RidgeSoft™

Creating Shaft Encoders for Wheel Position Sensing

www.ridgesoft.com

Revision 1.1

Introduction

The tutorial *Programming Your Robot to Perform Basic Maneuvers* demonstrated how you can use timed sequences of commands to program your robot to perform simple maneuvers. With timing-based control, your robot lacks a means to recognize and compensate for varying operating conditions. As a result, its performance can be significantly affected by changes in battery voltage, friction and other factors.

By adding sensors to provide feedback, you can enable your robot to perform more consistently. Instead of using pre-programmed predictions, your program will be able to control your robot according to measurements of its actual performance.

In this tutorial you will use a Fairchild QRB1134 infrared photo-reflector to sense the position of each of your robot's wheels. By mounting each sensor adjacent to a wheel, as shown in Figure 1, it will provide an input signal to your program. Your program will use this signal to implement a shaft encoder, which will keep track of the position of the wheel.

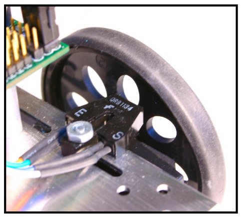
Figure 1 – Wheel Position Sensor

Although tracking the position of your robot's wheels may not seem too exciting, doing this will allow your robot to eventually keep track of its location. It will also allow your robot to navigate from place to place.

As you complete this tutorial you will learn how to sample input from an analog sensor, such as the QRB1134 infrared photo-reflector. You will also learn about shaft encoding and you will create an encoder class that uses this sensor to monitor the position of an adjacent wheel.

Before You Get Started

This tutorial builds on topics covered in the following tutorials:

Creating Your First IntelliBrain Program
Programming Your Robot to Perform Basic Maneuvers
Creating a User Interface for Your Robot

If you are not already familiar with the concepts covered in these tutorials, you should complete them first before attempting this tutorial. This and other tutorials are available from the RidgeSoft web site, www.ridgesoft.com.

The programming steps in this tutorial build on the MyBot program developed in the *Creating a User Interface for Your Robot* tutorial.

Copyright © 2005 RidgeSoft, LLC

You will need an IntelliBrain-Bot educational robot kit to complete this tutorial.

Theory of Operation

The IntelliBrain-Bot educational robot comes with plastic wheels, as shown in Figure 1. Each wheel has eight spokes separated by eight oblong holes. These spokes and holes are fundamental to sensing movement of the wheel. By using an infrared photo-reflector sensor, your program can detect whether a spoke or a hole is in front of the sensor. By checking the sensor frequently, your program can check for movement of the wheel which will enable it to track the position of the wheel. This technique is known as "shaft encoding" because it relies on a sensor signal that "encodes" the angular position of a shaft, in this case the axle the wheel is mounted on.

You are probably wondering how your program can detect the spokes and holes in the wheels. A simple way to do this is to use an infrared photo-reflector sensor, such as the Fairchild QRB1134. These sensors come with the IntelliBrain-Bot kit. These are similar to the sensors that are used to turn faucets on and off automatically in public restrooms.

An infrared photo-reflector sensor consists of an infrared emitter and detector pair. The emitter emits infrared light. The detector detects infrared light. The detector has an output signal whose voltage varies depending on the intensity of the infrared light striking the detector. When the sensor is adjacent to a solid surface – for example, a spoke – infrared light from the emitter will reflect back on to the detector, as shown in the upper portion of Figure 2. The output voltage of the detector will be low in this case.

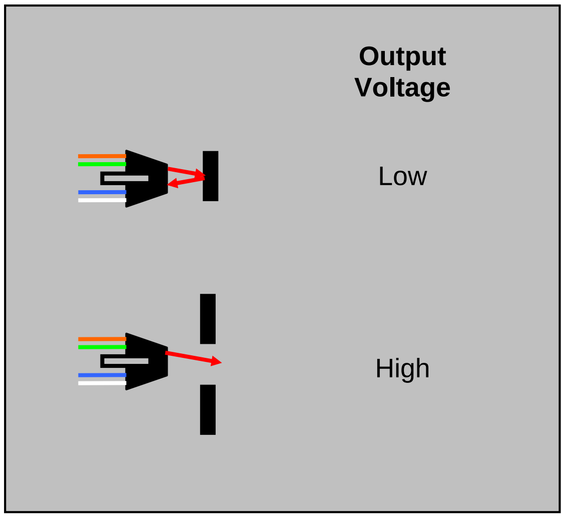

Figure 2 – Fairchild QRB1134 Photo-reflector Sensor Operation

When there is no surface adjacent to the sensor – for example, the sensor is adjacent to a hole in the wheel – infrared light from the emitter will not be reflected on to the detector. In this case, the output voltage of the detector will be high. This situation is depicted in the lower portion of Figure 2.

Copyright © 2005 RidgeSoft, LLC

By mounting a sensor adjacent to each wheel, as shown in Figure 1, your program will be able to use the output signal from the detector to sense whether a spoke or hole is adjacent to the sensor at any point in time. By checking the signal frequently your program can sense movement of the wheel.

Working with Analog Sensors

Now that you are familiar with how the photo-reflector sensors work and how you will be using them, you can extend the MyBot program from the *Creating a User Interface for Your Robot* tutorial to begin working with the sensors. The user interface foundation classes you built in that tutorial will come in handy for displaying sensor readings.

If you haven't already mounted the photo-reflector sensors on your IntelliBrain-Bot, you should do so now. Please refer to the *IntelliBrain-Bot Assembly Guide* (available from www.ridgesoft.com) for instructions on how to attach these sensors.

Sampling Analog Inputs

You will need to use the IntelliBrain robotics controller's built-in analog-to-digital (A-to-D) converter to "sample" the analog signal from the sensor. Fortunately, the RoboJDE software that comes with the IntelliBrain-Bot kit makes using the A-to-D converter very easy. All you have to do is invoke the sample method on the port object for the port to which your sensor connects.

You can get the port object using the getAnalogInput method of the IntelliBrain class. To obtain the analog input objects for the left and right wheel sensors, which you should have attached to analog ports 4 and 5, respectively, add the following lines to the main method of the MyBot class:

```
AnalogInput leftWheelInput = IntelliBrain.getAnalogInput(4);
AnalogInput rightWheelInput = IntelliBrain.getAnalogInput(5);
```

A convenient place to add these lines is right after the existing line to obtain the stopButton object. You will also need to add the following import statement near the beginning of the MyBot class:

```
import com.ridgesoft.robotics.AnalogInput;
```

You can now sample the A-to-D converter for either port by adding a line to your program to invoke the port's sample method. For example, you could use the following line to sample the left wheel sensor input:

```
int leftWheelSensorValue = leftWheelInput.sample();
```

However, don't add this to your program quite yet.

Copyright © 2005 RidgeSoft, LLC

Testing the Sensors

Whenever you add a sensor to your robot, it is a good idea to take the time to test the sensor. This will enable you to verify the sensor is connected and functioning properly. It will also allow you to experiment directly with the sensor to better understand how it works.

You can test both sensors by adding another screen to the user interface to display values sampled from the sensors. To do this, create a new class named WheelSensorScreen using the following code:

```
import com.ridgesoft.io.Display;
import com.ridgesoft.robotics.AnalogInput;

public class WheelSensorScreen implements Screen {
    private AnalogInput mLeftWheelInput;
    private AnalogInput mRightWheelInput;

    public WheelSensorScreen(AnalogInput leftWheelInput,
                             AnalogInput rightWheelInput) {
        mLeftWheelInput = leftWheelInput;
        mRightWheelInput = rightWheelInput;
    }

    public void update(Display display) {
        display.print(0,
               "L Wheel: " + mLeftWheelInput.sample());
        display.print(1,
               "R Wheel: " + mRightWheelInput.sample());
    }
}
```

As you can see, the constructor of this class has two arguments, which are the AnalogInput objects corresponding to the left and right wheel sensors. The update method uses these objects to sample the sensors before it prints the sampled values to the display.

In order to incorporate the new screen into your program, insert the following line into the list of screens in the MyBot class:

```
new WheelSensorScreen(leftWheelInput, rightWheelInput),
```

Once you have done this, you can test the sensors by downloading and running your program. After you have started the program, press the STOP button repeatedly to select the “Do Nothing” function. Then press the START button. Rotate the thumbwheel until the new screen appears, displaying values sampled from the sensors. Now turn one of the robot’s wheels slowly with your hand and observe how the sampled value changes as the spokes and holes pass by the sensor.

Copyright © 2005 RidgeSoft, LLC

Understanding Analog-to-Digital Conversion

The sample method uses the IntelliBrain controller's Analog-to-Digital converter to sample the current voltage at the analog signal pin. The sensor's output signal is on the white wire, which attaches to the signal pin, which is the third pin from the edge of the IntelliBrain controller board.

Each time your program invokes the sample method, the IntelliBrain controller's A-to-D converter samples the voltage on the signal pin. The A-to-D converter "converts" the analog voltage to a digital value. The sample method returns this value as an integer between 0 and 1023, inclusive. The digital value is proportional to the voltage with 0 corresponding to 0 volts and 1023 corresponding to 5 volts. Therefore, the sample method will return a value of 512 when the signal is at 2.5 volts. The A-to-D converter is not able to measure voltages outside of the 0 to 5 volt range.

Collecting and Analyzing Sensor Data

Your next step is to gain a better understanding of using the sensors to track motion of the wheels. You can do this by plotting a graph of the sensor output as the wheel turns under the power of the servo motor. You will need to add a new test function to your program that collects periodic samples of a sensor's signal and then prints the data so you can plot a graph of the signal.

Using RoboJDE, create a new class named "TestEncoder." This class will need to implement the Runnable interface, as follows:

```
public class TestEncoder implements Runnable
```

The Runnable interface requires that the class implement a method named run. This method will implement the data collection and reporting functionality discussed previously. Enter the following code for the run method:

```
public void run() {
    mServo.setPosition(100);
    try {
        Thread.sleep(500);
    } catch (InterruptedException e) {}

    int[] samples = new int[100];
    long nextTime = System.currentTimeMillis();
    for(int i = 0; i < samples.length; ++i) {
        samples[i] = mEncoderInput.sample();
        nextTime += 5;
        try {
            Thread.sleep(nextTime - System.currentTimeMillis());
        } catch (InterruptedException e) {}
    }

    mServo.setPosition(50);
    while (!mButton.isPressed());
    for (int i = 0; i < samples.length; ++i) {
```

Copyright © 2005 RidgeSoft, LLC

```
                System.out.println(
              Integer.toString(i * 5) + '\t' + samples[i]);
        }
    }
```

This method first applies full power to the servo motor. Then it instructs the executing thread to sleep for 500 milliseconds, giving the wheel a chance to accelerate to full speed. Following this it loops, repeatedly sampling the sensor signal and sleeping for 5 milliseconds. Once the sample array is full, after 100 samples, it stops the servo and waits for the START button to be pressed. After the START button is pressed the method will continue and execute a loop that prints the data to System.out, which outputs to both the LCD screen and the RoboJDE Run window.

In addition to the run method, the TestEncoder class will need a few member variables, a constructor and a toString method, as follows:

```
    private PushButton mButton;
    private Servo mServo;
    private AnalogInput mEncoderInput;

    public TestEncoder(AnalogInput encoderInput,
                       Servo servo,
                       PushButton button) {
        mEncoderInput = encoderInput;
        mServo = servo;
        mButton = button;
    }

    public String toString() {
        return "Test Encoder";
    }
```

This class will also need to declare several imports:

```
    import com.ridgesoft.robotics.AnalogInput;
    import com.ridgesoft.robotics.PushButton;
    import com.ridgesoft.robotics.Servo;
```

Once you have added this code, add a line just above the reference to the DoBeep function in the MyBot class to construct a TestEncoder, as follows:

```
    new TestEncoder(leftWheelInput, leftServo, startButton),
```

You will also need to add two lines to get the servo objects:

```
    Servo leftServo = IntelliBrain.getServo(1);
    Servo rightServo = IntelliBrain.getServo(2);
```

Place these lines further up in the main function, just after the lines referring to getAnalogInput.

Copyright © 2005 RidgeSoft, LLC

Finally, add an import statement for the Servo class:

```
import com.ridgesoft.robotics.Servo;
```

Now build the program and download it to your robot. Hold the robot in your hand or situate it such that the wheels are suspended above the table top. This will ensure your robot does not run off the table when the wheel starts turning.

Start the program running. Then select the "Test Encoder" function using the STOP button to toggle through the available functions and the START button to make your selection. When the wheel stops, press the START button. The samples your program collected will print to the RoboJDE Run window. Plot this data by hand or import it into a spreadsheet program to use your computer to plot it. Your graph should be similar to the chart shown in Figure 3.

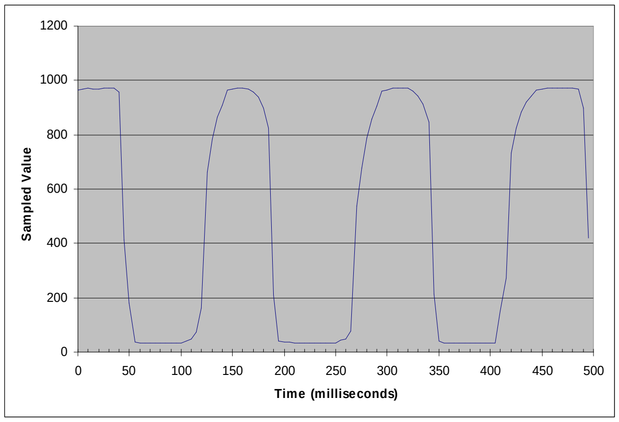

Figure 3 - Sampled Wheel Sensor Data

Recalling from the earlier discussion of the operation of the QRB1134 photo-reflector, the sampled value is low when a spoke is in front of the sensor and high when a hole in the wheel is in front of the sensor. The rising and falling edges of the signal correspond to the edges of the spokes.

By examining the graph, you will note the time between rising edges is roughly 150 milliseconds. This is the time for one full hole and one full spoke to pass in front of the sensor, which corresponds to one eighth of a revolution of the wheel. Multiplying by eight, it takes approximately 1.2 seconds for the wheel to rotate one revolution. Considering the servo was at full power and the wheel was spinning freely, this is the top speed of the wheel, approximately 50 revolutions per minute.

It is important to note there are twice as many spoke edges as there are spokes. Therefore, by sensing the edges of the spokes, rather than the spokes themselves, your program can sense the wheel position with twice the accuracy, measuring the wheel position to one sixteenth of a revolution. Your program can easily detect spoke edges by detecting the sensor signal transitions from low to high and high to low.

In order for your program to detect every spoke and every hole it must sample the sensor at least twice during the period it takes for a spoke and a hole to pass by the sensor. That is, two samples per one eighth turn of the wheel. If your program samples at a lower rate than this, it will miss some spokes or holes, making it unable to reliably track the position of the wheel. Therefore, since it

Copyright © 2005 RidgeSoft, LLC

takes 150 milliseconds for the wheel to rotate one eighth of a revolution at full speed, your program must sample the sensor at least once every 75 milliseconds. This will ensure it detects every spoke and every hole. However, there is a downside to sampling too frequently.

If your program samples too frequently, it will expend more computing power than is necessary to effectively track the position of the wheels, leaving less computing power available for other tasks that bolster your robot's intelligence.

Shaft Encoding

Now that you are familiar with interfacing your software with photo-reflector sensors, your next step is to implement a shaft encoder class that keeps track of the position of a wheel using one of these sensors.

"Shaft encoding" is a method of tracking the angular position and or/velocity of a rotating shaft, which in this case is the position of a wheel attached to the output shaft of a servo motor. Shaft encoders are very popular and widely used. They are commonly used in odometers, speedometers and tachometers on cars, motorcycles and bicycles. Anytime you need to sense the position or velocity of an axle or wheel you should consider the option of using a shaft encoder.

As its name suggests, a shaft encoder encodes the position of a shaft. Although, shaft encoding sounds sophisticated, it is, in fact, quite simple. The shaft encoder you will implement is simply a counter that counts up as the wheel rotates forward and counts down as the wheel rotates backwards. The counter records one count, or "click," each time the shaft rotates a fraction of a revolution. In the case of your robot, the passage of each spoke edge constitutes a "click" that your encoder will count.

An interesting form of an encoder that really does make a clicking sound is that of a playing card attached to a bicycle such that it clicks as the wheel turns. You may have added one of these "encoders" to your bicycle when you were a child. The playing card makes a clicking sound each time a spoke snaps past the card. The sound made by the card changes as the speed of the wheel changes. Just by listening to the clicking sound coming from this encoder you can tell how fast the wheel is turning.

Instead of using a playing card to generate clicks, your robot's shaft encoder uses a photo-reflector sensor to generate the electronic equivalent of clicks. The passage of each spoke edge results in a low to high or high to low signal transition, which constitutes a click of the encoder. Interestingly, if you attached the photo-reflector output to an amplifier and a speaker, you would hear clicking just like you do when a playing card snaps across the spokes of a bicycle wheel.

Copyright © 2005 RidgeSoft, LLC

Creating an AnalogShaftEncoder Class

You now have the basic information you need to implement your own shaft encoder class. Your program will need to use two instances of this class, one for each wheel on your robot.

You must implement the class such that it can predictably sample its sensor at least every 75 milliseconds, regardless of other tasks your program performs. This will be easy to accomplish by making use of Java's multi-threading capability. All you need to do is create a new class – AnalogShaftEncoder – which is a subclass of the Thread class. Each instance of this class will have its own thread to monitor the photo-reflector.

Go ahead and create the AnalogShaftEncoder class. Modify the class definition such that the AnalogShaftEncoder class extends the Thread class and implements two interfaces, ShaftEncoder and DirectionListener, as follows:

```
public class AnalogShaftEncoder extends Thread
        implements ShaftEncoder, DirectionListener
```

The ShaftEncoder interface is defined by RoboJDE. It is a good idea for your class to implement this interface. This will allow your new class to interoperate with other software that is written to use the ShaftEncoder interface. It will also provide you the option of easily modifying your program to use higher precision quadrature shaft encoders, such as Nubotics WheelWatcher WW-01 encoders.

The DirectionListener interface is a new interface that you must create. By implementing this interface, the AnalogShaftEncoder class will be able to "listen" for changes in the direction of the motor. As you extend your robot's control program, you will use this interface to inform the encoder of the direction the motor is being powered, forward or reverse. This will let the encoder know whether it should count up or count down. Use RoboJDE to create a new class named DirectionListener consisting of the following code:

```
public interface DirectionListener {
    public void updateDirection(boolean isForward);
}
```

The AnalogShaftEncoder class will need a number of member variables to manage its data. Add the following variables to the class:

```
private AnalogInput mInput;
private int mLowThreshold;
private int mHighThreshold;
private boolean mIsForward;
private int mCounts;
private int mPeriod;
```

This class will also need the following imports:

```
import com.ridgesoft.robotics.AnalogInput;
import com.ridgesoft.robotics.ShaftEncoder;
```

The DirectionListener interface requires that the implementing class, in this case AnalogShaftEncoder, implement the updateDirection method. Add the following method for this purpose:

```
public void updateDirection(boolean isForward) {
    mIsForward = isForward;
}
```

Similarly, the ShaftEncoder interface requires getCounts and getRate methods:

```
public int getRate() {
    return 0;     // rate calculation not supported
}

public int getCounts() {
    return mCounts;
}
```

Note: The full implementation of the getRate method is left as an exercise.

The run method of the AnalogShaftEncoder will need to loop forever, checking if the edge of a spoke has passed by the sensor since the previous check. After each check, the thread will need to sleep to allow other threads to execute. The longer the thread sleeps, the less CPU time it will consume, but if it sleeps too long it will miss spoke edges. It will also be a good idea to enclose all of the code in the run method in a try-catch block to catch and report any errors that occur. You don’t expect there to be any errors, but if there is a problem with your code it will be much easier to debug if it prints out a stack trace rather than just terminating the thread silently.

Use the following code for the overall structure of the run method:

```
public void run() {
    try {
        // take initial sensor sample
            :
        while (true) {
            // sample the sensor and count
            // spoke edges
                :
            Thread.sleep(mPeriod);
        }
    }
    catch (Throwable t) {
        t.printStackTrace();
    }
}
```

The first thing the method will need to do is take an initial sample of the sensor to determine if the signal is high or low, as follows:

```
boolean wasHigh = false;
if (mInput.sample() > mLowThreshold)
    wasHigh = true;
```

Add these lines below the take initial sample comment.

In order to count spoke edges, the run method will need to keep track of whether the sensor signal was high or low on the previous check. Each time the code detects a transition between low and high it will record a click by adjusting the counter. The following code implements this:

```
int value = mInput.sample();
if (wasHigh) {
    if (value < mLowThreshold) {
        if (mIsForward)
            mCounts++;
        else
            mCounts--;
        wasHigh = false;
    }
}
else {
    if (value > mHighThreshold) {
        if (mIsForward)
            mCounts++;
        else
            mCounts--;
        wasHigh = true;
    }
}
```

Add these lines in the while loop.

The low and high thresholds make the state of the wasHigh variable sticky – that is, they add hysteresis. The state will only change with large swings in the sampled value, making the encoder less susceptible to signal noise that would cause false counting.

Lastly, your AnalogShaftEncoder class will need a constructor to initialize new instances of the class when they are created:

```
public AnalogShaftEncoder(AnalogInput input,
                          int lowThreshold,
                          int highThreshold,
                          int period,
                          int threadPriority) {
    mInput = input;
    mLowThreshold = lowThreshold;
    mHighThreshold = highThreshold;
```

Copyright © 2005 RidgeSoft, LLC

```
        mPeriod = period;
        mIsForward = true;
        mCounts = 0;
        setPriority(threadPriority);
        setDaemon(true);
        start();
    }
```

Testing the Encoder

As a final step, you will need to add a screen that will allow you to view both encoder counters. This will permit you to test your encoders. Create the EncoderCountsScreen class to serve this purpose:

```
import com.ridgesoft.io.Display;
import com.ridgesoft.robotics.ShaftEncoder;

public class EncoderCountsScreen implements Screen {
    private ShaftEncoder mLeftEncoder;
    private ShaftEncoder mRightEncoder;

    public EncoderCountsScreen(ShaftEncoder leftEncoder,
                                ShaftEncoder rightEncoder) {
        mLeftEncoder = leftEncoder;
        mRightEncoder = rightEncoder;
    }

    public void update(Display display) {
            int leftCounts = mLeftEncoder.getCounts();
            int rightCounts = mRightEncoder.getCounts();
            display.print(0, "L enc: " + leftCounts);
            display.print(1, "R enc: " + rightCounts);
    }
}
```

You will also need to update your main class, MyBot, to put the new classes to use. Add the following lines to the main method just after the servo lines you added previously:

```
ShaftEncoder leftEncoder = new AnalogShaftEncoder(
                        leftWheelInput, 250, 750, 30,
                        Thread.MAX_PRIORITY);
ShaftEncoder rightEncoder = new AnalogShaftEncoder(
                        rightWheelInput, 250, 750, 30,
                        Thread.MAX_PRIORITY);
```

These lines create both encoders, initializing them with a low threshold of 250, a high threshold of 750, a sample period of 30 milliseconds and maximum thread priority. Setting the thread priority to the maximum will ensure other tasks don’t interfere with sampling the sensors.

Finally, add the EncoderCountsScreen to the screen list:

Copyright © 2005 RidgeSoft, LLC

```
      new EncoderCountsScreen(leftEncoder, rightEncoder),
```

You can now build, download and test your encoders. Once you have downloaded the program, select the “Do Nothing” function, start the program, turn the thumbwheel to select the EncoderCountsScreen and then gently turn one of your robot’s wheels with your hand. Notice the encoder counter will increase as you turn the wheel.

The counter will always increment regardless of the direction you turn the wheel. It will not decrement when you turn the wheel backwards. This happens because you are turning the wheel manually and, therefore, the AnalogEncoderClass has no way of knowing which way the wheel is turning. As you implement code to use the servo motors to power the wheels, you will use the updateDirection method to provide the AnalogEncoderClass with the information it needs to determine if it should count up or count down for each click.

Conclusion

You have now created a simple shaft encoder class that uses an infrared photo-reflector sensor to sense and count spoke edges as a wheel turns. With additional programming you can use two instances of this class to enable your robot to track its location and to navigate to specific locations.

Exercises

1. Using the WheelSensorScreen, observe the digital values sampled from both wheel sensors. Gently turn one of your robot’s wheels with your hand. Record the minimum and maximum digital value you observe from that wheel’s sensor. Also, make note if the sensor is adjacent to a hole or a spoke when you observe the minimum and maximum values. What are the signal voltages corresponding to these values?
2. Draw a graph showing the integer value the IntelliBrain controller’s A-to-D converter will return for signal voltages between 0 and 5 volts.
3. Draw a graph showing how you expect the digital value sampled from the sensor to vary over one revolution of the wheel. How many spokes will pass in front of the sensor? How many holes will pass in front of the sensor? How many spoke edges will pass in front of the sensor?
4. Modify the TestEncoder class to apply less power to the servo by specifying 55 instead of 100 for the position parameter when invoking the setPosition method. Collect and plot the data for both full power and the new power level. What effect does changing the power have? How many revolutions per minute does the wheel turn at each power level?
5. Add another test function to your program to make your robot move straight forward until both encoders have counted 100 counts.
6. Using the test function from the previous exercise, experiment with the sampling frequency of your encoders. What happens if you sample at too low a frequency? What happens if you sample at too high a frequency?

7. Complete the implementation of the AnalogShaftEncoder.getRate method such that is returns a meaningful value. One way to do this is to keep another counter that resets at a specified period. Just prior to resetting the counter, copy it to a member variable that holds the recent rate value. The getRate method can then return this variable as the measure of the rate. Hint: It will be easier to implement this if you chose to calculate the rate over a whole number of sample periods. This will allow you to use the existing timing mechanism incorporated in the run method.
8. Substitute WheelWatcher WW-01 quadrature shaft encoder sensors from Nubotics (www.nubotics.com) for the encoders you created. These encoders will provide greater precision than the encoders you created in this tutorial, improving the accuracy of your robot.

Copyright © 2005 by RidgeSoft, LLC. All rights reserved.

RidgeSoft™, RoboJDE™ and IntelliBrain™ are trademarks of RidgeSoft, LLC.

Java™ and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other brand or product names are trademarks of their respective owners.

RidgeSoft, LLC
PO Box 482
Pleasanton, CA 94566
www.ridgesoft.com