
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  GOTO Main

About_Face:
  FOR pulses = 1 TO 10                          ' back up from edge
    PULSOUT LMotor, LRevFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  FOR pulses = 1 TO 30                          ' turn around
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  GOTO Main

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

Read_Line_Sensors:
  HIGH LLineSnsrPwr                             ' activate sensors
  HIGH RLineSnsrPwr
  HIGH LLineSnsrIn                              ' discharge caps
  HIGH RLineSnsrIn
  PAUSE 1
  RCTIME LLineSnsrIn, 1, leftSense              ' read left sensor
  RCTIME RLineSnsrIn, 1, rightSense             ' read right sensor
  LOW LLineSnsrPwr                              ' deactivate sensors
  LOW RLineSnsrPwr
```

```
' convert readings to bits
LOOKDOWN leftSense, >=[1000, 0], lineLeft     ' 0 = on black, 1 = on line
LOOKDOWN rightSense, >=[1000, 0], lineRight
RETURN
```

How It Works

This program incorporates a very handy technique created by Andy Lindsay that allows the operator to use the SumoBoard's Reset switch as an On-Off button. This is useful for controlling the SumoBot in a competition without fumbling for the actual power switch.

The code at **Run_Check** reads a byte from the BASIC Stamp's EEPROM; located at the address called **RunStatus**. When the program is first downloaded to the BASIC Stamp, the value stored in **RunStatus** will be $00 (don't run). The BASIC Stamp **READ**s this value into *temp*, inverts it (to $FF) then **WRITE**s it back to the EEPROM for the next reset cycle. Next, the value of *temp* is tested for zero. If it is zero, the program jumps to the **Start_Delay** code, otherwise the BASIC Stamp is put into low-power mode with **END** and the program halts.

The routine called **Start_Delay** is a requirement of Mini-Sumo competition rules. The rules state that the robot must not start until five seconds after the command is given by the judge. This code is very simple: it lights the LED to show that the SumoBot is preparing to start and inserts a five second **PAUSE**. At the end of the delay, the LED will be extinguished and the SumoBot will start moving.

The heart of the program begins at **Main**. The first step is to read the sensors with **Read_Line_Sensors.** With the QTI sensor data, the SumoBot is moved using the same logic developed earlier:

L	R	Action
0	0	Move forward
0	1	Spin to left
1	0	Spin to right
1	1	Back up and turn around

BRANCH takes care of routing the program to the right code based on the QTI inputs. The only thing that might not be clear is that there is no delay (20 ms) in the **Go_Fwd** routine. It isn't needed when the SumoBot is in the middle of the ring (not touching the border line) because reading the QTI sensors takes about 20 milliseconds.

Challenges

1. Experiment with the **`Spin_Left`** and **`Spin_Right`** routines so that the SumoBot generally moves toward the center after touching the border.

Experiment #4: SumoBot Sensors – IR Object Dection

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

What's Infrared?

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

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

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.

4

Infrared Headlights

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

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

The FREQOUT Trick

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

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

Figure 4.1 shows (a) the signal sent by the command **FREQOUT Pin, Period, 27036**. Tuned electronic receivers, such as the IR detectors we'll be using, can detect components of this signal that are called harmonics. The **FREQOUT** signal's two dominant low frequency harmonics are shown in Figures 4.1 (b) and (c). Figure 4.1 (b) shows the fundamental harmonic, and Figure 4.1 (c) shows the third harmonic. These harmonics are actually components of the unfiltered **FREQOUT** pulses shown in Figure 4.1 (a). The third harmonic shown in Figure 6.1 (c) can be controlled directly by entering commands such as **FREQOUT Pin, Period, 38500** (instead of **27036**) for 38.5 kHz, or **FREQOUT Pin, Period, 40000** for 40 kHz, etc.

Installing and Testing the IR Emitters/Detectors

The SumoBoard is specially designed to accommodate two IR emitter/detector pairs. Before we install them, they parts need to be adjusted so that they don't become damaged or misaligned during competition.

Figure 4.2 shows the IR emitter and detectors. To modify the emitters, carefully bend the leads 90 degrees at the point indicated in the diagram. You may also want to trim the leads a bit, so that the emitter seats into the socket just up to the shrink tube (trim the leads as shown to designate the positive lead [long]). When installed, the emitters will look a bit like a crab's eyes. Don't be tempted to bend the leads at the end of the shrink tube as IR light leaking from the end of the tube can affect the detector, and a bend in this position causes the IR emitter to protrude far enough forward to become misaligned by the opponent during the match.

Modify the detectors by trimming the leads to about 3/8" inches. This will cause the detectors to sit lower and more firmly in the sockets, reducing the chance of misalignment during competition.

Figure 4.1: FREQOUT Example Properties

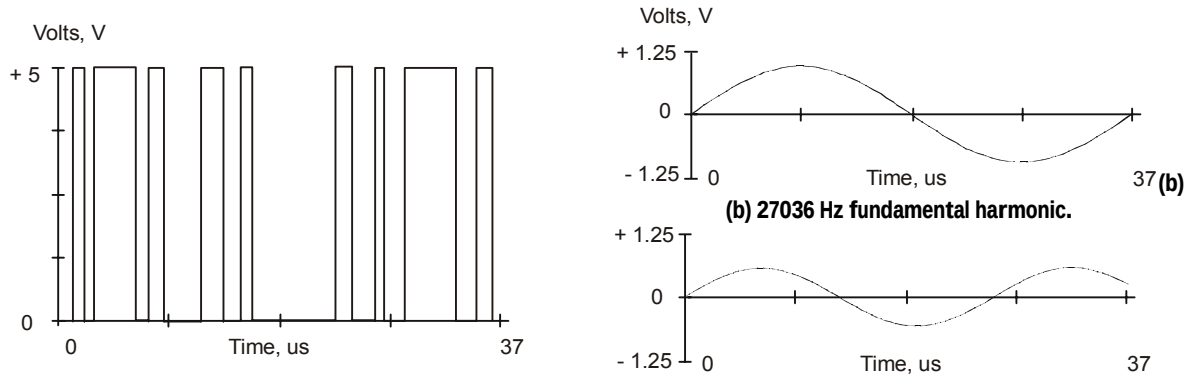


Figure 4.1: (a) Unfiltered freqout pulses sent by `FREQOUT pin, period, 27036`

(b) 27036 Hz fundamental harmonic.

(c) 38500 Hz third harmonic.

Figure 4.2: Emitter/Detector[3] Adjustments

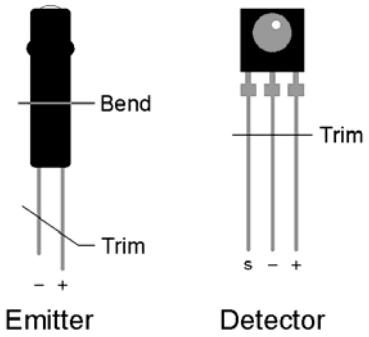


Figure 4.3 is the schematic for the SumoBot's IR object sensing. Build this circuit on your SumoBoard. Note that the 220 ohm resistors are already built into the SumoBoard PCB; just plug in the IR emitters and your

[3] Spare emitters (#350-00014) and detectors (#350-00013) can be ordered Parallax at www.parallaxinc.com

SumoBot will be ready. When aligning the IR emitter "headlights" it's a good idea to angle them slightly outward to give the SumoBot a wider field of vision.

Figure 4.3: SumoBot IR Object Detection Schematic

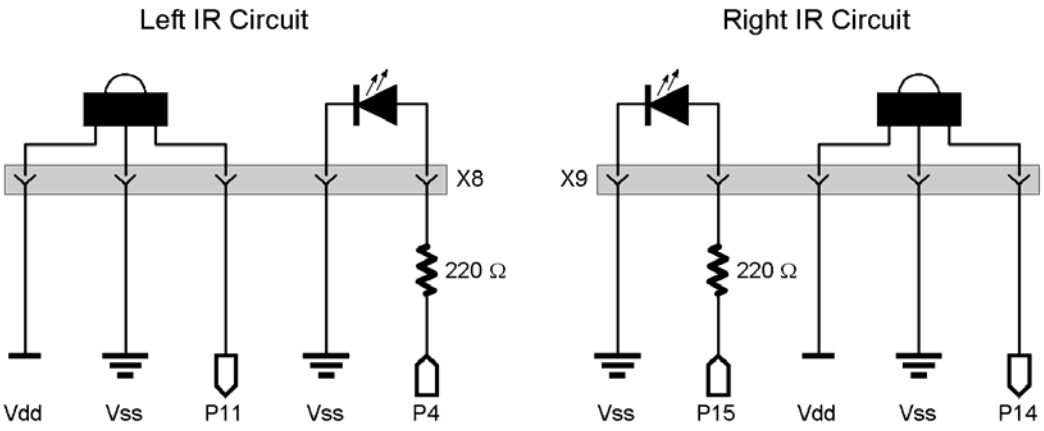
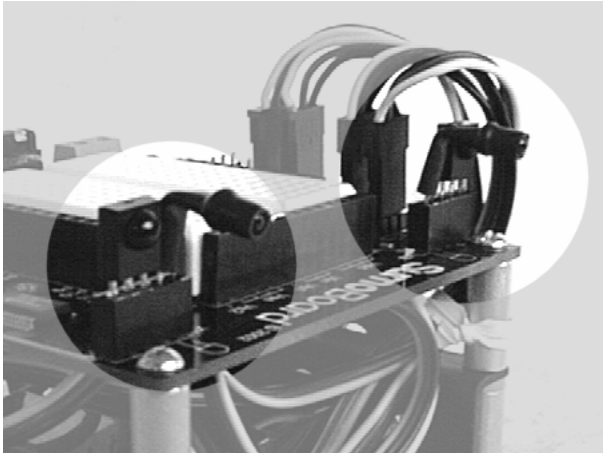


Figure 4.4: SumoBot IR Object Detection Components Installed



Testing the IR Pairs

The key to making each IR pair work is to send 1 millisecond of unfiltered 38.5 kHz **FREQOUT** harmonic followed immediately by testing the signal sent by the IR detector and saving its output value. The IR detector's normal output state when it sees no IR signal is high. When the IR detector sees the 38500 Hz harmonic sent by the IR LED, it's output will drop from high to low. Of course, if the IR does not reflect off an object, the IR detector's output simply stays high. Program 4.1 shows an example of this method of reading the detectors

```
' -----[ Title ]-----------------------------------------------------------------
' Mini-Sumo 4.1 : IR Sensor Test
' {$STAMP BS2}

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

LfIrOut         CON     4                       ' left IR LED output
LfIrIn          VAR     In11                    ' left IR sensor input
RtIrOut         CON     15                      ' right IR LED output
RtIrIn          VAR     In14                    ' right IR sensor input

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

irBits          VAR     Nib                     ' storage for IR target data
irLeft          VAR     irBits.Bit1
irRight         VAR     irBits.Bit0

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

Read_Left:
  FREQOUT LfIrOut, 1, 38500                     ' modulate IR LED
  irLeft = ~LfIrIn                              ' read input (1 = target)

Read_Right:
  FREQOUT RtIrOut, 1, 38500                     ' modulate IR LED
  irRight = ~RtIrIn                             ' read input (1 = target)

Display:
  DEBUG Home
  DEBUG "L  R", CR
  DEBUG "----",CR
  DEBUG BIN1 irLeft, "  ", BIN1 irRight

  PAUSE 20
  GOTO Read_Left
```

How It Works

Two bit variables are declared to store the value of each IR detector output. The first **FREQOUT** command in the **Read_Left** routine is different. The command **FREQOUT LfIrOut, 1, 38500** sends the on-off pattern shown in Figure 4.1 via left IR LED circuit by causing it to flash on and off rapidly. The harmonic contained in this signal either bounces off an object, or not. If it bounces off an object and is seen by the IR detector, the IR detector sends a low signal to I/O pin **LfIrIn**. Otherwise, the IR detector sends a high signal to **LfIrIn**. The invesion operator (~) is used so that a "hit" (reflection from the opponent) is indicated by "1" and a "miss" is indicated by a "0." So long as the next command after the **FREQOUT** command is the one testing the state of the IR detector's output, it can be saved as a variable value in RAM. The statement **irLeft = ~LfIrIn** checks **LfIrIn**, and saves the value ("1" for hit or "0" for miss) in the *irLeft* bit variable. This process is repeated for the other IR pair, and the IR detector's output is saved in the *irRight* variable. The **DEBUG** command then displays the values in the debug window.

SumoBot Motion Control

The next task is to link the SumoBot's ability to "see" with the motors so that an object – the opponent – can be tracked. Load and run progrma 4.2 to see a demonstration of linking IR object control to the SumoBot's motors.

```
' -----[ Title ]-----------------------------------------------------------
' Mini-Sumo 4.2 : IR Scan
' {$STAMP BS2}

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

LMotor          CON     13                      ' left servo motor
RMotor          CON     12                      ' right servo motor

LfIrOut         CON     4                       ' left IR LED output
LfIrIn          VAR     In11                    ' left IR sensor input
RtIrOut         CON     15                      ' right IR LED output
RtIrIn          VAR     In14                    ' right IR sensor input

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

LFwdFast        CON     1000                    ' left motor forward; fast
LFwdSlow        CON      800                    ' left motor forward; slow
LStop           CON      750                    ' left motor stop
LRevSlow        CON      700                    ' left motor reverse; slow
```

```
LRevFast        CON      500                    ' left motor reverse; fast

RFwdFast        CON      500                    ' right motor forward; fast
RFwdSlow        CON      700                    ' right motor forward; slow
RStop           CON      750                    ' right motor stop
RRevSlow        CON      800                    ' right motor reverse; slow
RRevFast        CON     1000                    ' right motor reverse; fast

ClrEOL          CON     11                      ' clear to end of line (DEBUG)

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

irBits          VAR     Nib                     ' storage for IR target data
irLeft          VAR     irBits.Bit1
irRight         VAR     irBits.Bit0
lastIr          VAR     Nib                     ' info from last reading

pulses          VAR     Byte                    ' counter for motor control

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

Main:
  GOSUB Read_IR_Sensors
  BRANCH irBits, [Scan, Follow_Right, Follow_Left, Hold]

Scan:
  BRANCH lastIr, [Move_Fwd, Scan_Right, Scan_Left, Move_Fwd]

Move_Fwd:
  DEBUG Home, "Forward", ClrEOL
  GOTO Main

Scan_Right:                                     ' spin right, slow
  DEBUG Home, "Scan Right", ClrEOL
  PULSOUT LMotor, LFwdSlow
  PULSOUT RMotor, RRevSlow
  PAUSE 20
  GOTO Main

Scan_Left:                                      ' spin left, slow
  DEBUG Home, "Scan Left", ClrEOL
  PULSOUT LMotor, LRevSlow
  PULSOUT RMotor, RFwdSlow
  PAUSE 20
  GOTO Main
```

```
Follow_Right:                                   ' spin right, fast
  DEBUG Home, "Follow Right", ClrEOL
  PULSOUT LMotor, LFwdFast
  PULSOUT RMotor, RRevFast
  PAUSE 20
  lastIr = irBits                               ' save last direction found
  GOTO Main

Follow_Left:                                    ' spin left, fast
  DEBUG Home, "Follow Left", ClrEOL
  PULSOUT LMotor, LRevFast
  PULSOUT RMotor, RFwdFast
  PAUSE 20
  lastIr = irBits
  GOTO Main

Hold:                                           ' on target
  DEBUG Home, "On Target", ClrEOL
  FOR pulses = 1 TO 3
    PULSOUT LMotor, LStop
    PULSOUT RMotor, RStop
    PAUSE 20
  NEXT
  lastIr = irBits
  GOTO Main

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

Read_IR_Sensors:
  FREQOUT LfIrOut, 1, 38500                     ' modulate left IR LED
  irLeft = ~LfIrIn                              ' read input (1 = target)
  FREQOUT RtIrOut, 1, 38500                     ' modulate right IR LED
  irRight = ~RtIrIn                             ' read input (1 = target)
  RETURN
```

How It Works

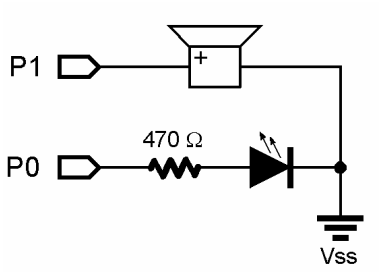
This program is functionally similar to the line detection program. The IR sensors are scanned and, due to variable aliasing, the state of both sensors is held in the Nibble variable *irBits*. If no target is detected the SumoBot will go into scanning mode. The code at Scan will look for the opponent in the last known direction (held in the variable *lastIr*). When a target is detected, the SumoBot will move quickly in the target direction. The direction is stored in the event the target is lost so that the SumoBot will scan in the last know target direction.

If you run the program with the power switch in position 1, the DEBUG window will display the program's logic based on a target placed in front of the SumoBot. If you run the program with the power switch in position 2, the SumoBot will rotate in the direction of the target or scan in the suspect direction.

Experiment #5: SumoBot Competition Code

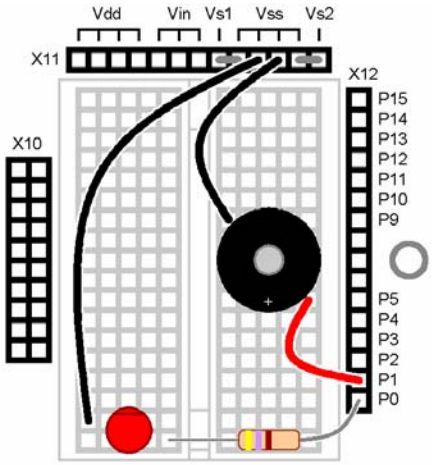
Time to put it all together and compete. The program in this chapter brings all the SumoBot's systems together and adds a bit of intelligent control. Add a piezo speaker as shown in Figures 5.1 and 5.2, then load and run program 5.1.

Figure 5.1 Start LED and Piezo Speaker Schematic



5

Figure 5.2: Start LED and Piezo Speaker Connections on the SumoBoard



```
' -----[ Title ]-----------------------------------------------------------------
' Mini-Sumo 5.1 : Basic Competition Program
' {$STAMP BS2}

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

LMotor          CON     13                      ' left servo motor
RMotor          CON     12                      ' right servo motor

LLineSnsPwr     CON     10                      ' left line sensor power
LLineSnsIn      CON      9                      ' left line sensor input
RLineSnsPwr     CON      7                      ' right line sensor power
RLineSnsIn      CON      8                      ' right line sensor input

LfIrOut         CON      4                      ' left IR LED output
LfIrIn          VAR     In11                    ' left IR sensor input
RtIrOut         CON     15                      ' right IR LED output
RtIrIn          VAR     In14                    ' right IR sensor input

Speaker         CON      1                      ' piezo speaker
StartLED        CON      0                      ' display start delay

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

LFwdFast        CON     1000                    ' left motor forward; fast
LFwdSlow        CON      800                    ' left motor forward; slow
LStop           CON      750                    ' left motor stop
LRevSlow        CON      700                    ' left motor reverse; slow
LRevFast        CON      500                    ' left motor reverse; fast

RFwdFast        CON      500                    ' right motor forward; fast
RFwdSlow        CON      700                    ' right motor forward; slow
RStop           CON      750                    ' right motor stop
RRevSlow        CON      800                    ' right motor reverse; slow
RRevFast        CON     1000                    ' right motor reverse; fast

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

leftSense       VAR     Word                    ' left sensor raw reading
rightSense      VAR     Word                    ' right sensor raw reading
blackThresh     VAR     Word                    ' QTI black threshold setting
lineBits        VAR     Nib                     ' decoded sensors value
lineLeft        VAR     lineBits.Bit1
lineRight       VAR     lineBits.Bit0
```

```
irBits          VAR     Nib                     ' storage for IR target data
irLeft          VAR     irBits.Bit1
irRight         VAR     irBits.Bit0
lastIr          VAR     Nib                     ' info from last reading

pulses          VAR     Byte                    ' counter for motor control
temp            VAR     Byte

' -----[ EEPROM Data ]-----------------------------------------------------------

RunStatus       DATA    $00                     ' run status

' -----[ Initialization ]--------------------------------------------------------

Run_Check:                                      ' user Reset button as On-Off
  READ RunStatus, temp                          ' read current status
  temp = ~temp                                  ' invert status
  WRITE RunStatus, temp                         ' save status for next reset
  IF (temp = 0) THEN Set_Threshold              ' run now?
  END                                           '   -- no ... next time

' Sets black threshold to 1/4 the average of the two sensor readings.
' SumoBot must be placed over black playing surface before this code runs.

Set_Threshold:                                  ' set QTI black threshold
  GOSUB Read_Line_Sensors
  blackThresh = (leftSense / 8) + (rightSense / 8)

Start_Delay:                                    ' mandatory five second delay
  FOR temp = 1 TO 5
    HIGH StartLED                               ' show active
    PAUSE 900
    INPUT StartLED                              ' blink each second
    FREQOUT Speaker, 100, 2500, 3000            ' beep each second
  NEXT

  GOTO Lunge                                    ' start aggressive!

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

Main:
  GOSUB Read_Line_Sensors

  ' If not on the Shikiri line (border), continue to look for opponent,
```

```
  ' otherwise, spin back toward center and resume search

  BRANCH lineBits, [Search_For_Opponent, Spin_Left, Spin_Right, About_Face]

' --[ Border Avoidance ]--

Spin_Left:                                        ' right sensor was active
  FOR pulses = 1 TO 20
    PULSOUT LMotor, LRevFast
    PULSOUT RMotor, RFwdFast
    PAUSE 20
  NEXT
  GOTO Lunge

Spin_Right:                                       ' left sensor was active
  FOR pulses = 1 TO 20
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  GOTO Lunge

About_Face:                                       ' both sensors on Shikiri line
  FOR pulses = 1 TO 10                            ' back up from edge
    PULSOUT LMotor, LRevFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  FOR pulses = 1 TO 30                            ' turn around
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RRevFast
    PAUSE 20
  NEXT
  GOTO Lunge

' --[ IR Processing ]--

Search_For_Opponent:
  GOSUB Read_IR_Sensors

  ' If opponent is not in view, scan last known direction.  Turn toward
  ' opponent if seen by one "eye" -- if both, lunge forward

  BRANCH irBits, [Scan, Follow_Right, Follow_Left, Lunge]
```

```
Scan:
  BRANCH lastIr, [Move_Fwd, Scan_Right, Scan_Left, Move_Fwd]

Move_Fwd:
  GOSUB Creep_Forward
  GOTO Main

Scan_Right:                                       ' spin right, slow
  FOR pulses = 1 TO 5
    PULSOUT LMotor, LFwdSlow
    PULSOUT RMotor, RRevSlow
    PAUSE 20
  NEXT
  GOSUB Creep_Forward                             ' keep moving
  GOTO Main

Scan_Left:                                        ' spin left, slow
  FOR pulses = 1 TO 5
    PULSOUT LMotor, LRevSlow
    PULSOUT RMotor, RFwdSlow
    PAUSE 20
  NEXT
  GOSUB Creep_Forward
  GOTO Main

Follow_Right:                                     ' spin right, fast
  PULSOUT LMotor, LFwdFast
  PULSOUT RMotor, RRevSlow
  lastIR = irBits                                 ' save last direction found
  GOTO Main

Follow_Left:                                      ' spin left, fast
  PULSOUT LMotor, LRevSlow
  PULSOUT RMotor, RFwdFast
  lastIR = irBits
  GOTO Main

Lunge:                                            ' locked on -- go get him!
  FOR pulses = 1 TO 10
    PULSOUT LMotor, LFwdFast
    PULSOUT RMotor, RFwdFast
    GOSUB Read_Line_Sensors
    IF (lineBits = %11) THEN Match_Over           ' in sight and we're on the line
  NEXT
  GOTO Main
```

```
' If SumoBot can see the opponent with both "eyes" and both QTIs are
' detecting the border, we must have pushed the opponent out.

Match_Over:
  FOR pulses = 1 TO 10                          ' stop motors
    PULSOUT LMotor, LStop
    PULSOUT RMotor, RStop
    PAUSE 20
  NEXT
  INPUT LMotor
  INPUT RMotor

  FOR temp = 1 TO 10                            ' make some noise
    HIGH StartLED
    FREQOUT Speaker, 100, 2500, 3000            ' beep
    INPUT StartLED                              ' blink LED
    PAUSE 100
  NEXT

  Dirs = $0000                                  ' disable all outputs
  GOTO Run_Check                                ' reset for next round

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

Read_Line_Sensors:
  HIGH LLineSnsPwr                              ' activate sensors
  HIGH RLineSnsPwr
  HIGH LLineSnsIn                               ' discharge QTI caps
  HIGH RLineSnsIn
  PAUSE 1
  RCTIME LLineSnsIn, 1, leftSense               ' read left sensor
  RCTIME RLineSnsIn, 1, rightSense              ' read right sensor
  LOW LLineSnsPwr                               ' deactivate sensors
  LOW RLineSnsPwr

  ' convert readings to bits
  lineBits = %00
  LOOKDOWN leftSense, >=[blackThresh, 0], lineLeft
  LOOKDOWN rightSense, >=[blackThresh, 0], lineRight
  RETURN

Read_IR_Sensors:
  FREQOUT LfIrOut, 1, 38500                     ' modulate left IR LED
  irLeft = ~LfIrIn                              ' read input (1 = target)
```

```
  FREQOUT RtIrOut, 1, 38500                    ' modulate right IR LED
  irRight = ~RtIrIn                            ' read input (1 = target)
  RETURN

Creep_Forward:
  FOR pulses = 1 TO 10
    PULSOUT LMotor, LFwdSlow
    PULSOUT RMotor, RFwdSlow
    PAUSE 20
  NEXT
  RETURN
```

How It Works

After the **Run_Check** routine developed in the last experiment, the SumoBot reads the QTI sensors and calculates a suitable threshold level for the playing field. The net effect is to calculate the average sensor reading ((left + right) / 2) and divide this value by four. The *blackThresh* variable holds this level and is recalculated for each match. This way the SumoBot can deal with variable ambient lighting conditions.

The **Start_Delay** function serves the same purpose; it's just a bit fancier. In this case we've added a piezo speaker. At the start, the LED illuminates and then blinks on each second mark. The LED blink coincides with a beep from the piezo speaker.

Since many matches start with the Mini-Sumo robots facing each other, the subroutine called **Lunge** is called to get the SumoBot moving quickly. This routine drives the SumoBot forward as fast as possible. It will also be used later when the opponent is located. If your club rules specify that the SumoBot will not face the opponent from the start, you may wish to substitute a rotation command to get the SumoBot oriented toward the opponent as quickly as possible.

The core of the program is at **Main**. The code starts by reading the QTI sensors (now using the calibrated black level for the Sumo ring). If the QTI sensor indicate that the SumoBot is safely on the playing surface, a **BRANCH** command will send the code **Search_For_Opponent**. If the SumoBot is touching the Tawara line (border), the edge avoidance logic previously developed will be used. When the border is touched, the *lastIr* variable is cleared so the SumoBot will move toward the center after turning in from the border.

The **Search_For_Opponent** routine employs a bit of logic to minimize random searching. This code calls the **Read_IR_Sensors** routine and will **BRANCH** accordingly, based on the logic developed in program 4.2. If the opponent is not currently in sight, the SumoBot will scan in the direction last known. Early on, when the opponent has not been spotted, the SumoBot will creep forward in order to cut down the range between itself and the opponent.

If **`Read_IR_Sensors`** returns a hit, the SumoBot will move quickly in the direction of the opponent. The hit direction is recorded in case the opponent escapes and the SumoBot is forced back into Scan mode. When the opponent is locked on (`irBits = %11`), the SumoBot will go into "Lunge" mode to get to the opponent quickly and move it out of the ring.

The best outcome is when the SumoBot can "see" the opponent with both "eyes" and both QTI sensors detect the Tawara border. Under these conditions the assumption is that the opponent has been successfully pushed from the ring and a Yuko point is to be awarded. Many matches will be won when the SumoBot removes the competitor from the ring and an angle; the competitor will be out but the SumoBot will continue to push, or may go into edge-avoidance and search mode. In this case the Judge will award the Yuko point and the SumoBot can be stopped by pressing the Reset button.

Final Competition Notes

The SumoBot – like any robot – will perform best with fresh batteries. The first sign of weak batteries will be a degradation in opponent detection range. Make sure that you take plenty of fresh batteries to each contest you participate in.

When fully assembled and loaded with batteries, the SumoBot will weigh less than the 500 gram (17.6 oz) limit imposed by standard Mini-Sumo rules. This could be a disadvantage against a heavier robot. There are many ways to add weight to the SumoBot chassis; one of the easiest is with Prather Products stick on weight. These stick-on weights are particularly convenient to add weight to the back of the scoop. This helps keep the scoop on the playing surface. Prather Products weights are available in hobby shops.

Appendix A: Parts Listing and Sources

All parts used in the SumoBot kit are available individually from the Parallax Component Shop[4]. If you can't readily find the component you are looking for in the Component Shop enter the name of it in the on-line search box using the stock code.

Parallax Part #	Description	Qty/Kit
Electronic Components		
550-27401	QTI line sensor	2
350-00017	Infrared emitter	2
350-00014	Infrared detector	2
350-00006	Red LED	1
350-00001	Green LED	1
150-04710	470 ohm resistor, 1/4 watt	2
900-00001	Piezo speaker	1
800-00016	Jumper wires (bag of 10)	1
550-27400	SumoBot Printed Circuit Board with BASIC Stamp 2	1
753-00001	Battery Pack with Tinned Wires	1
805-00001	3-pin extension cable (for QTI)	2
900-00008	Parallax Continuous-Rotation Servo	2
Mechanical Parts		
720-27403	SumoBot chassis	1
720-27404	SumoBot scoop	1
721-00001	Plastic wheel	2
721-00002	Rubber tire (for plastic wheel)	4
Hardware		
713-00001	5/8" round aluminum standoff	4
713-00002	1-1/4" round aluminum standoff	2
700-00028	1/4" 4/40 machine screw – panhead	4
700-00016	3/8" 4/40 machine screw – flathead	2
700-00002	3/8" 4/40 machine screw – panhead	12
710-00002	1" 4/40 machine screw - panhead	2
700-00003	4/40 machine nut	12
700-00015	Nylon washer	2

[4] www.parallaxinc.com → Component Shop

Miscellaneous		
122-27400	SumoBot Manual	1
800-00003	Serial (programming) cable	1
27000	Parallax CD	1
700-00064	Parallax Screwdriver	1

Appendix B: Mini-Sumo Competition Rules

Reprinted with permission from Bill Harrison, Sine Robotics.

Section 1. Definition of a Match

Article 1 **Definition**

The match shall be contested by two teams (At the event, one team consists of one robot with two team members, one of which is a leader. Other team members must watch from the audience), according to these Rules for Sumo matches (hereafter called "these Rules"), with each team's robot made by each team (either a remote-controlled model or a standalone model) competing to get the effective points (hereafter called "Yuko"), within the perimeter of the defined Sumo Ring. The judges will decide which team wins. A single person can also compete with a Robot Sumo, with the same rules that apply to teams.

Section 2. Requirements for the Ring Area

Article 2 **Definition of Ring Area**

The Ring Area means the Sumo Ring and the space outside the Ring. Anywhere outside this Ring Area is called Outer Area.

Article 3 **Sumo Ring**

1. The Ring shall be in circular shape with its height being 2.5 cm and its diameter 77 cm (including the outside of the line that divides the inside of the Ring from its outside). The Ring shall be of black hard rubber (made by Toyo Linoleum; long vinyl sheet NC #R289, or it's equivalent) adhered on top of Ring.

2. Shikiri lines (where robots stand at the beginning of the match) are the two parallel lines with 10 cm distance between the lines, drawn in the center of the Ring. The Shikiri lines are painted in brown (or equivalent for reflection of IR light), 1 cm wide and 10 cm long.

3. The Ring shall be marked by a white circular line of 2.5 cm thickness. The Ring is within the outside of this circular line.

Article 4 **Space**

There should be the space of more than 50 cm wide outside the outer side of the Ring. This space can be of any color except white, and can be of any materials or shape, as long as the basic concept of these rules are observed. This area, with the ring in the middle, is to be called the: "Ring Area". If there are markings or part of the ring platform outside these dimentions, this area will also be considered in the Ring Area.

Section 3. Requirements for Robots

Article 5 **Specifications**

1. A robot must be in such a size that it can be put in a square tube of 10 cm wide and 10 cm deep. A robot can be of any height. A robot must not be in such a design that its body will be physically separated into pieces when a match starts. The robot with such a design shall lose the match. The design to stretch a robot's body or its parts shall be allowed, but must remain a single centralized robot. Screws or nuts or other robot parts, with a mass of less than 5 grams total, falling off from a robot's body shall not cause the loss of match.

2. The mass of a robot must be under 500 grams including the attachments and parts, but excluding the weight of a proportional system (the transmitter or control box held by the operator, hereafter called "Prop") for remote-controlled models.

3. The radio frequencies for radio-controlled robots must be either 27 MHZ (1-6 bands) or 40 MHZ (61, 63, 65, 67, and 69 bands). 40 MHZ (71-83 bands) cannot be used.

4. Only one Prop can be used for one robot. Radio control Prop must be one of Futaba's, JR's, Sanwa's, or Kondo Kagaku's.

5. For stand-alone robots, any control mechanisms can be employed.

6. Stand-alone models must be so designed that a robot starts operating a minimum of five seconds after a start switch is pressed (or any method that invokes the operation of a robot).

7. Microcomputers in a robot can be of any manufacturers and any memory sizes can be chosen.

Article 6 **Restrictions on Robot Design**

1. Jamming devices, such as an IR LED intended to saturate the opponents IR sensor, are not allowed. Do not disturb your opponent's radio-control by putting into a robot's body such devices as a jamming device.

2. Do not use parts that could break or damage the Ring. Do not use parts that are intended to damage the opponents robot or it's operator. Normal pushes and bangs are not considered intent to damage.

3. Do not put into a robot's body devices that can store liquid, powder, or air, in which are thrown at the opponent.

4. Do not use any inflaming devices.

5. Do not use devices that throw things at your opponent.

6. Do not stick a robot down onto the Ring, using sucking devices or glue, or use any type of sticky tires (such as double sticky foam tape) or any device to assist in adding more down force (such as a vacuum device).

Section 4. Match Principles

Article 7 **Match Principles**

1. One match shall consist of 3 rounds, within a total time of 3 minutes, unless extended by the Judges.

2. The team who wins two rounds or receives two "Yuko" points first, within the time limit, shall win the match. A team recieves a Yuko point when they win a round. If the time limit is reached before one team can get two Yuko points, and one of the teams has received one Yuko point, the team with one Yuko point shall win.

3. When the match is not won by either team within the time limit, the extended match shall be fought during which the team who receives the first Yuko point shall win.

However, the winner/loser of the match may be decided by judges or by means of lots, or there can be a rematch.

4. One Yuko point shall be given to the winner when the judges' decision was called for or lots were employed.

Section 5: Match Procedure

Article 8 **Start**

With the chief judge's instructions, the two teams bow in the Outer Ring (For example, stand facing each other, outside the ring platform or "ring area", with ring between), go up to the Ring, and place a robot on or behind the Shikiri line or the imaginary extended Shikiri line. (A robot or a part of a robot may not be placed beyond the front edge of the Shikiri line toward the opponent.). A match starts with the following rules:

1. For remote-controlled robots, start operating a Prop when the chief judge announces the start of a round.

2. For stand-alone robots, be ready to press a start switch. Press the switch when the chief judge announces the start of the round. After 5 seconds, the robot is allowed to start operating, before which players must clear out of the Ring Area.

Article 9 **Stop & Resume**

The match stops and resumes when a judge announces so.

Article 10 **End of Match**

The match ends when the judge calls the winner. Both contestants bow after removing their robots.

Section 6: Time of Match

Article 11 **Time of Match**

One Match will be contested for a total of 3 minutes, starting and ending by the chief judge's announcements. For stand-alone robots, the clock shall start ticking 5 seconds after the start is announced.

Article 12 An extended match shall be for 3 minutes, if called by the Judge.

Article 13 The following are not included in the time of the Match:

1. The time elapsed after the chief judge announces Yuko and before the match resumes. 30 seconds shall be the standard before the match resumes.

2. The time elapsed after a judge announces to stop the match and before the match resumes.

Section 7: Yuko

Article 14 **Yuko**

One Yuko point shall be given when:

1. You have legally forced the body of your opponent's robot to touch the space outside the Ring, which includes the side of the ring its self.

2. A Yuko point is also given in the following cases:
   2.1. Your opponent's robot has touched the space outside the Ring, on its own.
   2.2. Either of the above takes place at the same time that the End of the Match is announced.

3. When a robot has fallen on the Ring or in similar conditions, Yuko will not be counted and the match continues.

4. When judges' decision is called for to decide the winner, the following points will be taken into considerations:
   4.1. Technical merits in movement and operation of a robot

4.2. Penalty points during the match
4.3. Attitude of the players during the match

5. The match shall be stopped and a rematch shall start when:
   5.1. Both robots are in clinch and stop movements for 5 seconds, or move in the same orbit for 5 seconds, with no progress being made. If it is not clear if progress is being made or not, the Judge can extend the time limit for a clinch or orbiting robots up to 30 seconds.
   5.2. Both robots move, without making progress, or stop (at the exact same time) and stay stopped for 5 seconds without touching each other. However, if one robot stops it's movement first, after 5 seconds, he shall be considered not having the will to fight, and the opponent shall receive a Yuko, even if the opponent also stops. If both robots are moving and it isn't clear if progress is being made or not, the Judge can extend the time limit up to 30 seconds.
   5.3. If both robots touch the outside of the ring at about the same time, and it can not be determined which touched first, a rematch is called.

Section 8: Violations

Article 15 Violations

If the players perform the deeds as described in Articles 6, 16 and 17, the players shall be declared as violating the rules.

Article 16 The player utters insulting words to the opponent or to the judges or puts voice devices in a robot to utter insulting words or writes insulting words on the body of a robot, or any insulting action.

Article 17 A Player

1. Enters into the Ring during the match, except when the player does so to bring the robot out of the Ring upon the chief judge's announcement of Yuko or stopping the match. To enter into the Ring means:
   1.1. A part of the player's body is in the Ring, or
   1.2. A player puts any mechanical kits into the Ring to support his/her body.
2. Performs the following deeds:
   2.1. Demand to stop the match without appropriate reasons.

2.2. Take more than 30 seconds before resuming the match, unless the Judge announces a time extension..
2.3. Start operating the robot before the chief judge announces the start of the match (for remote-controlled robots).
2.4. Start operating the robot within 5 seconds after the chief judge announces the start of the match (for stand-alone robots).
2.5. Do or say that which should disgrace the fairness of the match.

Section 9: Penalties

Article 18 **Penalties**

Those who violate the rules with the deeds described in Articles 6 and 16 shall lose the match. The judge shall give two Yuko points to the opponent and order the violator to clear out. The violator is not honored with any rights.

Article 19 Each occasion of the violations described in Article 17 shall be accumulated. Two of these violations shall give one Yuko to the opponent.

Article 20 The violations described in Article 17 shall be accumulated throughout one match.

Section 10: Injuries and Accidents During the Match

Article 21 **Request to Stop the Match**

A player can request to stop the game when he/she is injured or his/her robot had an accident and the game cannot continue.

Article 22 **Unable to Continue the Match**

When the game cannot continue due to player's injury or robot's accident, the player who is the cause of such injury or accident loses the match. When it is not clear which team is such a cause, the player who cannot continue the game, or who requests to stop the game, shall be declared as the loser.

Article 23 **Time Required to Handle Injury/Accident**

Whether the game should continue in case of injury or accident shall be decided by the judges and the Committee members. The decision process shall take no longer than five minutes.

Article 24 **Yuko Given to the Player Who Cannot Continue**

The winner decided based on Article 22 shall gain two Yuko points. The loser who already gained one Yuko point is recorded as such. When the situation under Article 22 takes place during an extended match, the winner shall gain one Yuko point.

Section 11: Declaring Objections

Article 25 **Declaring Objections**

No objections shall be declared against the judges' decisions.

Article 26 The lead person of a team can present objections to the Committee, before the match is over, if there are any doubts in exercising these rules. If there is no Committee member present, the objection can be presented to the Judge, before the match is over.

Section 12: Requirements for Identifications for Robots

Article 27 **Identifications for Robots**

Some type of name or number, to identify the robot (as registered in the contest) must be easily readable on the robot's body, while the robot is in competition.

Section 13: Miscellaneous

Article 28 **Flexibility of Rules**

As long as the concept and fundamentals of the rules are observed, the rules shall be so flexible that they will be able to encompass the changes in the number of players and of the contents of matches.

Article 29 **Change in Rules**

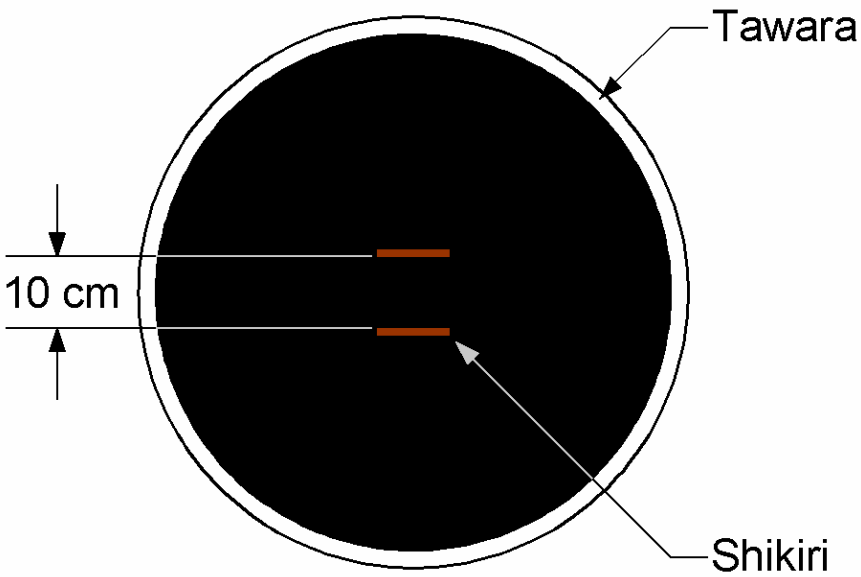
Any changes to or obsolescence of these rules shall be decided by the General Committee Meeting based on the Sumo Match Committee Rules.

Appendix C: Mini-Sumo Ring

If you're handy with tools, you can build your own Mini-Sumo ring. Many home improvement centers carry precut circles (wood, MDF, Melamine) that are very close to the official dimension and can be used to create a suitable practice ring.

Mini-Sumo Ring Specifications:

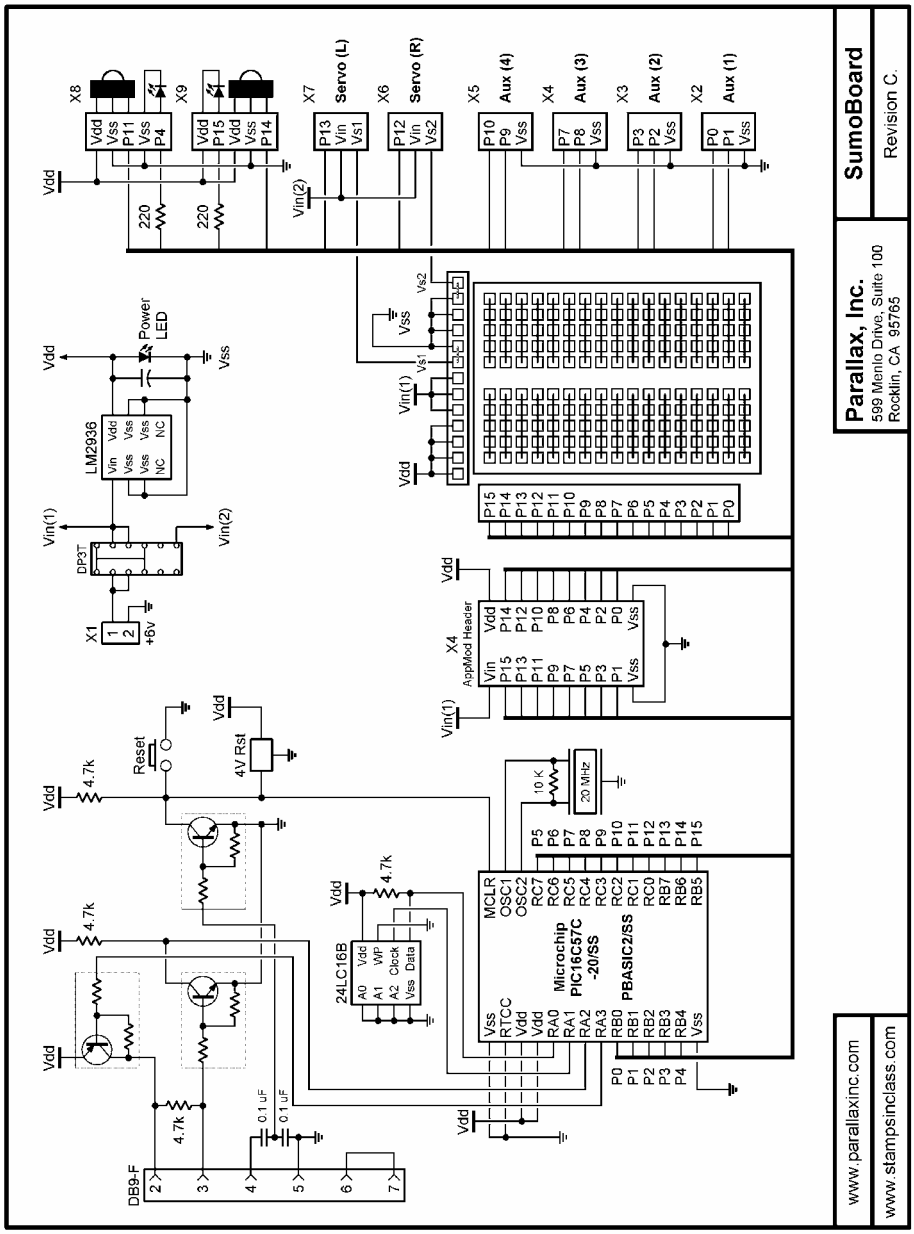
- Diameter — 77 cm / 30.3 in.
- Height — 2.5 cm / 1 in.
- Surface — Hard Rubber
- Colors
  - Ring — Black
  - Shikiri (start line) — Brown[5] (10 cm x 1 cm / 3.9 in. x 0.39 in.)
  - Tawara (border) — White (2.5 cm / 1 in.)



[5] Some clubs specify that the Shikiri (start) lines shall have a reflective value of no greater than 20% of the Tawara (border)

Appendix D: SumoBoard Schematic

(See next page)


SumoBoard
Revision C.
Parallax, Inc.
599 Menlo Drive, Suite 100
Rocklin, CA 95765
www.parallaxinc.com
www.stampsinclass.com
Servo (L)
Servo (R)
Aux (4)
Aux (3)
Aux (2)
Aux (1)
Power LED
LM2936
DP3T
+6v
AppMod Header
Reset
4V Rst
24LC16B
Microchip PIC16C57C -20/SS
PBASIC2/SS
20 MHz
10 K
4.7k
220
0.1 uF
DB9-F