Massachusetts Institute of Technology

Robotics: Science and Systems I
Lab 3: Light Sensors and Braitenberg Behaviors

Distributed: Wednesday, 16 February 2011 at 3pm

Wiki Materials and Briefings Due: Tuesday, 22 February 2011 at 3pm

(Note: Tuesday, Feb 22 is a virtual Monday; we will have class and lab that day.)

Objectives and Lab Overview

Your objective in this lab is to understand how to integrate and use sensors on your robots. You will add two light sensors to your robot. You will mount the sensors and calibrate them. Then you will use the sensors to control the robot's movement according to the light levels it senses from them in accordance with the historic Braitenberg behaviors.

Self Assessment: On your team wiki page (see § 1 below), before you start the lab, each team member should rate his or her proficiency in each listed area **as of the start of this lab** on the given skill point chart for Lab 2 in the ”Beginning of Lab” column (1=Not at all proficient; 2=slightly proficient; 3=reasonably proficient; 4=very proficient; 5=expert):

- **Mechanics**: How proficient are you at mechanical assembly?
- **Electronics**: How proficient are you at soldering and working with electronics?
- **Version Control**: How comfortable are you using a Version Control system?
- **Java Coding**: How proficient are you at programming in Java?
- **Motion Control**: How proficient are you at designing and implementing reactive robot behaviors?

When filling this out, be creative, and be honest – your answers will help you progress through the material (knowing your strengths and weaknesses is crucial to effective teamwork), and will aid the class staff in providing support for your work in the class. Your self-assessments will **not** affect your grade in RSS.

Additionally, each team member should update his or her saved progress on their wiki areas, reflecting all time spent on the lab. Make both of these actions a habit (allocating skill points and updating time spent in lab), so the staff will have a better handle on how best to support your team as you progress through the labs and the challenge itself.

To start the lab, you should have:

- Your robot from previous lab
- Two light sensors, wire, connectors, heat shrink tubing, heat gun (shared), soldering iron and solder
- A photocopied handout on Braitenberg vehicles from the book entitled ”Vehicles: Experiments in Synthetic Psychology” by Valentino Braitenberg.

Check for any errata for the lab in the course wiki, and for any additional materials linked from the lab handouts webpage.

Check the Lab Handouts area of the RSS wiki for errata and additional materials before proceeding.

Physical Units

We remind you to use MKS units (meters, kilograms, seconds, radians, watts, etc.) throughout the course and this lab. In particular, this means that *whenever you state a physical quantity, you must state its units*. Also show units in your intermediate calculations.

1 Establish Your Wiki Area for this Lab

Each RSS Lab requires a team-authored set of wiki materials. Create space for these in your team's Lab Materials area. Title the page with the lab name and use the Wiki formatting syntax to post and organize any question responses, images, video clips etc.

As you work through the lab, add text and upload screenshots and photos to the wiki. In particular, be sure that *every italicized question and deliverable throughout the lab is answered on the wiki*.

2 Incorporating the Sensors

Recall from the lecture on sensors that a light detecting sensor is incorporated using a voltage divider. The voltage at the analog input pin of your μORCboard is determined by the voltage between the resistor and the photocell.

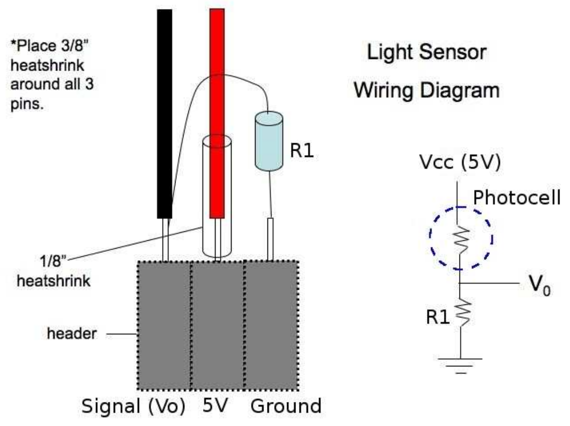

Figure 1: Wiring diagram of the light sensor.

1. Consult the wiring of the light sensor on the exemplar, and Figure 1.

   Write down the formula for the voltage divider which gives V_{out} in terms of $V_{in} = 5\mathrm{V}$, the photocell resistance R_{photo}, and the resistor R_1. If your photocell has a resistance of 180 Ohms in bright light and 200k Ohms in the shade, what is the voltage range of your light sensor as a function of R_1?

2. Suppose your photocell has an average resistance around 47k Ohm in ambient light. You can measure the real range of resistance of your photo cell by connecting it to your multimeter set to the ohm meter setting.

   What is a desirable value for the resistor?

3. Suppose you use a resistor an order of magnitude more resistive than your answer above.

   What will be the effect? What can you do to compensate?

4. Using Figure 1 and the exemplar robot as guides, build and mount two light sensors to your robot's frame. For each sensor, connect the 3-pin female header to either channel 0 or 1 of the ADC ports on the μORCboard. Be careful to observe the polarity of the connector as well as the male header on the μORCBoard. If you have questions, consult the exemplar robot.

5. Use ORCspy to examine the sensor measurements. *What range of values do you see during an interval of ambient light? What range of values do you observe as you shine a light source in front of a sensor and then close the interval (with roughly discrete steps in distance) between the light source and the sensor?* Cover the sensor with your palm. *What range of values do you observe now? Record these values on the wiki. Are they linear?*

Deliverables: Add a picture of your completed sensors to the wiki. Answer the questions above. Please include a description of how you mounted the sensors and let us know what was easy and what was difficult.

3 Populate your Working Area with the Scaffolding Code

Follow these steps to get the scaffolding code for the lab (refer to the Subversion source code control handout as needed):

1. Wherever you choose to develop your code, update the scaffolding code in your working copy of the RSS-I-pub repository.

   (a) Move to the RSS-I-pub directory:

   ```
   cd ~/RSS-I-pub
   ```

   (b) Update the working copy of the RSS-I-pub repository:

   ```
   svn update
   ```

2. Add the newly updated Braitenberg code to your group's repository by executing the following four steps:

   (a) Go to your working copy of your group's repository:

   ```
   cd ~/RSS-I-group/trunk
   ```

   (b) Copy the Braitenberg source into your working copy of your group's repository, using the svn "export" utility:

   ```
   svn export ~/RSS-I-pub/labs/Braitenberg/
   ```

   (c) Next, use the `svn add` command to put the Braitenberg directory and its contents under svn control:

   ```
   svn add Braitenberg
   ```

   (d) Finally, commit the changes in your local working copy of the repository to your group's repository on the svn server:

   ```
   svn commit -m "(Your commit comment here)"
   ```

3. Import the new Braitenberg files into Eclipse

   (a) Start Eclipse and from the menu bar select *File* → *New* → *Project*. In the New Project dialog that appears, select "Java Project from Existing Ant Buildfile" and click the Next button.

   (b) In the New Java Project dialog, click the "Browse…" button beside the "Ant buildfile" text box. Browse to `/home/rss-student/RSS-I-group/trunk/Braitenberg` and select the "build.xml" file.

   (c) Back in the New Java Project dialog, change the name of your project to something more descriptive than "build" and choose `"javac" task found in target "compile [default]"` as the javac declaration used to defined the project. Also, be sure to check the "Link to the Builfile in the file system" checkbox. When you're done, click the "Finished" button.

   (d) If, after importing all of the source files, Eclipse complains about `RobotBase` not being found, you need to tell Eclipse to add motorControl.jar file to its build path. From the context menu that results from Right-clicking on your Braitenberg project in the Project Explorer view, select *Build Path* → *Add External Archives*. Then select "motorControl.jar" in the `Braitenberg` directory.

   (e) The scaffolding files are designed to use the published solution code from the Motor Control lab. If you would rather use your own motor driver code, change all occurances of `import MotorControlSolution.*;` in Behavior.java, LightSensors.java, and Photocell.java to `import MotorControl.*;`. Then, in the project Properties dialog, (accessed from the context menu that results from Right-clicking on your Braitenberg project in the Project Explorer view and selecting Properties), select "Java Build Path", activate the "Projects" tab, click the "Add" button, and then select your MotorControl project.

4. Once you have completed your code, you'll need to run it by following these steps:

   (a) In the Project Explorer view, right-click on the build.xml file to bring up its context menu. Then select *Run As → Ant Build...* being sure to select "Ant Build..." not "Ant Build".

   (b) In the "Check Targets to Execute" listbox, ensure that both the "compile" and "run" checkboxes are selected. Then, click the Apply button followed by the Run button.

4 Calibrating the Sensors

It is important to calibrate sensors relative to ambient light because ambient light affects your sensor readings. Ambient light varies from room to room or even in the same room at different times of the day when you have different lighting conditions. Ideally, you would like for your robot to adapt to ambient light.

To keep things simple, in this lab your robot should run a calibration method that explicitly senses the ambient light upon startup and sets an offset term and a linear scale term. Later, another method will use these stored values to return a calibrated reading from the light sensors at run-time.

1. Fill in the `calibrate()` method in `Photocell.java` to set the instance variables `offset` and `scale`. Have your method take multiple sensor measurements of the ambient light over a duration of at least 5 seconds (the frequency is up to you). A method named `getRawValue()` is already provided in `Photocell.java` which returns the raw value of the sensor.

   Assume that 5 volts is always the maximum reading, and that it corresponds to saturation of your sensor. Assume that the light sensor readings scale linearly with the light level. Make your calibrated values range from 0 to 100 . That is, 0 should indicate ambient light and 100 should indicate saturation (bright light).

   Based on the sensor behavior you saw in Part 1 via the ORCspy, is a linear scaling sufficient? What other scaling could be done to improve the information from the sensors?

2. Fill in the `getValue()` method in `Photocell.java` which returns the calibrated value of the light sensor, based on its current reading, `offset`, and `scale`.

Deliverables: Answer the question regarding sufficiency of linear scaling.

5 Searching for a Light Source

In this part of the lab you will use the light sensors to guide the motion of your robot toward a **fixed** (i.e. non-moving) light source. Ultimately, your robot will search for a light source, head toward it, and stop and beep once it arrives.

Hint 1: Use a bright light source, and hold it at the same height as the light sensors. A bare 75W incandescent light bulb works well, but is of course **hot** – be careful not to touch it or let it touch your robot.

Hint 2: During the daytime, sunlight may enter the room through the windows. How can you make your calibration procedure reasonably robust even when some sunlight is present?

1. In `Behavior.java`, implement `search()` to find the direction of a bright light source (note that a convenience `Behavior.setDesiredAngularVelocity()` method has been provided). It is up to you to define what is considered a "bright" light. The `search()` function is called repeatedly (from `Behavior.go()`) until it returns `true`. *How far away can the light source be for the robot to detect it in your implementation?* If the robot does not see any light at the start (that is, it is not directly pointed a the light source), it should search for it by rotating. *What will your robot do when there are two light sources? Or when a single light source is equally spaced from both light sensors?* Test your algorithm for three different distances and three different orientations of the light source; describe and sketch the result of each experiment (you can upload sketches to the wiki by photographing them with your PDA or the RSS camera).

2. Implement `Behavior.goToLight()` which is invoked whenever `search()` returns `true`. After the initial `search()`, the robot should not make any more turns. Make the speed of the robot decrease in proportion to its

distance from the light source, and beep (a `beep()` method is provided in `Behavior.java`) when the robot is close to the light source (again, by your definition).

Deliverables: Document your methods and prepare a written explanation of the algorithms for the locate and tracking methods. Include your test data (screen shots of runs work well) and answers to the questions above.

6 Braitenberg Behaviors

But emotions don't seem like a very useful simulation for a robot... I don't want my toaster or my vacuum cleaner appearing emotional.
— Detective Spooner, *I, Robot*

In his book entitled *Vehicles: Experiments in Synthetic Psychology*, Valentino Braitenberg describes a series of thought experiments in which vehicles with simple internal structure behave in unexpected ways. He experimented with simple control mechanisms that generate complex behaviors with psychological interpretations such as aggression, love, foresight, and even optimism. All these behaviors have very simple internal representation and it is surprising that the resulting perceived response of the robot can have such complex manifestation. In his book, Braitenberg gives these machines as evidence for the *law of uphill analysis and downhill invention* which refers to how it is more difficult to guess internal structure of a creature from the observation of behavior than it is to create the structure that gives the behavior.

The simplest Braitenberg vehicle is equipped with one sensor and one motor. This is described in the Braitenberg handout. We shall skip implementing this vehicle but it is worth noting the vehicle responds linearly to sensory input. Your next tasks involve implementing several Braitenberg behaviors.

1. Implement `Behavior.vehicle2a()` and invoke it from the appropriate spot in `Behavior.go()`. Connect the behavior of each light sensor to the motor on the same side. When the sensor gets closer to a source, its corresponding motors moves more quickly. As the sensor moves away from a light source, the corresponding motor slows down. This is termed excitatory response. Test your program and record how your robot responds to fixed light sources in 3 different locations, as well as its response to a moving light source. *What emotion does your robot's behavior most resemble?*
2. Implement `Behavior.vehicle2b()`. Connect the behavior sensor to the motor on the opposite side of the robot. *Write your prediction on what will happen under different light source positions and test the robot's response in 3 different example settings, plus in a setting with a moving light source. How do your predictions match the robot's response? What emotion does your robot's behavior most resemble?*
3. Implement `Behavior.vehicle3a()` and `Behavior.vehicle3b()`. You can implement Vehicle 3a by generalizing your code for Vehicle 2a, making the connections inhibitory rather than excitatory. That is, when the sensor gets closer to a source its corresponding motor should slow down, and when the sensor gets farther from a source its corresponding motor should speed up. *Repeat your predictions and experiments recording them.* Likewise, for Vehicle 3b, generalize your code for Vehicle 2b, making the connections inhibitory rather than excitatory. *Write your prediction on what will happen under different light source positions and test the robot's response in 3 different example settings, plus in a setting with a moving light source. How do your predictions match the robot's response? What is the difference between the two vehicles' behavior under the same light source conditions? What emotions best describe your robot's behaviors?*
4. Design and implement your own vehicle behavior. For example, consider the Wimpy Robot behavior: your robot gets attracted by a light source, investigates with great curiosity for a while, then gets scared, turns around, and runs in the opposite direction as far and as fast as it can! Some other options include varying excitation and inhibition or varying the sensor response curve to create new behaviors. You can call the function you implement in the `Behaviors.CREATIVE` section of `Behavior.go()`. *Repeat your predictions and experiments recording them on your wiki. Select your favorite and describe the algorithm and observed results briefly on the wiki.*

Deliverables: On the wiki, post responses to each of the questions above, and a short video clip of each behavior. Be sure to explain your **documented** *algorithms and tests. Be prepared to demonstrate your work in lab.*

Finishing Up: Time and Self-Assessment

After preparing your wiki materials, it's time to update your self-assessment. Tally your individual time spent in your saved progress bars, including time spent posting to the wiki. Next, each of you should fill in the "End of Lab" column of your Lab 2 skillchart, providing self-assessments for each skill area **as of the end of the lab**.

7 Extra Credit: Localization

If you finish the lab early, consider the additional problem of how you can program your robot to localize itself in the presence of light sources with known fixed locations. To help you in this task, the course staff has marked out and mapped locations on the lab floor for the placement of three desk lamps. Using the coordinates provided by the staff, program your robot to complete two tasks:

- When placed between the three light sources, determine its orientation.
- After determining its orientation, determine its distance to the three light sources.

Deliverables: Chat with the staff about your approach and demo your robot.