Patterns, frameworks & Lejos

Part 3: Proximity detection & RCX communications
Creating a simple proximity sensor

• Light sensor can detect both visible & infrared light
• We can use the RCX IR port as a sort of strobe light, and use the light sensor to pick up reflections from objects in the robot’s path
Using the IR port as a flashlight

• We can use the Serial class to transmit numbers via the IR port
• Sending a single number turns the light on for about 40 ms
• Strategy:
  – take sensor reading with IR “light” off
  – turn light on & take second reading
• If nothing is in front of the robot, light will dissipate into distance, and there will be little, if any difference between the readings; a large chip in reflected light would indicate an obstacle
import josx.platform.rcx.*;

public class IRTest extends Thread implements SensorConstants {
    private final byte[] packet = {127};

    public IRTest() {
        Sensor.S2.setTypeAndMode(SENSOR_TYPE_LIGHT, SENSOR_MODE_RAW);
        Sensor.S2.activate();
        Serial.setRangeLong();
        this.start();
    }

    public static void main (String[] args) {
        new IRTest();
    }
}
public void run () {
    int oldVal;
    int newVal;
    while (true) {
        oldVal = Sensor.S2.readValue();
        Serial.sendPacket(packet, 0, 1);
        try {
            Thread.sleep(5);
        } catch (Exception e) {}  
        newVal = Sensor.S2.readValue();
        int diff = Math.abs(oldVal - newVal);
        if(diff > 80) {
            Sound.playTone(diff, 20);
            LCD.showNumber(diff);
        }
        try {
            Thread.sleep(160);
        } catch (Exception e) {}
    }
}
Factors that affect readings

- Objects further away produce fewer “hits” than close objects.
- Readings are not always consistent, even at the same distance from the same object.
- Surface text-we affects the amount of reflected light—light-colored, smooth, hard surfaces give stronger readings than dark, soft, rough surfaces.
- Large objects produce stronger signals than small objects.
- Field of view limited; sensor can only detect what is directly in front of it.
ProximityListener interface

/**
 * @author Brian Bagnall
 * @version 2002
 */

public interface ProximityListener
{
    /**
     * Called when proximity sensor detects an object
     *
     * @param ping Strength of detected signal
     */
    public void objectDetected (int ping);
}
import josx.platform.rcx.*;
import java.util.*;

public class ProximityDetector extends Thread implements SensorConstants {
    private final byte [] packet = {127};

    private Sensor lightSensor;
    private Vector proximityListeners;

    public ProximityDetector(Sensor lightPort) {
        proximityListeners = new Vector(2,2);
        lightSensor = lightPort;
        lightSensor.setTypeAndMode(SENSOR_TYPE_LIGHT, SENSOR_MODE_RAW);
        lightSensor.activate();
        Serial.setRangeLong();
        this.start();
    }
public void run() {
    int oldVal;
    int newVal;
    while (true) {
        oldVal = Sensor.S2.readValue();
        Serial.sendPacket(packet, 0, 1);
        try {
            Thread.sleep(5);
        } catch (Exception e) {}
        try {
            Thread.sleep(5);
        } catch (Exception e) {}
        newVal = Sensor.S2.readValue();
        int diff = Math.abs(oldVal - newVal);
        if (diff > 80) {
            notifyListeners(diff);
        }
        try {
            Thread.sleep(160);
        } catch (Exception e) {}
    }
}
ProximityDetector continued

```java
public void addProximityListener (ProximityListener listener) {
    proximityListeners.addElement(listener);
}

private void notifyListeners (int ping) {
    for (int x=0; x<proximityListeners.size(); x++) {
        ProximityListener proxy = (ProximityListener) proximityListeners.elementAt(x);
        proxy.objectDetected(ping);
    }
}
```
import josx.platform.rcx.*;

public class ProxTest implements ProximityListener {
    public ProxTest() {
        Motor.A.forward();
        Motor.B.forward();
    }

    public static void main (String [] args) {
        ProximityDetector pd = new ProximityDetector(Sensor.S2);
        pd.addProximityListener(new ProxTest());
        try {
            Button.RUN.waitForPressAndRelease();
        } catch (Exception e) {} }
public synchronized void objectDetected (int ping) {
    Motor.A.backward();
    Motor.C.backward();
    try {
        Thread.sleep(500);
    } catch (Exception e) {}
    Motor.A.forward();
    try {
        Thread.sleep(700);
    } catch (Exception e) {}
    Motor.C.forward();
}
Reliability

- Sensor does very well when approaching white objects
- Has trouble with black objects, especially when approaching at an angle
- Have to play around with amount of change we want light sensor to detect
ProximitySensor class

• The functionality described in the last several slides has been encapsulated into the Lejos ProximitySensor class

• This class simplifies the use of the light sensor for proximity detection, just as the Navigator classes simplify navigation problems
RCX Communications

- Although IR can be used for proximity detection, its primary purpose is communication
- A robot can transmit data to the IR tower attached to the PC, and the PC can transmit via the IR tower to a robot
Because a robot’s memory is limited, complex algorithms should be performed, whenever possible, on a PC.

The robot can collect information about its environment & transmit the data to the PC for analysis.

The robot can then rely on guidance from the PC to solve complex problems.
RCX Communications

• Two RCXs can also communicate with one another via IR signals
  – Can create “super brick” with 6 inputs & outputs, 64K of memory - one brick is the controller, while the other just follows orders
  – Robots can also be built to interact with each other
Communications API

• Communications classes located in two Lejos packages:
  – java.io
  – josx.rcxcomm

• The Lejos java.io package is similar to the original Java API package; it contains classes that describe the most basic data I/o streams, including:
  – InputStream
  – OutputStream
  – DataInputStream
  – DataOutputStream