PRELIMINARY DRAFT NOTICE, June 26, 2003

JavelinBot is a manuscript with Boe-Bot projects designed for use the Javelin Stamp. JavelinBot files (manuscript, program listings, and a revised stamp.util.os package) are available for download from:

http://www.parallax.com/javelin/javelinbot.asp

- or -

http://groups.yahoo.com/group/javelinstamp/files/JavelinBot/

To attempt these projects, you can replace the BASIC Stamp 2 in your Boe-Bot with a Javelin Stamp. If you do not already have a Boe-Bot or Javelin Stamp, you can use the Javelin Stamp Starter Kit and Robotics Parts Kit together to create your own JavelinBot.

Please note that the preliminary draft material in the JavelinBot manuscript is much more advanced than what's in the Robotics! text for the Boe-Bot. It is also not supported by Parallax Technical Support. If you need assistance with the material in this manuscript, you can post questions to the JavelinStamp Yahoo Group:

http://groups.yahoo.com/group/javelinstamp/

To see a list of existing posts regarding the JavelinBot, use the search term:

JavelinBot

in the JavelinStamp Yahoo Group's *Search Archive* field.

COPYRIGHT, TRADEMARKS AND REPRODUCTION

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 of 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.

Chapter #7: Object Detection Using Infrared

Using Infrared Headlights to See the Road

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 Javelin can also use an infrared LED and detector to detect objects to the front and side of your traveling J-Bot.

Infrared

Infra means below, so Infra-red 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 infra-red	800-1000
Infra-red	1000-2000
Far infra-red	2000-10,000nm

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.

The J-Bot infrared object detection scheme has a variety of uses. The J-Bot can use infrared to detect objects without bumping into them. As with the photoresistors, infrared can be used to detect the difference between black and white for line following. Infrared can also be used to determine the distance of an object from the J-Bot. The J-Bot can use this information to follow objects at a fixed distance, or detect and avoid high ledges.

Infrared Headlights

The infrared object detection system we'll build on the J-Bot 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 J-Bot uses infrared LEDs for

headlights as shown in Figure 5.1. They emit infrared, and in some cases, the infrared reflects off objects, and bounces back in the direction of the J-Bot. The eyes of the J-Bot are the infrared detectors. The infrared detectors send signals to the Javelin indicating whether or not they detect infrared reflected off an object. The brain of the J-Bot, the Javelin, 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.

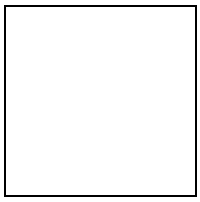


Figure 5.1: Object detection with IR Headlights.

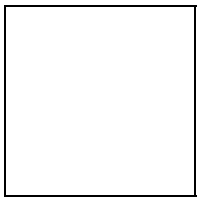
The Frequency Trick

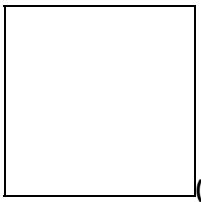
Since the IR detectors only see IR signals in the neighborhood of 38.5 kHz, the IR LEDs have to be flashed on and off at that frequency. The actual frequency will be 38.4 kHz since this is a frequency that the J-Bot can generate. A 555 timer can be used for this purpose, but the 555 timer circuit is 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 J-Bot enthusiasts found an interesting trick that made the 555 timer scheme unnecessary. This scheme uses the `PWM` object without the RC filter that's normally used to smooth the signal into a sine-wave. Even though the highest frequency `PWM` is designed to transmit is 57.6 kHz, the unfiltered `PWM` output can generate a 38.4 kHz signal with useful properties for a 38.5 kHz IR detector.

Those familiar with the Basic STAMP-based BOEBOT will know that the IR detection was done by varying frequency sent by the IR LED. The IR detector responds to the different frequencies based upon the distance to an obstacle. This approach will not work with the J-Bot because the Javelin STAMP cannot generate pulses fast enough or accurately enough. On the other hand, the J-Bot has a better way of doing things.

To start with, the Javelin has a digital-to-analog (DAC) virtual peripheral. This can generate voltages between 0 and 5 volts in a stepped fashion (see Fig. 5.2a). Attach an IR LED to the DAC output and the output intensity of the LED will vary depending upon the voltage. Combine the DAC/IR LED combination with a 38.4 kHz PWM output (see Fig. 5.2b) and the J-Bot can generate a modulated output of varying intensity (see Fig. 5.2c). Add in the IR detector and objects can be detected based on their distance from the IR LED/detector. An object that is very close to the J-Bot will be detected regardless of the amount of light being emitted by the LED. An object father away will only be detected when the light is more intense.





(b) PWM 38.4 kHz output

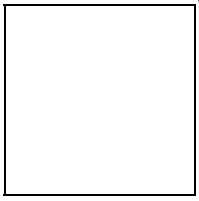


Figure 5.2: (a) Stepped DAC output **(c) Stepped 38.4 kHz output**

The DAC circuit uses a 1K Ω resistor and a 10µf capacitor. The resistor is connected to the DAC virtual peripheral output pin. The other end of the resistor is connected to the capacitor which is in turn connected to ground. The DAC virtual peripheral charges the capacitor to the desired voltage using pulses. The IR LED is connected to the resistor and capacitor. The other end of the LED is connected to the 220 Ω resistor which is connected to the PWM output pin. The circuit is shown in figure 5.3.

Activity #1: Building and Testing the New IR Transmitter/Detector

Parts

(2) Shrink wrapped IR LEDs
(2) IR detectors
(2) 220 Ω resistors
(2) 10µf capacitors
(2) 1K Ω resistors
(misc) wires

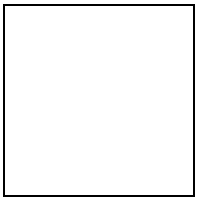


Figure 5.3: IR detector schematic symbol and part on top row and IR LED schematic symbol and part on bottom row.

Build It!

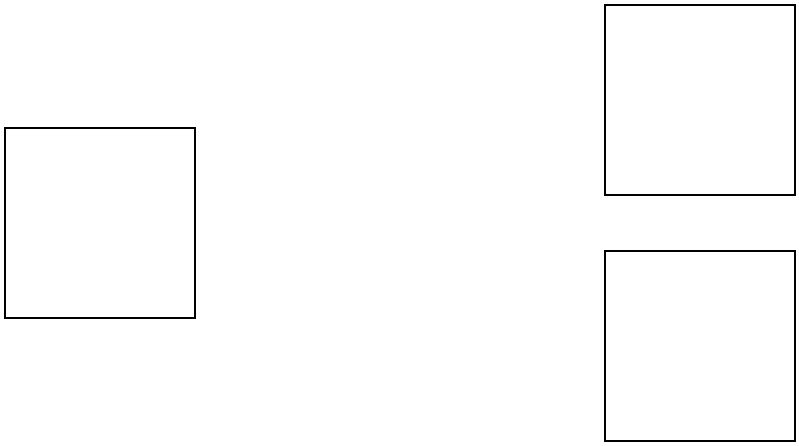


Figure 5.4: IR headlights (a) Schematic **(b) wiring diagram.**

Two circuits will be used by the IR obstacle detection object defined later in this chapter. The circuits are identical as in the prior chapter where a pair of photoresistors was used.

Testing the IR Pairs

The key to making each IR pair work is to modulate DAC output at 38.4 kHz. Only one IR LED and its matching detector will be used at a time to prevent interference with light from the other circuit.

- Enter and run the IrRangeTest1.
- This program makes use of the Message window, so leave the serial cable connected to the JSDB while the IrRangeTest1 program is running.

```
import stamp.core.*;
```

```
/**
 * IR Range Finder Test
 *
 * @version 1.0 9/7/02
 * @author Parallax, Inc.
 */

 public class IrRangeTest1 {
  static PWM pwmLeft  = new PWM(CPU.pin1,1,2);
  static DAC dacLeft  = new DAC(CPU.pin7);
  static PWM pwmRight = new PWM(CPU.pin1,1,2);
  static DAC dacRight = new DAC(CPU.pin7);

  public static int getRange ( DAC dac, PWM pwm ) {
    int range = 0 ;

    // Turn on virtual peripherals
    dac.start();
    pwm.start();

    // Step through light intensities
    for (int i = 0; i < 15; i++){
      // Set light level
      dac.update(255 - (13*i));

      // Allow detector to settle
      CPU.delay(200);

      // Count levels in range
      if(CPU.readPin(CPU.pin0)) {
        range++;
      }
    }

    // Turn off virtual peripherals
    dac.stop();
    pwm.stop();

    return range ;
  }

  public static void main() {
    pwmLeft  = new PWM(CPU.pin1,1,2);
    pwmLeft.stop();
    dacLeft  = new DAC(CPU.pin7);
    dacLeft.stop() ;
    pwmRight = new PWM(CPU.pin1,1,2);
    pwmRight.stop();
    dacRight = new DAC(CPU.pin7);
    dacRight.stop();

    while(true){
      System.out.print   ( "L " ) ;
      System.out.print   ( getRange ( dacLeft, pwmLeft )) ;
      System.out.print   ( "   R " ) ;
      System.out.println ( getRange ( dacRight, pwmRight )) ;
    }
  }

}
```

- While the program is running, point the IR detectors so nothing nearby could possibly reflect infrared back at the detectors. The best way to do this is to point the J-Bot up at the ceiling. The Message window output should display both left and right values as equal to "15."

- By placing your hand in front of an IR pair, it should cause the Message window display for that detector to change from "15" down to "0." Removing your hand should cause the output for that detector to return to a "15" state. This should work for each individual detector, and you also should be able to place your hand in front of both detectors and make both their outputs change from "15" to "0."

- If the IR Pairs passed all these tests, you're ready to move on; otherwise, check your program and circuit for errors.

How the IR Range Detection Program Works

The main method allocates the virtual peripherals but stops each before allocating the next virtual peripheral. This is because the constructor for the virtual peripherals starts the virtual peripheral.

> **FYI** The Javelin STAMP can only have 6 active virtual peripherals at one time. Stopping a virtual peripheral makes it inactive. Starting one makes it active.

The while loop in the main method repeats forever so it is best to run the program using the debugger. The method repeatedly prints out the range results for each DAC/PWM pair. The work determining the range is done by the getRange class method. We do not have to create our own objects at this point so we will stick with class methods.

The getRange method starts both virtual peripherals at the beginning of the method and stops both when the method is done. This means only two virtual peripherals will be active at one time. The for loop repeats sixteen times (0 to 15) and generates a voltage using the DAC. A dac.update value of 255 corresponds to 5 volts. This means the voltage will start at 5 volts and step down. The minimum value will be 60 (255 – (13*15)) or about 1 volt. The IR LED will generate any light if the value is below this point. Even at this point the amount of light is very small.

There is a small delay after the LED starts emitting the modulated signal. This allows the modulation to start since updating the PWM frequency does not cause the PWM object to change its frequency immediately. Likewise, the delay allows the IR detector to respond to the reflected infrared light.

The loop checks the IR detector response on each iteration. In theory, the transition from not detecting an obstacle to detecting one will occur consistently but in practice there are fluctuations in the signals from the IR detector. By counting the IR detector responses the results will typically vary only by one or two units.

Your Turn

- Experiment with different ranges instead of the 0 to 15 used in the example. Keep in mind that you must change the multiplier in the dac.update call. Does a finer granularity provide more information or do the results fluctuate too much to make a difference in accuracy?

- Try different color objects when testing the range capabilities. Do different colors or textures generate the same range results?

Activity #2: Detection Class – Infrared

The infrared sensor class uses the same BaseSensor class as the photoresistor example in the previous chapter. The task/sensor object architecture is repeated here to take advantage of the delays required for the DAC output voltage to settle and for the IR detector to recognize any reflected light from the LED. This infrared system actually works better with the multitasking system because virtual peripherals handle all the background operations and timing is not critical.

The following file shows IrRangeSensor class.

```
package JBot ;

import stamp.core.*;
import java.lang.Math.* ;

/**
 * IR Range Sensor Class
 * <p>
 * IR range sensors support
 *
 * @version 1.0 10/2/02
 * @author Parallax, Inc.
 */

public class IrRangeSensor extends BaseSensor {
  protected IrRangeSensorTask sensorTask ;

  protected int direction ;
  protected int distance ;
  protected boolean obstacleDetected = false ;
  protected int deadband ;

  static final int noObstacle = 15 ;

  /**
   * Create IR range sensor object and support task
   *
   * @param leftDacPin CPU.pin to use for DAC output
   * @param leftPwmPin CPU.pin to use for PWM output
   * @param rightDacPin CPU.pin to use for DAC output
   * @param rightPwmPin CPU.pin to use for PWM output
   * @param deadband deadband limit
   */
  public IrRangeSensor ( int leftDacPin
                       , int leftPwmPin
                       , int leftDetectorPin
                       , int rightDacPin
                       , int rightPwmPin
                       , int rightDetectorPin
                       , int deadband ) {
    sensorTask = new IrRangeSensorTask
                       ( this
                       , leftDacPin
                       , leftPwmPin
                       , leftDetectorPin
                       , rightDacPin
                       , rightPwmPin
                       , rightDetectorPin ) ;
    this.deadband = deadband ;
  }
```

```
/**
 * Indicate whether an obstacle has been detected.
 * Normally used when polling versus using an event.
 *
 * @returns obstacle detected
 */
public boolean obstacleDetected () {
  sensorTask.checkSensors() ;

  return obstacleDetected ;
}

/**
 * Indicate initial obstacle position.
 * For simple detection systems the detection of an object
 * on the right and left will return front.
 *
 * @returns obstacle's relative direction (left, right, etc.)
 */
public int obstacleDirection () {
  return direction ;
}

/**
 * Get the distance to an obstacle in the specified direction.
 * A value of <code>none</code> indicates no object detected.
 *
 * @param direction to get range for
 *
 * @returns distance to an obstacle for the specified direction
 */
public int obstacleDistance ( int direction ) {
  return distance ;
}

/**
 * Update results based on sensor information.
 * Called by sensor task when results available.
 *
 * @param resultLeft photoresistor rcTime result
 * @param resultRight photoresistor rcTime result
 */
protected void saveResults ( int resultLeft, int resultRight ) {
  // Current results are valid
  // Save obstacle status
  switch (   (( resultLeft < noObstacle ) ? 1 : 0 )
           + (( resultRight < noObstacle ) ? 2 : 0 )) {
  default:
  case 0:
    obstacleDetected = false ;
    break;

  case 1:
    direction = left ;
    distance = resultLeft ;
    obstacleDetected = true ;
    break;

  case 2:
    direction = right ;
    distance = resultRight ;
    obstacleDetected = true ;
```

```
      break;

    case 3:
      // Both sensors indicate an obstacle
      if ( Math.abs ( resultLeft - resultRight ) < deadband ) {
        // Both distances are close together
        direction = front ;
        distance = ( resultLeft > resultRight ) ? resultLeft : resultRight ;
      } else if ( resultLeft > resultRight ) {
        // Left sensor has a higher value
        direction = left ;
        distance = resultLeft ;
      } else {
        // Right sensor has a higher value
        direction = right ;
        distance = resultRight ;
      }
      obstacleDetected = true ;
      break;
    }

    notify();
  }
}
```


The IrRangeSensor class very similar to the PhotoresistorSensor class. The last obstacle detected information is maintained in object variables that are updated by the IrRangeSensorTask calling the sensor's saveResults method. The noObstacle constant definition is used to determine when no object is detected. The value is 15 which will be the distance returned by the task if no modulate IR light is detected by the IR detector. This also means that if an obstacle is detected then the range value will be between 0 and 14.

The range is in no particular units but a 0 distance means that an obstacle is very close or in contact with the J-Bot. As in the prior chapter, the deadband value is used to determine whether an object is in front of the J-Bot when the range values from the left and right detector are close.

The IrRangeSensor constructor creates and starts the IrRangeSensorTask. The IrRangeSensorTask constructor requires the six pins used for each pair of DACs, PWMs and IR detector inputs. Of course, the task needs the reference to the sensor object as well.

The following is the IrRangeSensorTask class definition.

```
package JBot ;


import stamp.core.*;
import stamp.util.os.* ;

/**
 * IR Range Sensor Class
 * <p>
 * Supports IrRangeSensor.
 * Should not be called directly by another other object.
 *
 * @version 1.0 10/2/02
 * @author Parallax, Inc.
 */

public class IrRangeSensorTask extends Task {
```

```
protected IrRangeSensor sensor ;

protected PWM leftPwm ;
protected DAC leftDac ;
protected int leftDetectorPin ;

protected PWM rightPwm ;
protected DAC rightDac ;
protected int rightDetectorPin ;

protected PWM pwm ;
protected DAC dac ;
protected int detectorPin ;

protected int iteration ;
protected int range ;
protected int leftResult ;

final static int startChecking = 1 ;
final static int checkLeftPin = 2 ;
final static int checkRightPin = 3 ;

protected IrRangeSensorTask
            ( IrRangeSensor sensor
            , int leftDacPin
            , int leftPwmPin
            , int leftDetectorPin
            , int rightDacPin
            , int rightPwmPin
            , int rightDetectorPin ) {
  this.sensor = sensor ;
  leftPwm  = new PWM(leftPwmPin,1,2);
  leftPwm.stop();
  leftDac  = new DAC(leftDacPin);
  leftDac.stop() ;
  this.leftDetectorPin = leftDetectorPin ;

  rightPwm = new PWM(rightPwmPin,1,2);
  rightPwm.stop();
  rightDac = new DAC(rightDacPin);
  rightDac.stop();
  this.rightDetectorPin = rightDetectorPin ;
}


/**
 * Check sensors if not already doing so
 */
protected void checkSensors() {
  if ( state == stopped ) {
    // Task was done. Start it again.
    nextState (startChecking) ;
    start () ;
  }
}

/**
 * Change voltage and setup delay
 */
protected void changeVoltage () {
  dac.update(255 - (13*iteration));
  sleep(1100,state);
}
```

```
  /**
   * Start IR sensor operation.
   * Saves DAC, PWM and detector pin for use by checkRange and changeVoltage.
   * Resets iteration and range counters.
   * Starts PWM and DAC virtual peripherals
   * Setup initial voltage and delay
   *
   * @param dac DAC virtual peripheral
   * @param pwm PWM virtual peripheral
   * @param detectorPin pin connected to IR detector output
   * @param nextState next state after delay
   */
  protected void startPulse ( DAC dac, PWM pwm, int detectorPin, int nextState
) {
    iteration = 0 ;
    range = 0 ;

    this.dac = dac ;
    this.pwm = pwm ;
    this.detectorPin = detectorPin ;

    dac.start();
    pwm.start();
    nextState ( nextState ) ;
    changeVoltage () ;
  }

  /**
   * Check IR detector.
   * Change voltage and delay if not done checking.
   *
   * @result true if done checking
   */
  protected boolean checkRange () {
    if ( CPU.readPin ( detectorPin )) {
      ++ range ;
    }

    if ( iteration == 15 ) {
      // Done checking. Turn everything off
      pwm.stop();
      dac.stop();
      return true ;
    } else {
      // Update counter, change voltage and set delay
      ++ iteration ;
      changeVoltage () ;
      return false ;
    }
  }

  /**
   * Multitasking support
   */
  public void execute () {
    switch ( state ) {
    case startChecking:
      // Start left DAC and PWM and then delay
      startPulse ( leftDac, leftPwm, leftDetectorPin, checkLeftPin );
      break;

    case checkLeftPin:
```

```
      if ( checkRange ()) {
        // Save results
        leftResult = range ;

        // Start right DAC and PWM and then delay
        startPulse ( rightDac, rightPwm, rightDetectorPin, checkRightPin );
      }
      break;

    case checkRightPin:
      if ( ! checkRange ()) {
        // Delay setup, exit method
        break;
      }

      // Save results and stop task
      sensor.saveResults ( leftResult, range ) ;

    default:
    case initialState:
      stop() ;
      break;
    }
  }
}
```


The IrRangeSensorTask constructor allocates the virtual peripherals in the same fashion as the prior section so the DAC and PWM objects are stopped. They will be restarted as needed.

The dac, pwm and detectorPin variables will contain the currently active DAC, PWM, and detector pin values since these are maintained while the task checks the range for one side or the other. These variables are used by the changeVoltage, startPulse and checkRange methods. If the object variables were not used then these methods would need a corresponding set of parameters.

Skip to the execute method. This has a structure similar to the PhotoresistorSensorTask in the last chapter. The initialState stops the task which will be restarted by the IrRangeSensor the checkSensors method. The startChecking state will be entered when the checkSensors method restarts the task. This state calls the startPulse method that clears the range and iteration counters, sets the dac, pwm and detectorPin variables, starts the PWM and DAC objects, sets the DAC voltage and will cause the task to sleep until things have stabilized. When this method returns the virtual peripherals are configured to send modulated IR light via the IR LED.

The checkLeftPin state will be entered when the task is done sleeping. The task remains in the checkLeftPin state and calls the checkRange method repeatedly until the IR LED has been operated in all 16 voltage levels. The checkRange method checks the IR detector pin and updates the range counter if light is detected. The iteration variable is incremented. The checkRange method will return true when all iterations have been performed and the DAC and PWM have been turned off. This result is used to determine when the next side is to be checked. The startPulse method for the right side is called in the execute method at this point after the range value is saved.

The process is repeated for the right side. Note that the checkRightPin state uses the checkRange method but the layout is slightly different to allow the initialState's stop task method call to do double duty and stop the task after checkRange returns false. At this point the sensor's saveResults method is called using the saved leftResult and the current range from the right side.

Testing the IR range sensor object is relatively easy using the following program.

```
import stamp.core.*;
import stamp.util.os.* ;

import JBot.*;

/**
 * IR Range Sensor Test
 *
 * @version 1.0 9/7/02
 * @author Parallax, Inc.
 */

public class IrRangeSensorTest1 extends Task {
  protected IrRangeSensor sensor =
    new IrRangeSensor ( CPU.pin7    // left DAC pin
                      , CPU.pin1    // left PWM pin
                      , CPU.pin0    // left detector pin
                      , CPU.pin8    // right DAC pin
                      , CPU.pin3    // right PWM pin
                      , CPU.pin2    // right detector pin
                      , 2           // deadband
                      ) ;

  public void execute() {
    if ( sensor.obstacleDetected ()) {
      int direction = sensor.obstacleDirection () ;

      System.out.print   ( "Dir=" ) ;
      System.out.print   ( direction ) ;
      System.out.print   ( "   Range=" ) ;
      System.out.println ( sensor.obstacleDistance (direction)) ;
    } else {
      System.out.println ( "none" ) ;
    }

    sleep(state,200);
  }


  public static void main() {
    new IrRangeSensorTest1 () ;

    Task.TaskManager();
    System.out.println ( "Should not have terminated!" ) ;
  }

}
```

The test program creates a task that in turns creates an IR sensor with its task. The IrRangeSensorTest1 execute method simply polls the sensor and prints the current status. If the information is being displayed too quickly then an additional state can be added and the task can sleep before or after the status is printed.

Activity #3: Object Detection and Avoidance

An interesting thing about the IR detectors is that their outputs are just like the whiskers. The main difference is that the whiskers only indicate contact with an obstacle whereas the IR range finder determines distance allowing the J-Bot to avoid obstacles before coming in contact with them.

Converting the Whiskers Program For IR Object Detection/Avoidance

Changing the whisker obstacle avoidance program, AvoidObstacleTaskWhiskerTest1, to work with the IrRangeSensor is extremely easy since all the work is already handled by the AvoidObstacleTask. This object takes a sensor object as a constructor parameter. It is a matter of filling in the blanks to make things work.


```
import stamp.core.*;
import stamp.util.os.* ;
import JBot.* ;

/**
 * Test AvoidObjstacleTask class
 * <p>
 * Tun the J-Bot so it avoids obstacles.
 *
 * @version 1.0 7/23/02
 * @author Parallax Inc.
 */

public class AvoidObstacleTaskIrRangeTest1 {
  public static void main () {
    new AvoidObstacleTask
      ( new IrRangeSensor ( CPU.pin7    // left DAC pin
                          , CPU.pin1    // left PWM pin
                          , CPU.pin0    // left detector pin
                          , CPU.pin8    // right DAC pin
                          , CPU.pin3    // right PWM pin
                          , CPU.pin2    // right detector pin
                          , 1           // deadband
                          )
      , new RampingJBot ( new MultitaskingJBot ())) ;

    Task.TaskManager () ;
    System.out.println ( "All done" ) ;
  }
}
```

How the Roaming with Whiskers Adjusted for IR Pairs Program Works

Actually there is not much to this change. The AvoidObstacleTask starts up with the newly created IR range sensor object. This is polled to determine when an object has been detected otherwise the J-Bot continues moving forward. The task already handles all the movement control.

It is possible to adjust the deadband value. Setting the value to 0 will mean that the J-Bot will only backup if it detects an object directly in front. That is, both sensors return the same range. Increasing the value to 2 makes the J-Bot a little less sensitive so head on detection is more forgiving.

Activity #4: The Drop-off Detector

The IR detector and LEDs were aimed parallel to the floor. This allows the J-Bot to detect objects directly in front and slightly to the sides. Aiming these down towards the floor allows the J-Bot to determine when there is a drop-off such as the edge of a table.

There are a number of approaches that can be used to replicate the operation of the AvoidObstacleTask in avoiding obstacles to avoiding drop-offs. One is to come up with a new class like AvoidObstacleTask that works in a slightly different way. It moves forward while an obstacle is detected in front.

Another method is to come up with a new sensor object that indicates a drop off as an obstacle. We take this approach because changing the AvoidObstacleTask would be a little more difficult. Also, creating the new sensor object is simply a matter of extending the IrRangeSensor object.


```
package JBot ;

import stamp.core.*;
import java.lang.Math.* ;

/**
 * IR Range Sensor Class that detects drop offs
 * <p>
 * IR range sensors support
 *
 * @version 1.0 10/2/02
 * @author Parallax, Inc.
 */

public class IrRangeDropOffSensor extends IrRangeSensor {
  private int direction ;
  private int threshold ;

  /**
   * Create IR range sensor object and support task
   *
   * @param leftDacPin CPU.pin to use for DAC output
   * @param leftPwmPin CPU.pin to use for PWM output
   * @param rightDacPin CPU.pin to use for DAC output
   * @param rightPwmPin CPU.pin to use for PWM output
   * @param deadband deadband limit
   */
  public IrRangeDropOffSensor
                       ( int leftDacPin
                       , int leftPwmPin
                       , int leftDetectorPin
                       , int rightDacPin
                       , int rightPwmPin
                       , int rightDetectorPin
                       , int deadband ) {
    super ( leftDacPin
          , leftPwmPin
          , leftDetectorPin
          , rightDacPin
          , rightPwmPin
          , rightDetectorPin
          , deadband ) ;

    // Assumes J-Bot is not looking at a drop off

    if ( super.obstacleDetected ()) {
      threshold = 10 +
        super.obstacleDistance ( super.obstacleDirection ()) ;
    } else {
```

```
    // Use default otherwise

    threshold = 20 ;
  }
}


/**
 * Indicate whether an obstacle has been detected.
 * Normally used when polling versus using an event.
 *
 * @returns obstacle detected
 */
public boolean obstacleDetected () {
  int distance ;

  // obstacle status is the opposite of the normal sensor

  if ( super.obstacleDetected ()) {
    direction = super.obstacleDirection () ;
    distance  = super.obstacleDistance ( direction ) ;

    // Distance is less than the threshold if drop off not detected
    if ( distance > threshold ) {
      // Obstacle direction is the opposite when looking down
      switch ( direction ) {
      case left:
        direction = right ;
        break;

      case right:
        direction = left ;
        break;

      default:
        direction = front ;
        break;
      }

      return true ;
    }
  }

  return false ;
}

/**
 * Indicate initial obstacle position.
 * For simple detection systems the detection of an object
 * on the right and left will return front.
 *
 * @returns obstacle's relative direction (left, right, etc.)
 */
public int obstacleDirection () {
  return direction ;
}

/**
 * Get the distance to an obstacle in the specified direction.
 * A value of <code>none</code> indicates no object detected.
 *
 * @param direction to get range for
 *
```

```
   * @returns distance to an obstacle for the specified direction
   */
  public int obstacleDistance ( int direction ) {
    return 0 ;
  }
}
```

Combine this object with the usual AvoidObstacleTask object as in the following program and everything works.


```
import stamp.core.*;
import stamp.util.os.* ;
import JBot.* ;

/**
 * Test AvoidObjstacleTask class
 * <p>
 * Tun the J-Bot so it avoids obstacles.
 *
 * @version 1.0 7/23/02
 * @author Parallax Inc.
 */

public class AvoidDropOffTaskIrRangeTest1 {
  public static void main () {
    new AvoidObstacleTask
      ( new IrRangeDropOffSensor
                      ( CPU.pin7    // left DAC pin
                      , CPU.pin1    // left PWM pin
                      , CPU.pin0    // left detector pin
                      , CPU.pin8    // right DAC pin
                      , CPU.pin3    // right PWM pin
                      , CPU.pin2    // right detector pin
                      , 1 )           // deadband
      , new RampingJBot ( new MultitaskingJBot ())) ;

    Task.TaskManager () ;
    System.out.println ( "All done" ) ;
  }
}
```

The AvoidObstacleTask object uses the new IrRangeDropOffSensor object. The sensor indicates it has detected an obstacle when the IrRangeSensor detects nothing. This will occur when a drop-off is located.

Your Turn

- The approach presented does not detect minor vertical differences that would occur if the J-Bot is running on a table that has a lower ledge around its perimeter. How could this be handled? Hint: Write a new class to replace the AvoidObstacleTask class that checks for changes in distance.

- Using the new class, check for both drop-offs and obstacles. Hint: drop offs will occur when the change in distance is positive. Obstacles can be detected when the change is negative.

Summary and Applications

This chapter covered a unique technique for infrared object detection. By shining infrared light into the J-Bot's path and looking for its reflection, object detection can be accomplished. Infrared LED circuits are used to send a 38.4 kHz signal by using a DAC and PWM virtual peripheral objects.

Building on the BaseSensor class, the IrRangeSensor class was created. This was used in conjunction with the existing AvoidObstacleTask to allow the J-Bot to navigate around obstacles.

By tilting the IR LED and detectors toward the floor the J-Bot can detect drop-offs such as the edge of a table. This was accomplished by extending the IrRangeSensor in a novel fashion.

Real World Example

Infrared is one of the more popular amenities on electronic products. TV remotes, palmtop computers, and fancy calculators all use infrared for communication. A variety of communication schemes exist for transmitting data. A TV remote control, for example, sends a high signal by flashing its IR transmitter at 38.5 kHz. A low signal is no IR. The detectors in some TVs, VCRs, etc. are identical to the receiver used in the J-Bot.

The detection scheme in the automatic door openers common in convenience and grocery stores relies on the same theory of operation for object detection used by the J-Bot. Whenever you trigger one of these door openers, it's because you walked into and broke the IR beam being reflected back at the receiver. Infrared detectors also are mounted on many different conveyer belts. Factories use them to count products as they fly by, and grocery stores use them to detect when the groceries have reached the end of the conveyer belt. In grocery stores, these belts automatically move the groceries forward so the checker can reach them. To prevent the conveyer belt from piling groceries on the scanner, an IR detector is mounted at the end of the conveyer belt. When the Twix candy bar interrupts the IR beam shining across the conveyer belt, the IR detector's output changes. When this change is detected by a microcontroller, it stops the motor that runs the conveyer belt.

J-Bot Application

The unique thing about IR detectors is that they allow the J-Bot to detect objects without actually touching them. In maze competitions where you lose points by touching the walls, this is a real plus.

Questions and Projects

Questions

1. What does infrared mean? How does infrared differ from near infrared?
2. What are the two kinds of filters built into the IR detectors in the J-Bot Kit? What does each do?
3. Describe what each of the two IR detector outputs mean.

4. Why is a DAC and a PWM virtual peripheral used to build an infrared range finder?

Exercises

1. Modify IrRangeSensor so that the right IR pair is checked before the left pair.

2. Modify the AvoidObstacleTask so that it makes the J-Bot follow objects instead of avoiding them. Describe the problems you encounter, if any.

Projects

1. The IR range finder sensor object is not calibrated in terms of inches.

   - Create a program that translates the 0-15 values from the infrared detection system so it returns results in terms of inches.

   - Modify the obstacleDistance method so it returns values in inches.

2. One of the shortcomings of IR object detection is that the J-Bot's IR detectors do not detect black. That's because black absorbs IR instead of reflecting it. The J-Bot tends to run into black objects when roaming with IR because it doesn't see them. Add whiskers to your breadboard and create a sensor class that extends BaseSensor and uses an IrRangeSensor object and a WhiskerSensor object. Hint: the new sensor class object methods will check the whisker sensor first and then the IrRangeSensor.

Remember: Wrap the portions of the whiskers with electrical tape that could come into contact with circuits other than the whisker contact posts.