Lab 6

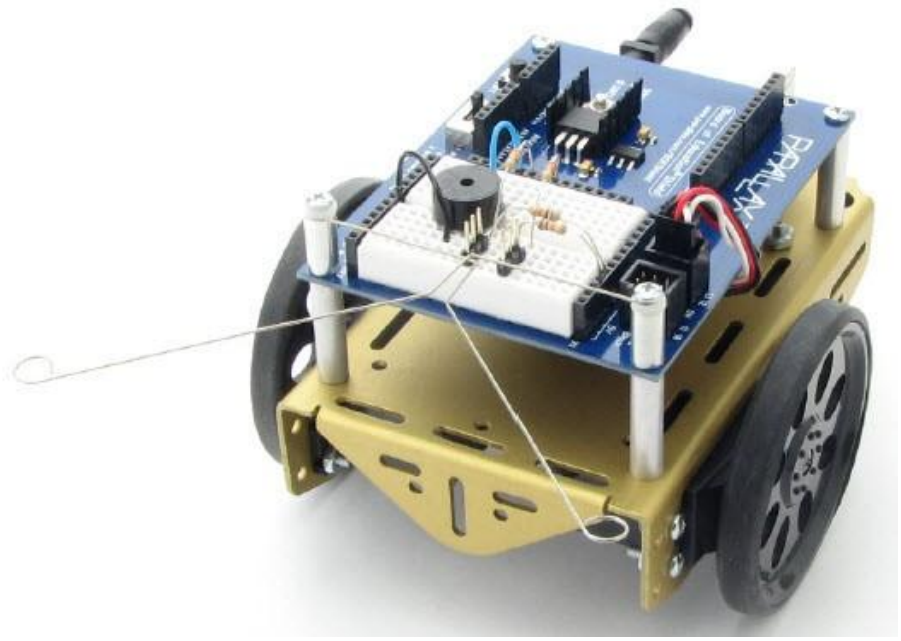
Tactile Navigation with Whiskers

Tactile switches are also called bumper switches or touch switches, and they have many uses in robotics. A robot programmed to pick up an object and move it to another conveyer belt might rely on a tactile switch to detect the object. Automated factory lines might use tactile switches to count objects, and to align parts for a certain step in a manufacturing process. In each case, switches provide inputs that trigger some form of programmed output. The inputs are electronically monitored by the equipment's processor, which takes different actions depending on if the switch is pressed or not pressed.

In this lab, you will build tactile switches, called whiskers, onto your BOE Shield-Bot and test them. You will then program the BOE Shield-Bot to monitor the states of these switches, and to decide what to do when it encounters an obstacle. The end result will be autonomous navigation by touch.

Tactile Navigation

Whisker switches give the BOE Shield-Bot the ability to sense its surroundings through touch as it roams around, much like a cat's whiskers.`



Activity 1: Build and Test the Whiskers

Remember subsystem testing? First, we'll build the whiskers circuits and write code to check their input states before using them in navigation sketches.

Whisker Circuit and Assembly

Gather the wisker hardware in the parts list.

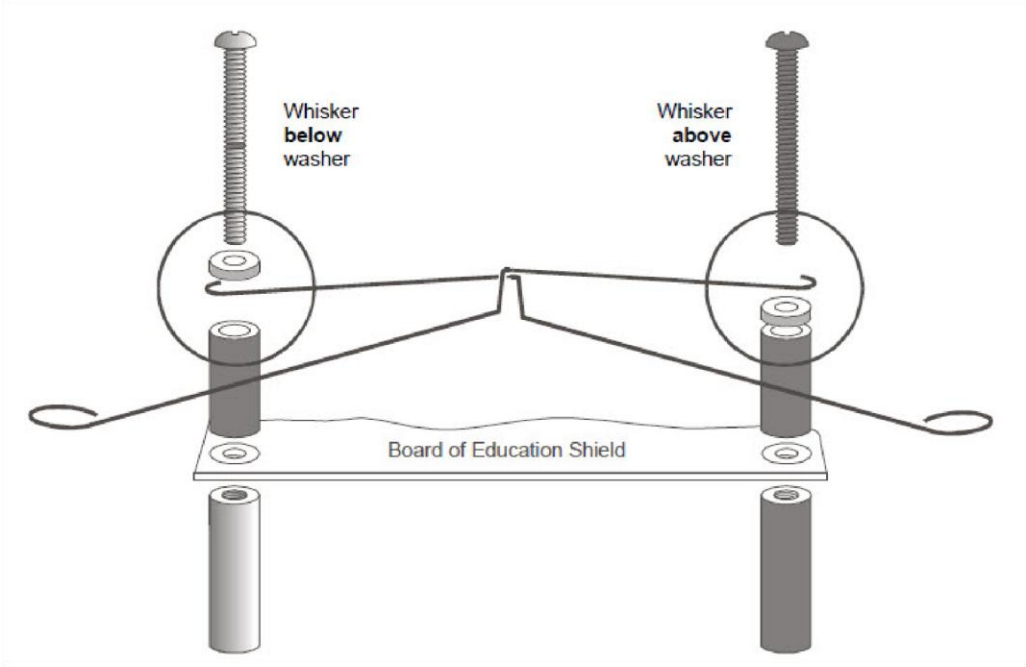
Disconnect power from your board and servos.

Parts List

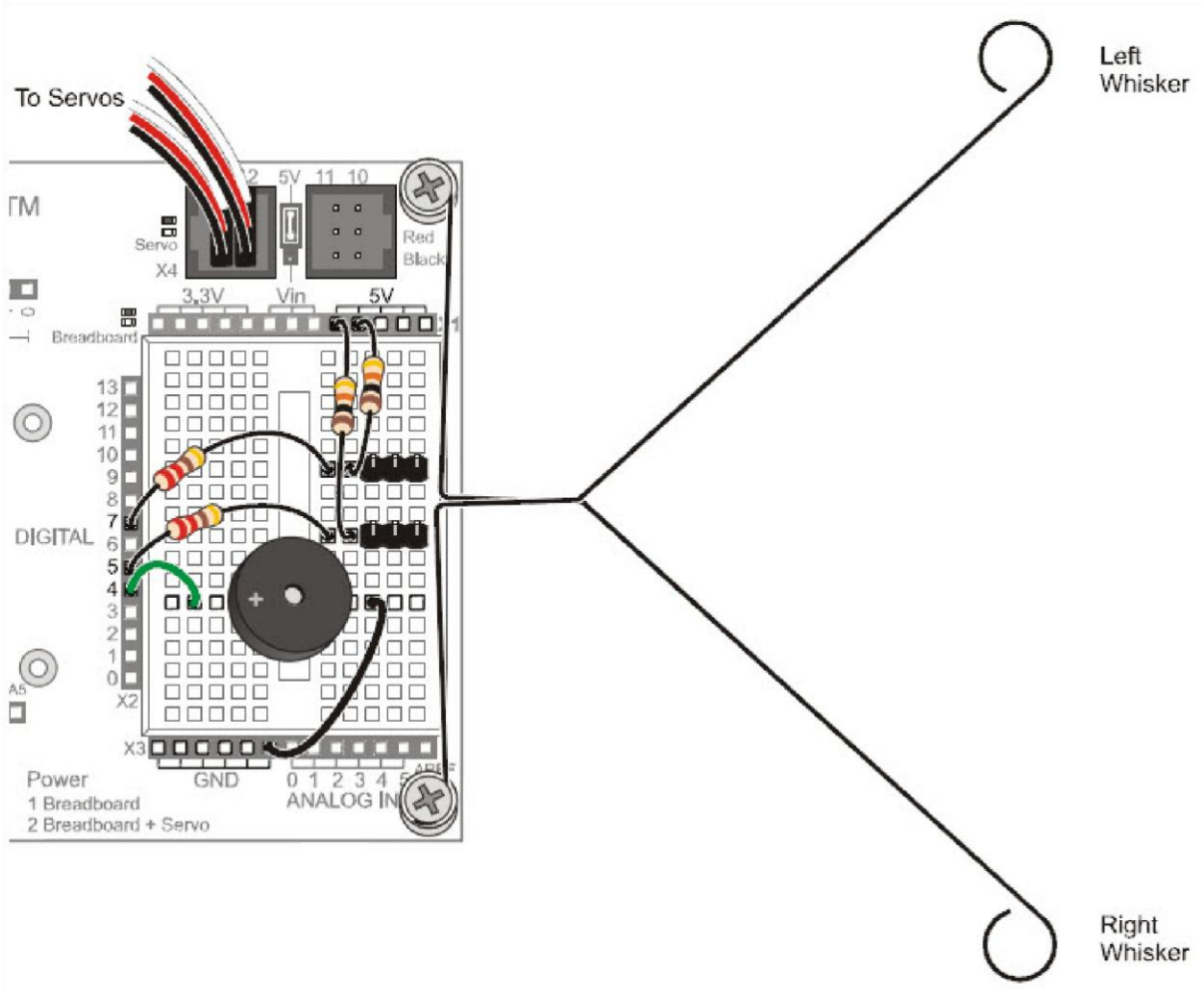
(2) whisker wires
(2) 7/8″ pan head 4-40 Phillips screws
(2) ½″ round spacer
(2) nylon washers, size #4
(2) 3-pin m/m headers
(2) resistors, 220 Ω (red-red-brown)
(2) resistors, 10 kΩ (brown-black-orange)

Building the Whiskers

- Remove the LED circuits that were used as signal monitors while testing the servo navigation.
- Remove the two front screws that hold your board to the front standoffs.
- Thread a nylon washer and then a ½″ round spacer on each of the 7/8″ screws.
- Attach the screws through the holes in your board and into the standoffs below, but do not tighten them all the way yet.
- Slip the hooked ends of the whisker wires around the screws, one above a washer and the other below a washer, positioning them so they cross over each other without touching.
- Tighten the screws into the standoffs.



- Use the 220 Ω resistors (red-red-brown) to connect digital pins 5 and 7 to their corresponding 3-pin headers.
- Use the 10 kΩ resistors (brown-black-orange) to connect 5 V to each 3-pin header.
- Make sure to adjust each whisker so that it is close to, but not touching, the 3-pin header on the breadboard. A distance of about 1/8″ (3 mm) is about right.



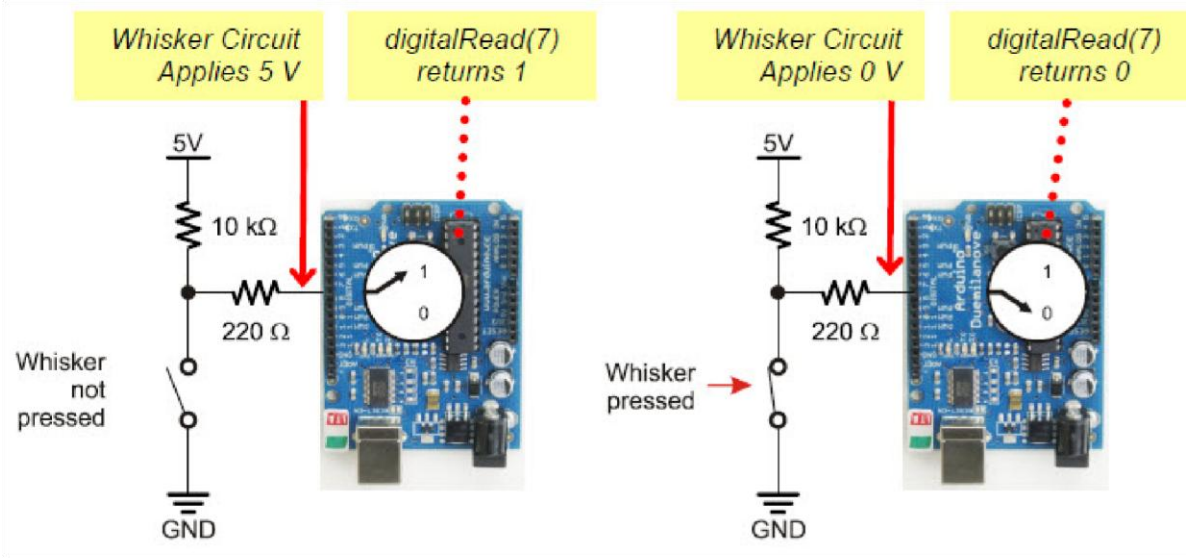
How Whisker Switches Work

The whiskers are connected to ground (Vss) because the plated holes at the outer edge of the board are all connected to Vss. The metal standoffs and screws provide the electrical connection to each whisker.

Since each whisker is connected to digital I/O, the Arduino can be programmed to detect which voltage is applied to each circuit, 5 V or 0 V. First, set each pin to input mode with `pinMode(pin, mode)`, and then detect the pin's state, HIGH or LOW, with `digitalRead(pin)` function.

Take a look at Figure 5-5. On the left, the r circuit applies 5 V when the whisker is not pressed, so `digitalRead(7)` returns 1 (`HIGH`). On the right, the circuit applies 0 V when the whisker is pressed, so `digitalRead(7)` returns 0 (`LOW`).

Most importantly, your sketch can store the return values in variables, such as `wLeft` and `wRight`, and then use them to trigger actions or make decisions. The next example sketch will demonstrate how.



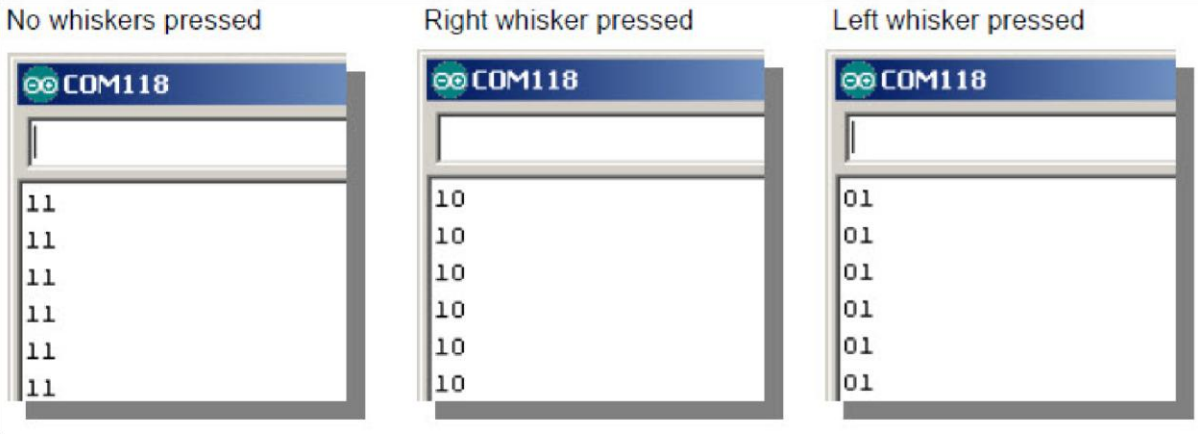
Switch Lingo: Each whisker is both the mechanical extension and the ground electrical connection of a *normally open* (off until pressed) *momentary* (on only while pressed) *single-pole* (one set of electrical contact points), *single-throw* (only one position conducts) switch.

Testing the Whiskers

The next sketch tests the whiskers to make sure they are functioning properly, by displaying the binary values returned by `digitalRead(7)` and `digitalRead(5)`. This way, you can press each whisker against its 3-pin header on the breadboard, and see if the Arduino's digital pin is sensing the electrical contact.

When neither whisker is pressed up against its 3-pin header, you can expect your Serial Monitor to display two columns of
1's, one for each whisker. If you press just the right whisker, the right column should report 0, and the display should read
10. If you press just the left whisker, the left column should report 1 and the display should read 01. Of course, if you press

both whiskers, it should display 00.



Active-low Output

The whisker circuits are wired for *active-low output*, which means that they each send a low signal when they are pressed (active) and a high signal when they are not pressed. Since **`digitalRead`** returns 0 for a low signal and 1 for a high signal, 0 is what tells your sketch that a whisker is pressed, and 1 tells it that a whisker is not pressed.

- ✓ Enter, save, and upload TestWhiskers to your Arduino.
- ✓ Reconnect the USB cable and set the 3-position switch to position 1.
- ✓ As soon as the sketch is finished uploading, open the Serial Monitor.
- ✓ Leave the USB cable connected so that the Arduino can send serial messages to the Serial Monitor.

Example Sketch: DisplayWhiskerStates

```
/*
 * Robotics with the BOE Shield -
DisplayWhiskerStates  * Display left and right
whisker states in Serial Monitor.  * 1 indicates
no contact; 0 indicates contact.
 */

void setup()                      // Built-in initialization block
{
tone(4, 3000, 1000);            // Play tone for 1 second

delay(1000);                      // Delay to finish tone

pinMode(7, INPUT);                // Set right whisker pin to input
pinMode(5, INPUT);                // Set left whisker pin to input

  Serial.begin(9600);              // Set data rate to 9600 bps
}


void loop()                       // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);    // Copy left result to wLeft

byte wRight = digitalRead(7);     // Copy right result to wRight

  Serial.print(wLeft);                  // Display left whisker state
Serial.println(wRight);                 // Display right whisker state

  delay(50);                            // Pause for 50 ms
}
```

Look at the values displayed in the Serial Monitor. With no whiskers pressed, it should display 11, indicating 5 V is applied to both digital inputs (5 and 7).

Press the right whisker into its three-pin header, and note the values displayed in the Serial Monitor. It should now read 10.

Release the right whisker and press the left whisker into its three-pin header, and note the value displayed in the Serial Monitor again. This time it should read 01.

Press both whiskers against both three-pin headers. Now it should read 00.

If the whiskers passed all these tests, you're ready to move on. If not, check your sketch and circuits for errors.

These steps are important! You need to make sure your circuit and code pass these tests before continuing.

How DisplayWhiskerStates Works

In the `setup` function, `pinMode(7, INPUT)` and `pinMode(5, INPUT)` set digital pins 7 and 5 to input so they

can monitor the voltages applied by the whisker circuits.

```
  pinMode(7, INPUT);          // Set right whisker pin to input
pinMode(5, INPUT);          // Set left whisker pin to input
```

In the `loop` function, each call to `digitalRead` returns a 0 if the whisker is pressed or 1 if it is not. Those values get copied to variables named `wLeft` and `wRight`, which are short for whisker-left and whisker-right.

```
  byte wLeft = digitalRead(5);     // Copy left result to
wLeft    byte wRight = digitalRead(7);    // Copy right
result to wRight
```

Next, `Serial.print` displays the value of `wLeft` to the Serial Monitor, and `Serial.println` displays the value of `wRight` and a carriage return.

```
  Serial.print(wLeft);             // Display left whisker state
  Serial.println(wRight);          // Display right whisker state
```

Before the next repetition of the `loop` function, there's a `delay(50)`. This slows down the number of messages the Serial Monitor receives each second. Although it's probably not needed, we leave it in to prevent possible computer buffer overruns (too much data to store) for older hardware and certain operating systems.

Your Turn – Nesting Function Calls

Your sketch doesn't actually need to use variables to store the values from `digitalRead`. Instead, the (1 or 0) value that `digitalRead` returns can be used directly by nesting the function call inside `Serial.print` and sending its return value straight to the Serial Monitor. In that case, your `loop` function would look like this:

```
void loop()                        // Main loop auto-repeats
{
  Serial.print(digitalRead(5));    // Display wLeft
Serial.println(digitalRead(7));  // Display wRight

  delay(50);                       // Pause for 50 ms
}
```

- ✓ Replace the `loop` function with the one above, upload the sketch, and test the whiskers to verify that it functions the same.

Field-Test the Whiskers

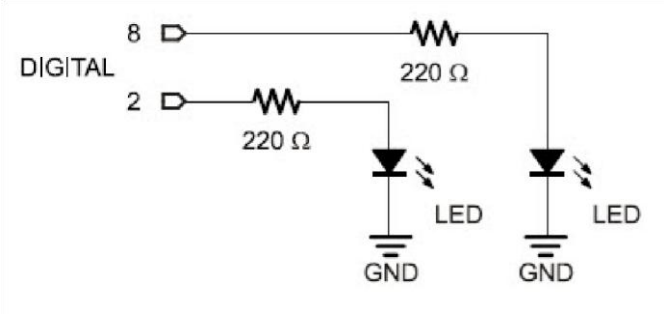
What if you have to test the whiskers at some later time away from a computer? In that case, the Serial Monitor won't be available, so what can you do? One solution would be to use LED circuits to display the whisker states. All it takes is a simple sketch that turns an LED on when a whisker is pressed or off when it's not pressed.

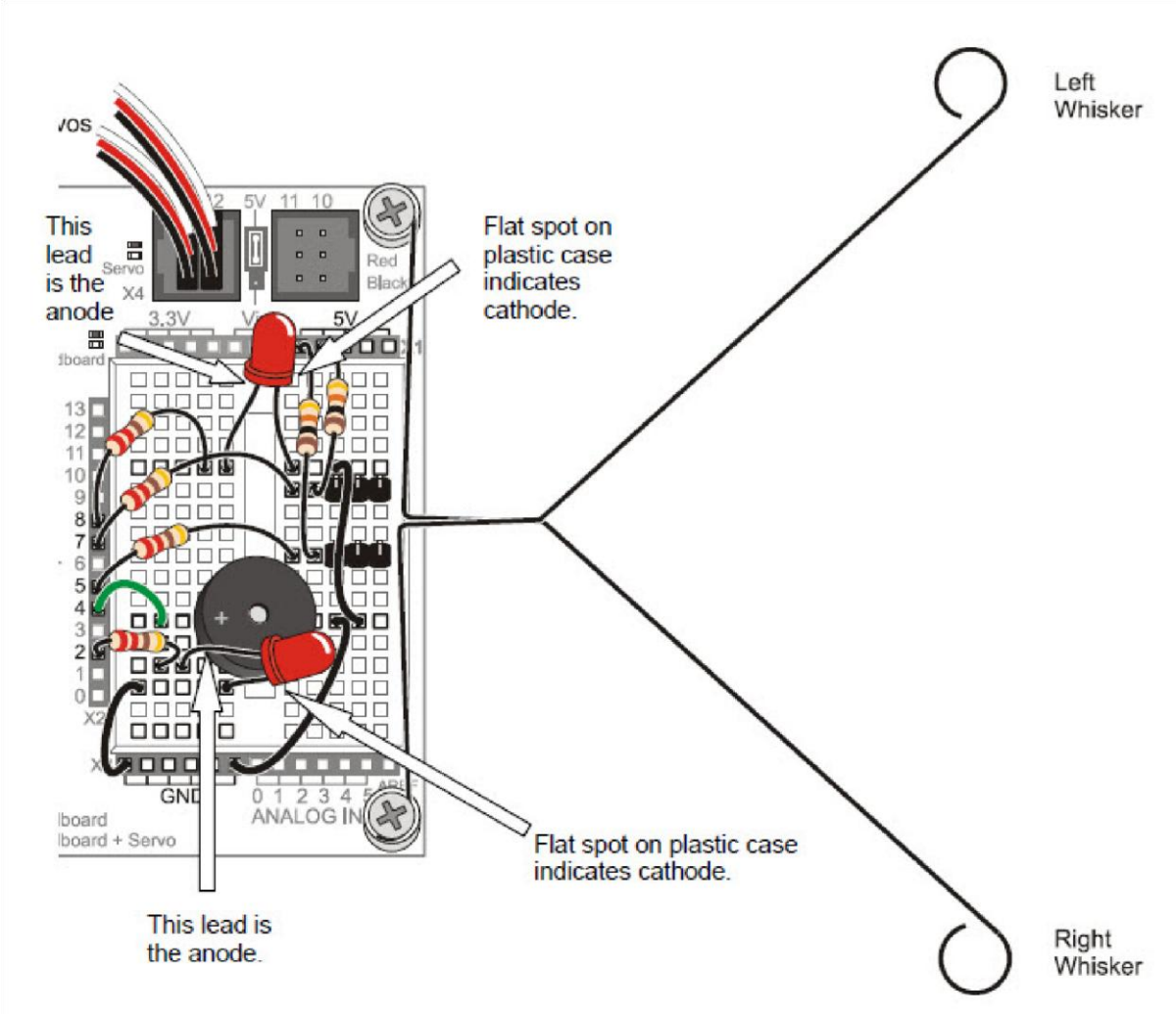
Parts List:

(2) resistors, 220 Ω (red-red-brown)
(2) LEDs, red

Build the LED Whisker Testing Circuits

- Unplug the BOE Shield-Bot's battery pack and USB cables.
- Add the circuit shown below.





Programming the LED Whisker Testing Circuits

Re-save WhiskerStates as TestWhiskersWithLEDs.

Add `pinMode` calls to the `setup` function, setting digital pins 8 and 2 to output.

```
   pinMode(8, OUTPUT);          // Left LED indicator -> output
 pinMode(2, OUTPUT);          // Right LED indicator -> output
```

To make the whisker input states control the LEDs, insert these two `if...else` statements between the `Serial.println(wRight)` and `delay(50)` commands:

```
  if(wLeft == 0)        // If left Whisker contact
{

digitalWrite(8, HIGH);    // Left LED on
  }
  else                        // If no left whisker contact
  {
    digitalWrite(8, LOW);     // Left LED off
  }

  if(wRight == 0)             // If right whisker contact
  {
    digitalWrite(2, HIGH);    // Right LED on
  }
  else                        // If no right whisker contact
  {
    digitalWrite(2, LOW);     // Right LED off
}
```

Recall that `if...else` statements execute blocks of code based on conditions. Here, if `wLeft` stores a zero, it executes the `digitalWrite(8, HIGH)` call. If `wLeft` instead stores a 1, it executes the `digitalWrite(8, LOW)` call. The result? The left LED turns on when the left whisker is pressed or off when it's not pressed. The second `if…else` statement does the same job with `wRight` and the right LED circuit.

- Set the BOE Shield's power switch to position 1.
- Reconnect the Arduino's programming cable.
- Save and upload TestWhiskersWithLeds to your Arduino.
- Test the sketch by gently pressing each whisker against its 3-pin header post in the breadboard. The red LEDs on the side of the breadboard where you pressed the whisker should emit light to indicate that the whisker has made contact.
- If both LEDs light up and just stay on no matter what, your power switch is probably in position 0. Switch it to position 1 and try again.

```
/*
* Robotics with the BOE Shield - TestWhiskersWithLeds  *
  Display left and right whisker states in Serial
  Monitor.
* 1 indicates no contact; 0 indicates contact.
* Display whisker states with LEDs.  LED on indicates
  contact;  * off indicates none.
 */

void setup()                                // Built-in initialization
block
```

```
{
  pinMode(7, INPUT);                    // Set right whisker pin to input
pinMode(5, INPUT);                      // Set left whisker pin to input
pinMode(8, OUTPUT);                     // Left LED indicator -> output
pinMode(2, OUTPUT);                     // Right LED indicator -> output

  tone(4, 3000, 1000);                  // Play tone for 1 second
delay(1000);                            // Delay to finish tone

  Serial.begin(9600);                   // Set serial data rate to
9600
}


void loop()                             // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);          // Copy left result to wLeft
byte wRight = digitalRead(7);           // Copy right result to wRight

  if(wLeft == 0)                        // If left whisker contact
  {
    digitalWrite(8, HIGH);              // Left LED on
                                                                    2

  }
  else                                  // If no left whisker contact
  {
    digitalWrite(8, LOW);               // Left LED off
  }

  if(wRight == 0)                       // If right whisker contact
  {
    digitalWrite(2, HIGH);              // Right LED on
  }
  else                                  // If no right whisker contact
  {
    digitalWrite(2, LOW);               // Right LED off
  }


  Serial.print(wLeft);                  // Display wLeft
Serial.println(wRight);                 // Display wRight
delay(50);                              // Pause for 50 ms
}
```

Navigation with Whiskers

Previously, our sketches only made the BOE Shield-Bot execute a list of movements predefined by you, the programmer. Now that you can write a sketch to make the Arduino monitor whisker switches and trigger action in response, you can also write a sketch that lets the BOE Shield-Bot drive and select its own maneuver if it bumps into something. This is an example of autonomous robot navigation.

Whisker Navigation Overview

The RoamingWithWhiskers sketch makes the BOE Shield-Bot go forward while monitoring its whisker inputs, until it encounters an obstacle with one or both of them. As soon as the Arduino senses whisker electrical contact, it uses `an if…else if…else` statement to decide what to do. The decision code checks for various whisker pressed/not pressed combinations, and calls navigation functions from to execute back-up-and-turn manoeuvres. Then, the BOE Shield-Bot resumes forward motion until it bumps into another obstacle.

Example Sketch: RoamingWithWhiskers

Let's try the sketch first, and then take a closer look at how it works.

- Set the 3-position switch to position 1.
- Reconnect the BOE Shield-Bot's battery pack to the Arduino.
- Enter, save, and upload RoamingWithWhiskers.
- Disconnect the BOE Shield-Bot from its programming cable, and set the power switch to 2.
- Put the BOE Shield-Bot on the floor, and try letting it roam. When it contacts obstacles in its path with its whisker switches, it should back up, turn, and then roam in a new direction.

```
// Robotics with the BOE Shield - RoamingWithWhiskers
// Go forward.  Back up and turn if whiskers indicate BOE Shield bot
bumped // into something.

#include <Servo.h>                              // Include servo library


Servo servoLeft;                          // Declare left and right servos
Servo servoRight;


void setup()                             // Built-in initialization block
```



```
{
  pinMode(7, INPUT);                     // Set right whisker pin to input
pinMode(5, INPUT);                      // Set left whisker pin to input

tone(4, 3000, 1000);                       // Play tone for 1 second
delay(1000);                               // Delay to finish tone

  servoLeft.attach(13);                  // Attach left signal to pin 13
servoRight.attach(12);                  // Attach right signal to pin 12
}


void loop()                                    // Main loop auto-repeats
```

```
{
byte wLeft = digitalRead(5);          // Copy left result to wLeft
byte wRight = digitalRead(7);           // Copy right result to wRight

  if((wLeft == 0) && (wRight == 0))          // If both whiskers contact
  {
    backward(1000);                          // Back up 1 second
    turnLeft(800);                           // Turn left about 120 degrees
  }
  else if(wLeft == 0)                      // If only left whisker contact
  {
    backward(1000);                          // Back up 1 second
    turnRight(400);                          // Turn right about 60 degrees
  }
  else if(wRight == 0)                      // If only right whisker contact
  {
    backward(1000);                          // Back up 1 second
    turnLeft(400);                           // Turn left about 60 degrees
  }
  else                                     // Otherwise, no whisker contact

{
forward(20);                              // Forward 1/50 of a second
}
}

void forward(int time)                       // Forward function
{
servoLeft.writeMicroseconds(1700);        // Left wheel counterclockwise
servoRight.writeMicroseconds(1300);        // Right wheel clockwise
delay(time);                               // Maneuver for time ms
}

void turnLeft(int time)                      // Left turn function
{
servoLeft.writeMicroseconds(1300);         // Left wheel clockwise
servoRight.writeMicroseconds(1300);        // Right wheel clockwise
delay(time);                               // Maneuver for time ms
}

void turnRight(int time)                     // Right turn function
{
servoLeft.writeMicroseconds(1700);         // Left wheel counterclockwise
servoRight.writeMicroseconds(1700);        // Right wheel counterclockwise
delay(time);                               // Maneuver for time ms
}

void backward(int time)                      // Backward function
{
servoLeft.writeMicroseconds(1300);         // Left wheel clockwise
servoRight.writeMicroseconds(1700);        // Right wheel counterclockwise
delay(time);                               // Maneuver for time ms



}
```

How RoamingWithWhiskers Works

The `if...else if...else` statement in the `loop` function checks the whiskers for any states that require attention. The statement starts with `if((wLeft == 0) && (wRight == 0))`. Translated to English, it reads "if the `wLeft` variable AND the `wRight` variable both equal zero." If both variables are zero, the two calls in the `if` statement's code block get executed: `backward(1000)` and `turnLeft(800)`.

```
  if((wLeft == 0) && (wRight == 0)) // If both whiskers contact
  {
    backward(1000);                 // Back up 1 second
    turnLeft(800);                  // Turn left about 120
degrees   }
```

In the `if…else if…else` statement, the sketch skips code blocks with conditions that are not true, and keeps checking until it either finds a condition that's true or runs out of conditions. When the sketch finds a true statement, it executes whatever is in its code block, then it skips to the end of the `if…else if…else` statement without checking any more conditions, and moves on to whatever else comes next in the sketch.

So, if both whiskers are not pressed, that first `if` statement is not true and its code block is skipped. The sketch will check the first `else if` statement. So, maybe the left whisker is pressed and the calls in this statement's code block will run. After backing up for one second and turning left for 0.4 seconds, the sketch skips the rest of the conditions and moves on to whatever comes after that last `else` statement.

```
  else if(wLeft == 0)               // If only left whisker contact
  {
    backward(1000);                 // Back up 1 second
    turnRight(400);                 // Turn right about 60
degrees   }
```

If it's the right whisker that detects an obstacle, the first two code blocks will be skipped, and the `if(wRight == 0)` block will run.

```
  else if(wRight == 0)              // If only right whisker contact
  {
    backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60
degrees   }
```

An `else` condition functions as a catch-all for when none of the statements preceding it were true. It's not required, but in this case, it's useful for when no whiskers are pressed. If that's the case, it allows the BOE Shield-Bot to roll forward for 20 ms. Why so little time before the loop repeats? The small forward time before rechecking allows the BOE Shield-Bot to respond quickly to changes in the whisker sensors as it rolls forward.

```
  else                                // Otherwise, no whisker contact
  {
    forward(20);              // Forward 1/50 of a second

      }
```

The forward, backward, turnLeft and turnRight functions were introduced in Chapter 4, Activity #5 [35], and are used in the MovementsWithSimpleFunctions [36]sketch. These functions certainly simplified the coding.

You can also modify the sketch's `if...else if...else` statements to make the LED indicators broadcast which maneuver the BOE Shield-Bot is running. Just add `digitalWrite` calls that send `HIGH` and `LOW` signals to the indicator LED circuits. Here is an example:

```
  if((wLeft == 0) && (wRight == 0)) // If both whiskers contact
  {
    digitalWrite(8, HIGH);          // Left LED on
digitalWrite(2, HIGH);          // Right LED on
backward(1000);                 // Back up 1 second
    turnLeft(800);                  // Turn left about 120 degrees
  }
  else if(wLeft == 0)               // If only left whisker contact
  {
    digitalWrite(8, HIGH);          // Left LED on
digitalWrite(2, LOW);           // Right LED off
backward(1000);                 // Back up 1 second
    turnRight(400);                 // Turn right about 60 degrees
  }
  else if(wRight == 0)              // If only right whisker contact
  {
    digitalWrite(8, LOW);           // Left LED off
digitalWrite(2, HIGH);          // Right LED on
backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60 degrees
  }
  else                              // Otherwise, no whisker contact
  {
    digitalWrite(8, LOW);           // Left LED off
digitalWrite(2, LOW);           // Right LED off
    forward(20);                    // Forward 1/50 of a second
  }
```

- Modify the `if...else if...else` statement in RoamingWithWhiskers to make the BOE Shield-Bot broadcast its maneuver using the LED indicators.
- Remember to set the digital pins to outputs in the setup function so they can actually supply current to the LEDS:

```
    pinMode(8, OUTPUT);          // Left LED indicator -> output
    pinMode(2, OUTPUT);          // Right LED indicator -> output
```