Introduction to Sensing

This chapter will introduce you to sensing. You will become familiar with the Devantech SRF05 ultrasonic range sensor and then use it to create a “tractor beam.” Your robot will be able to sense and follow your hand, as if there were an invisible beam attaching your robot to your hand.

Sonar Range Sensing

Your IntelliBrain-Bot includes a sonar range sensor like the one shown in Figure 5-1. This sensor is able to measure the distance to an object in front of your robot by “pinging.”

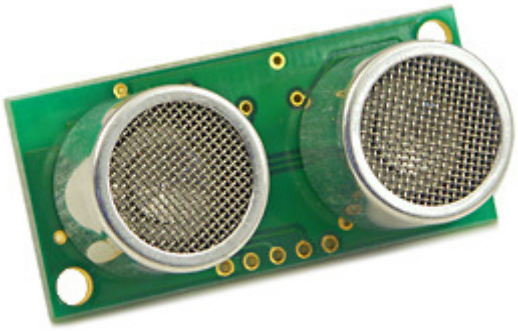

Fig. 5-1 – Sonar SRF05

This sensor detects objects by generating a short high frequency sound, and then listening for an echo. The sensor will hear an echo if there is an object in front of your robot, as shown in Figure 5-2. If there is no object, the sound will not be reflected, and the sensor will not detect an echo.

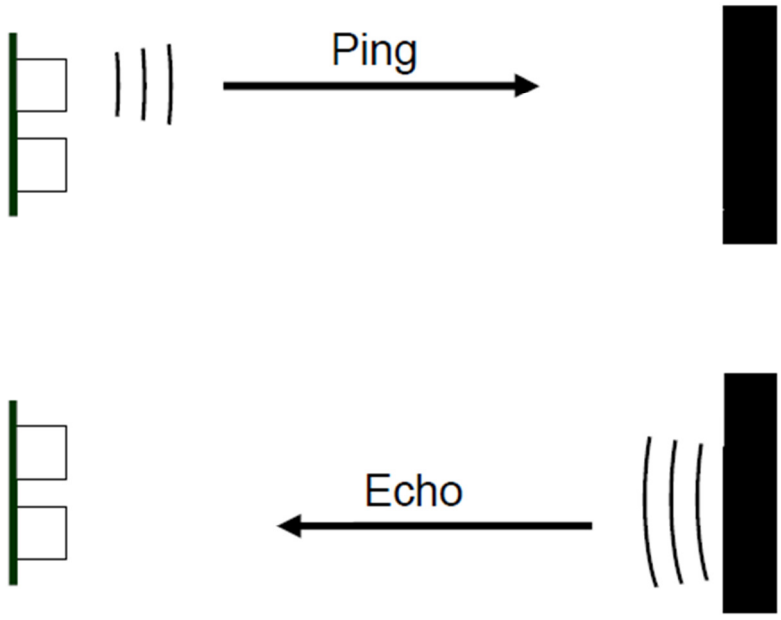

Fig. 5-2 – Sonar Ping and Echo

Your program can determine the distance to the object by measuring the time between issuing the ping and hearing the echo. The further away the object is, the longer it will take for the echo to return to the sensor.

Programming the SRF05 Sensor

The RoboJDE API includes a class that provides support for the SRF05 Sensor. This class is named ParallaxPing; This class was create for another sonar sensor model called Parallax Ping, but can be used with the SRF05 sonar, because the sensors are compatible. However, rather than using the ParallaxPing class we will program the sensor directly. This will enable you to better understand how the sensor functions.

The SRF05 sensor interfaces to the IntelliBrain 2 robotics controller via three wires. See Figure 5-3 for connections.

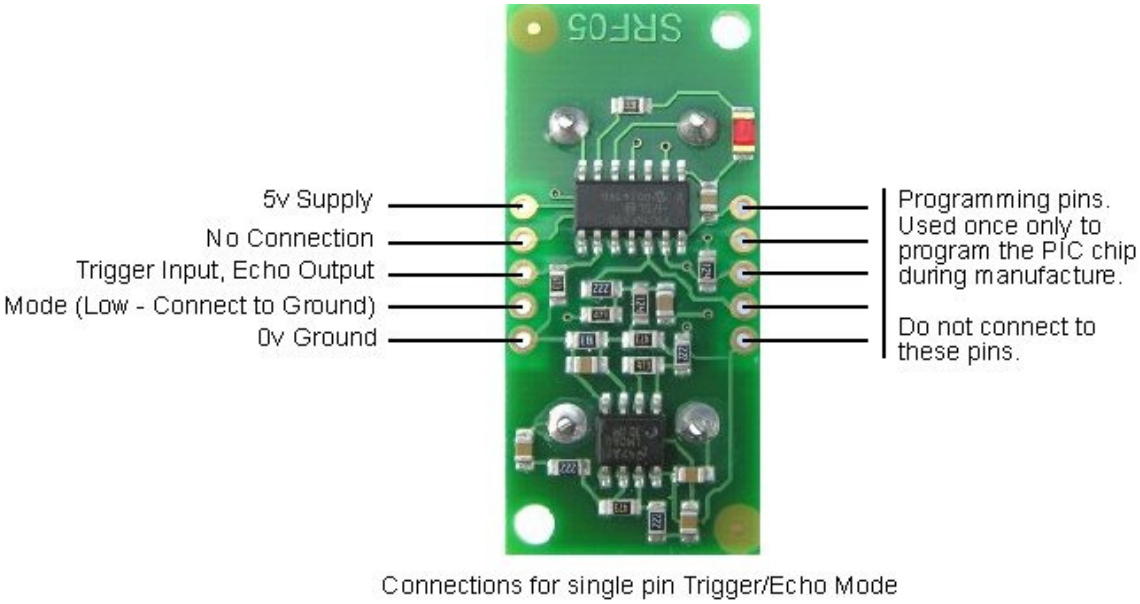

Fig. 5-3 – Sonar SRF05 connections

The red and black wires provide the sensor with power (red) and the ground (black). The white wire is the signal wire. Through clever design, one signal wire is used to both trigger the sensor to issue a sound pulse and to communicate the echo delay back to the robotics controller. Figure 5-4 illustrates this. The SRF05 sensor must be connected to one of four digital input/output IntelliBrain 2 robotics controller ports that provides pulse input and output features. These are the ports labeled IO3, IO4, IO5 and IO6. The SRF05 sensor should be attached to port IO4 on the IntelliBrain 2 robotics controller.

Your program can trigger the SRF05 sensor to send a sound pulse by quickly switching the IO4 port signal on and off. You can imagine this as if you were flipping a light switch on and off very quickly; however, your program must do it faster than humanly possible. In fact, the pulse is so short that it is measured in millionths of a second, or microseconds. As indicated in Figure 5-3, the trigger pulse must be greater than 10 microseconds (usec) long. This timing is so short that the pulse can't be generated by software; the microcontroller chip on the IntelliBrain 2 robotics controller must generate it.

Once the SRF05 sensor receives the trigger pulse, it issues a sound pulse after first holding off for 750 microseconds. The hold off period gives the software on the robotics controller a chance to switch the signal from an output to an input. On the IntelliBrain 2 robotics controller, the virtual machine software takes care of briefly switching the port to an output when your program calls the method to output a pulse, so your program doesn't need to be concerned with this. The SRF05 sensor sets the signal to its high (+5 volts) when it issues the sound pulse. It leaves the signal high until it hears the first echo, at which time it sets the signal low (0 volts). Therefore, the time for the sound burst to travel to the closest object and bounce back to the sensor is the amount of time the signal level is high. Your program can determine the echo delay by measuring the time the signal level is high. If the sensor does not hear an echo within 30,000 microseconds, it sets the signal low. There is also a minimum time the SRF05 sensor will leave the signal on. This is 100 microseconds. It is important to take note of the minimum and maximum echo delay. These limit the effective range of the SRF05 sensor.

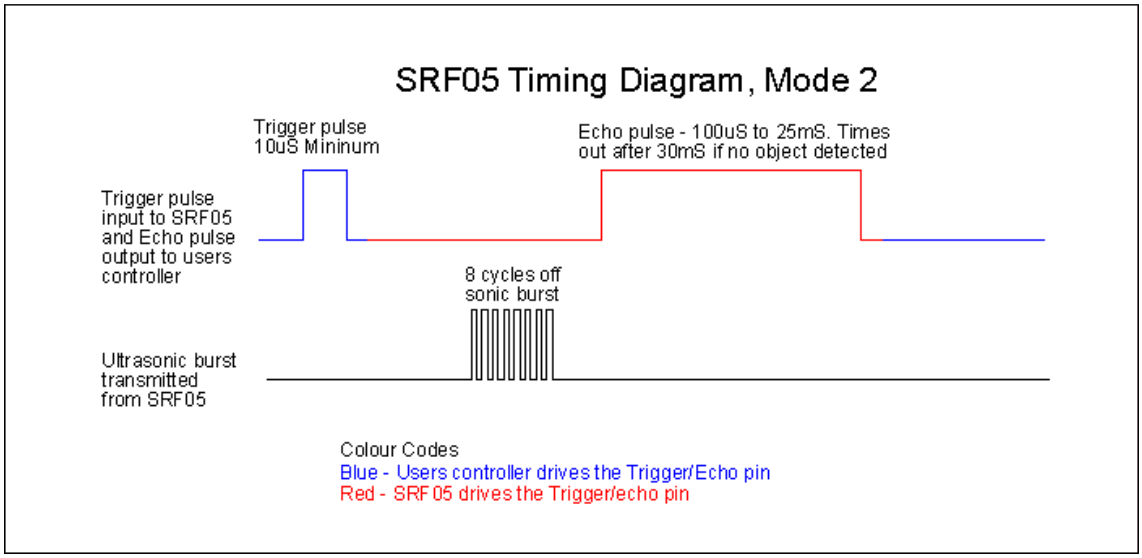

Figure 5-4 –SRF05 Sensor Signals

Create a program to measure and display the time for sound to travel from the SRF05 sensor to the nearest object by doing the following:

1. Review the API documentation for IntelliBrain.getDigitalIO and IntelliBrainDigitalIO.
2. Create a new project named SRF05Test.
3. Add the following import statements:

```
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrainDigitalIO;
```

4. Replace the comment in the main method with a try-catch statement.

```
try {

}
```

```
catch (Throwable t) {
   t.printStackTrace();
}
```

5. Add statements within the try statement to get the display object and print the name of your program.

```
Display display = IntelliBrain.getLcdDisplay();
display.print(0, "SRF05 Test");
```

6. Add statements to get the port object for IO4 and enable it to be used for pulse measurement.

```
IntelliBrainDigitalIO pingPort =IntelliBrain.getDigitalIO(4);
pingPort.enablePulseMeasurement(true);
```

7. Add a loop that loops forever.

```
while (true) {

}
```

8. Within the loop, use the pulse method of the port object to issue a trigger pulse 20 microseconds in duration.

```
pingPort.pulse(20);
```

9. Add a statement to put your program to sleep for 50 milliseconds, so it will not continue until after the longest possible echo delay pulse will be able to complete.

```
Thread.sleep(50);
```

10. Add a statement to read and display the duration of the echo delay pulse.

```
display.print(1, "Time: " + pingPort.readPulseDuration());
```

The readPulseDuration returns the pulse duration measurement made by the microcontroller chip.

11. Add another sleep method call such that your program takes two readings per second.

```
Thread.sleep(450);
```

12. Build, load and test your program.

Hold your hand steady in front of the sensor. Note the reading, then move your hand closer or further away, noting the new reading. The reading will increase as you move your hand away and decrease as you move it closer. The approximate range of the readings will be between 100 and 25000 microseconds.

Measuring the Speed of Sound

You can use your SRF05Test program to measure the speed of sound. With your program running, hold your robot such that the SRF05 sensor is 50 cm foot from a wall. The sound pulse

will travel 100 cm as it goes from the sensor to the wall and back to the sensor. You can calculate the speed of sound by dividing the distance traveled by the time of travel. In this case the distance is 100 cm and the time is the echo delay measured by your program.

speed of sound = distance traveled / time of travel

speed of sound = 100 cm / 2900 usec = 34482,76 cm/sec

When you do this experiment, you will measure an echo delay of approximately 1800 microseconds. Performing the calculation above, you will find the speed of sound is approximately **34483 cm/sec**.

Calculating Distance

Knowing the speed of sound, you can now modify your program to display distance rather than time. To do this, you will need to use the equation:

distance = rate * time

The rate is the speed of sound. The distance we are interested in is the distance to the nearest object. This is one half the distance the sound pulse travels, so we need to divide the result by two. Substituting yields:

distance = speed of sound * round trip time / 2

Substituting the value for the speed of sound and accounting for unit conversions yields:

distance = 34483 cm/sec * 1/1,000,000 sec/usec * round trip time / 2

or

distance = round trip time / 58

where round trip time is in microseconds and the result is in cm.

Update your program to display distance, as follows:

13. Replace the existing statement to display the sensor reading with a statement to read the round trip time and assign it to a new local variable.

```
int roundTripTime = pingPort.readPulseDuration();
```

14. Add statements to only display the distance if the sensor reading is within its valid range (100 – 25000), otherwise display "--" to indicate no object is in range.
(115 – 18500)

```
if (roundTripTime < 100 || roundTripTime > 25000)
   display.print(1, "Distance: --");
else
   display.print(1, "Distance: " + roundTripTime / 58 +
"cm");
```

15. Build, load and test your program. Hold your hand at various distances. Verify the distance values displayed are correct.

Sensor Performance

The SRF05 sensor can sense objects in a cone shaped region directly ahead of the sensor, as shown in Figure 5-5. The dimensions of the region vary based on the properties of the object being sensed, as well as the properties of the surrounding surfaces.

Figure 5-5 – Effective Sensing Region of SRF05 Sensor

Using the SRF05 Sensor

An interesting and fun application of the SRF05 sonar range sensor is to use it to create a "tractor beam" effect. A tractor beam is a science fiction device that forms an invisible connection between two objects.

Let's write a program using the SRF05 sensor to create a tractor beam. Once you have completed your program, your IntelliBrain-Bot educational robot will be able to form an invisible connection to an object. Your robot will move forward or back to maintain a fixed space between your robot and the object. When you place your hand in front of your robot, it will follow it forward and back, creating the illusion of an invisible beam between your hand and your robot.

Surprisingly, creating this program is not nearly as challenging as it might seem. All that your program needs to do is use the SRF05 sensor to repeatedly measure the distance to the nearest object, adjusting the power to the motors with each new measurement. If the distance is too large, your program will need to apply power to the motors to move your robot forward. If the distance is too small, your program will need to apply power to the motors to move your robot backwards. If the distance is just right, your program will need to turn power to the motors off.

Create your program using the following procedure:

1. Review the API documentation for the SonarRangeFinder and ParallaxPing classes. Pay particular attention to the getDistanceCm method.

2. Create a new project named "TractorBeam."

3. Add import statements for the library classes your program will refer to.

```
import com.ridgesoft.intellibrain.IntelliBrain;
```

```
import com.ridgesoft.io.Display;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.Motor;
import com.ridgesoft.robotics.ContinuousRotationServo;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;
```

4. Place a try-catch statement in the main method.

```
try {
}
catch (Throwable t) {
t.printStackTrace();
}
```

5. Add statements within the try clause to obtain the Display for the LCD device and then print the name of your program.

```
Display display = IntelliBrain.getLcdDisplay();
display.print(0, "Tractor Beam");
```

6. Create motor objects for the two motors.

```
Motor leftMotor = new ContinuousRotationServo(
IntelliBrain.getServo(1), false, 14);
Motor rightMotor = new ContinuousRotationServo(
IntelliBrain.getServo(2), true, 14);
```

Create a SonarRangeFinder object for the SRF05 sensor.

```
SonarRangeFinder pingSensor =
new ParallaxPing(IntelliBrain.getDigitalIO(3));
```

8. Create a loop that runs forever.

```
while (true) {

}
```

9. Add statements within the loop to issue a sonar ping. Wait long enough for the echo to be heard and then read the range in inches.

```
pingSensor.ping();
Thread.sleep(100);
float range = pingSensor.getDistanceCm();
```

10. Add statements to display the range or "--" if there is no object in range.

```
if (range > 0.0f)
   display.print(1,  Integer.toString((int)(range  +  0.5f))  +
"--");
else
   display.print(1, "--");
```

11. Build, load and test your program.

Your program will display the number of inches to the nearest object or "--" if there is no object in range.

Implementing the Tractor Beam Effect

In order to implement the tractor beam effect we need to use distance reading measurements to control the power applied to the motors. If the distance to the object is further than desired, your program must power the motors forward. If the distance to the object is less than desired, your program must power the motors backward. Otherwise, your program must turn the motors off. We will use a second press of the START button to activate the tractor beam.

1. Add the following statements immediately prior to the while loop.

```
PushButton startButton = IntelliBrain.getStartButton();
startButton.waitReleased();
boolean go = false;
```

2. Add the following statements immediately after the statement to read the range.

```
if (go) {

}
else if (startButton.isPressed()) {
   go = true;
}
```

These statements cause your program to wait for the second press of the START button before executing the code in the go clause of the if statement.

3. Within the go clause of the if statement above, add another if statement which checks to see if there is an object within 50 cm of your robot. If there is not, stop the motors.

```
if (range > 0.0f && range < 50.0f) {

}
else {
   leftMotor.stop();
   rightMotor.stop();
}
```

4. Add statements within the range check clause of the above if statement to: power the motors forward if there is an object more than 18 cm from your robot, power the motors in reverse if there is an object within 15 cm of your robot, or stop the motors if there is an object between 15 and 18 cm of your robot.

```
if (range > 18.0f) {
   rightMotor.setPower(Motor.MAX_FORWARD);
   leftMotor.setPower(Motor.MAX_FORWARD);
}
else if (range < 15.0f) {
   leftMotor.setPower(Motor.MAX_REVERSE);
   rightMotor.setPower(Motor.MAX_REVERSE);
```

```
}
else {
   leftMotor.stop();
   rightMotor.stop();
}
```

5. Build, load and test your program.

Place your hand in front of the sensor. If your hand is closer than 15 cm your robot should move backwards. If your hand is further than 18 cm inches your robot should move forward. Notice that your robot is a bit jerky when it starts and stops. This is because of the way your program is controlling the motors. In the next exercise we will make your robot operate more smoothly.

Proportional Control

In the previous exercise you were able to create the tractor beam effect by powering the motors to correct for a difference (error) between the actual and the desired distance to the nearest object. There is error if the nearest object is not between 15 and 18 cm away from your robot. Your program attempts to eliminate the error by powering the motors in the direction that will reduce the error. Your program currently does this in a simple minded way: by going forward or back at full power until there is no error. This results in a jerky response. Your robot will frequently overshoot the desired position because the motors are either on at full power or turned off. This approach is often called “bang-bang” control because your program bangs the power from one extreme to the other: full power or no power. You can imagine if you were in a car where the driver used this technique to control the speed it would result in a very uncomfortable ride!

We can correct the jerky behavior by modifying your program to use “proportional control.” With proportional control the amount of power applied to correct for the error is varied in proportion to the error. If the error is small, your program applies a small amount of power to the wheels. If the error is large, your program applies a large amount of power to the wheels. As the error decreases, your program decreases the power. If there is no error, your program turns the motors off. You can accomplish this by making the power proportional to error, as follows:

power = error * gain

The gain value is a constant multiplier which makes the power level proportional to the error. You can see the power will be zero if there is no error, and the greater the error, the greater the power. The value of the gain has to be carefully determined such that your robot will not overreact or under react to error. A reasonable gain value can be determined by experimentation.

With the current tractor beam exercise, there is no error when the object in front of your robot is exactly 15 cm away; therefore, the difference between the actual range and the desired range is the error:

error = range – 15.0

Convert your program to use proportional control as follows:

1. Replace the power setting statements with proportional control calculations.

```
if (range > 0.0f && range < 50.0f) {
   float gain = 2.0f;
   float error = range - 15.0f;
   int power = (int)(gain * error);
   if (power != 0) {
      leftMotor.setPower(power);
      rightMotor.setPower(power);
   else {
      leftMotor.stop();
      rightMotor.stop();
   }
}
else {
   leftMotor.stop();
   rightMotor.stop();
}
```

Based on experimentation, a gain value of 2.0 has been determined to work well, but you might have to change it for your robot. Note that the setPower method limits the actual power to the maximum or minimum if the value of the power argument is out of this range.

2. Build, load and test your program.

Your robot will now track your hand more smoothly.