Massachusetts Institute of Technology

Robotics: Science and Systems I

Lab 2: Chassis Build, Light Sensors and Braitenberg Behaviors

Distributed: Monday, 8 February 2010 at 3pm

Wiki Materials and Briefings Due: Tuesday, 16 February 2010 at 3pm

(Note: Tuesday, Feb 16 is a virtual Monday; we'll have class and lab that day.)

Objectives and Lab Overview

Your objective in this lab is to begin assembly of your robot body and implement its first exteroceptive sensing, a pair of light sensors. After you fabricate, mount, and calibrate the sensors, you will program the robot to exhibit several light-dependent behaviors originally described by Braitenberg. The lab gives you the opportunity to experiment with fairly simple means of reacting to sensory input and observe or predict a variety of behaviors for your robot.

This lab will give you the technical skills to incorporate and use a light sensor for reactive behavior. You will learn how to solder, how to handle batteries safely, and how to use tools for source code control under group development.

At the start of the lab, you will see a demonstration of the staff's exemplar robot exhibiting Braitenberg behaviors, and hear brief "labtures" (lab lectures) on soldering, battery safety and `svn`, the `subversion` source code control system (see the supplementary handouts).

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.

You should have these items on hand for the chassis assembly portion of the lab (Figure 1):

- 1 piece of pegboard
- 2 motors
- 2 solid wheels
- 2 caster wheels on mounting plates
- 2 motor screws

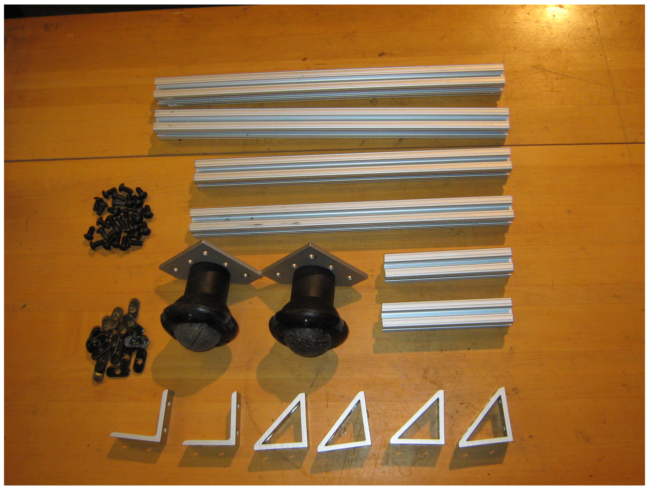
Figure 1: Materials needed for chassis assembly.

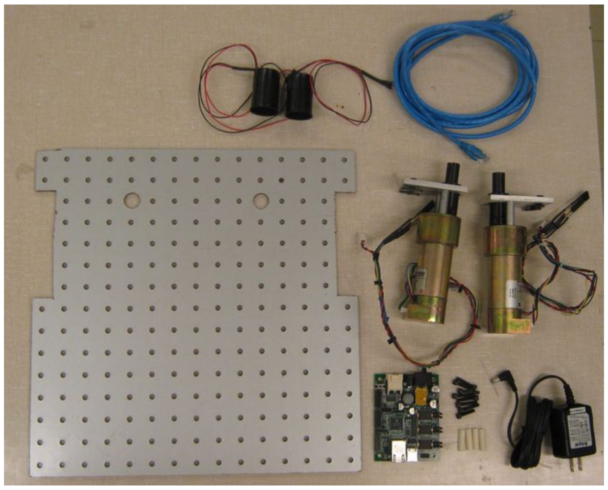
Figure 2: Materials needed for Braitenberg behaviors.

- 8 8-32 x 3/8” hex bolts
- 4 8-32 x 1” plastic hex standoff
- 3 15” aluminum 80/20 beams
- 2 13” aluminum 80/20 beams
- 2 5” aluminum 80/20 beams
- 6 brackets (2 L-brackets, 4 angle brackets)
- 42 1/4-20 x 1/2” hex bolts and sliders
- μOrc board, ethernet cable, AC power adaptor
- 1 lead-acid battery and cable with terminal connectors
- 2 light sensors and enclosures

For the Braitenberg portion of the lab, you should have these items on hand (Figure 2):

- Soldering handout
- Battery safety handout
- Two light sensors, wire, connectors, heat shrink tubing, a heat gun, a soldering iron and solder
- A photocopied handout on Braitenberg vehicles from the book entitled “Vehicles: Experiments in Synthetic Psychology” by Valentino Braitenberg
- Subversion source code control handout

Finally, you should have these items in your bin:

- Hex keys
- Multimeter (also used in Lab 1)

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

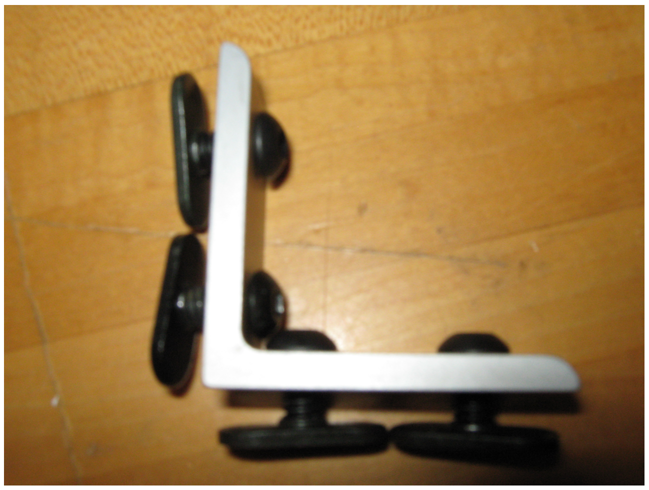

Figure 3: Rollbar assembly: Put bolts and sliders on the L-brackets.

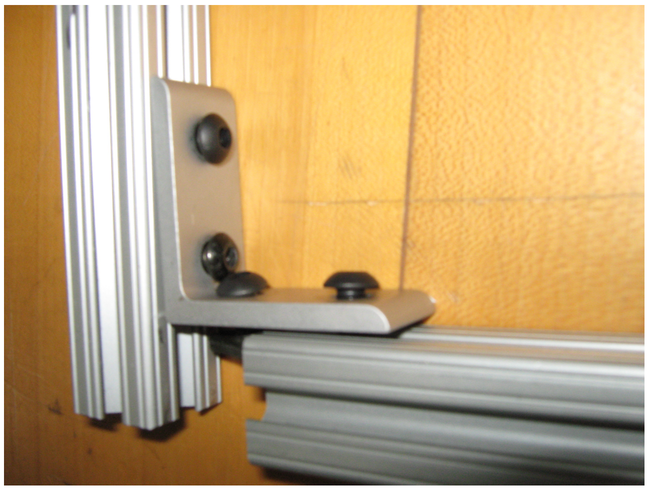

Figure 4: Slide the 5" segments of 80/20 onto the L-brackets.

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 at `http://projects.csail.mit.edu/rss/wiki/index.php/Main_Page`. 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 Assembling the Chassis

The road to success is always under construction.
— Lily Tomlin

You will now assemble the robot chassis. You may copy the staff's exemplar robot, follow the instructions given below, or perform some combination of the two. Some steps can be carried out in parallel. In general, defer tightening bolts until recommended by the handout; this will make your life easier when sliding together and adjusting the 80-20 segments.

1. Assemble the rollbar (see Figure 5).
   (a) Put four 1/4-20 bolts and sliders on each of the L-brackets, as shown in Figure 3. Do not tighten down the sliders yet.
   (b) Slide the L brackets onto a 15" 80/20 beam. Then slide the two 5" 80/20 beams onto the L-brackets, as shown in Figure 4.
2. Put 1/4-20 bolts and sliders on all the angle brackets, as shown in Figure 6. Do not tighten them down.
3. Put eight 1/4-20 bolts in the pegboard - one in each of the front corners, one behind each of the two side notches, and two catty-corner on each of the back corners. Place sliders on the ends of the bolts, but again do not tighten them down.
4. Slide the 15" beam of 80/20 onto the two front sliders, as shown in Figure 7. Slide two of the angle brackets onto the 80/20, placed so that their hypotenuses are facing inwards.
5. Slide the two 13" beams of 80/20 onto the two sides of the pegboard. As you reach the notches on the sides, place an angle bracket there and slide the 80/20 onto it as well. Be sure to orient the angle bracket so that its hypotenuse

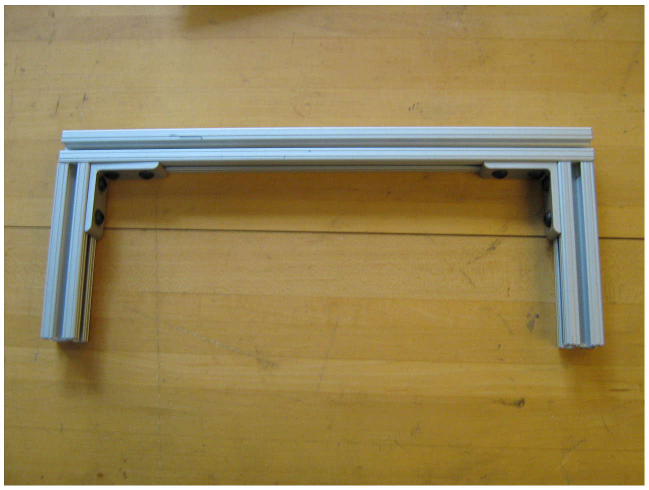
Figure 5: Completed rollbar.

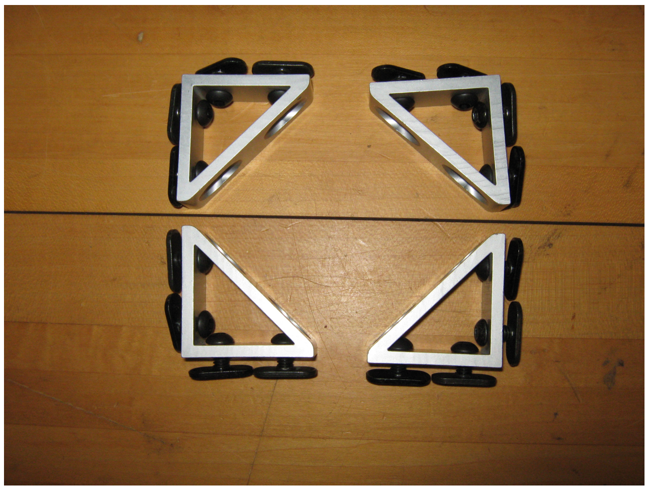
Figure 6: Angle brackets with sliders.

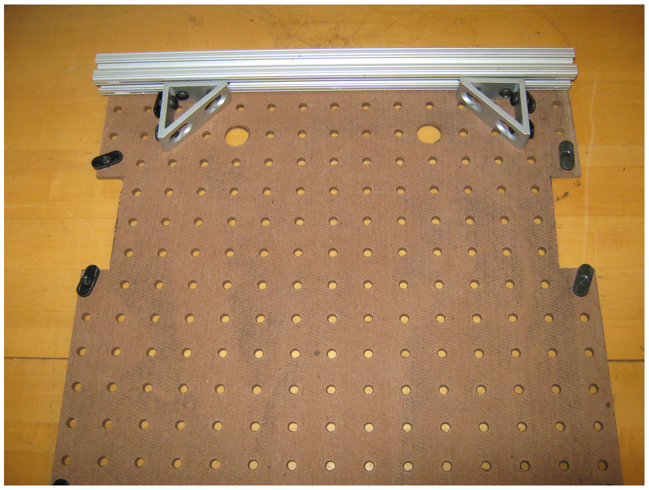
Figure 7: Slide 15” 80/20 and angle brackets onto front sliders. The placement of your sliders on the pegboard may look slightly different from that shown here.

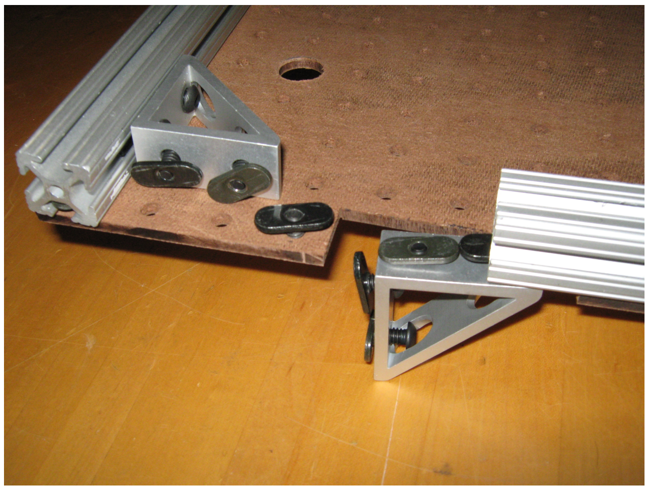
Figure 8: Slide the 13” 80/20 onto the sides of the pegboard, along with the angle brackets that will later attach to the rollbar.

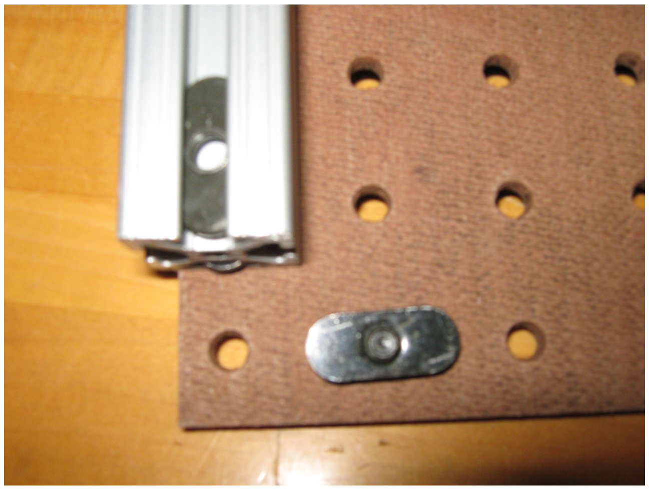

Figure 9: Slide the final piece of 80/20 onto the rear end of the robot.

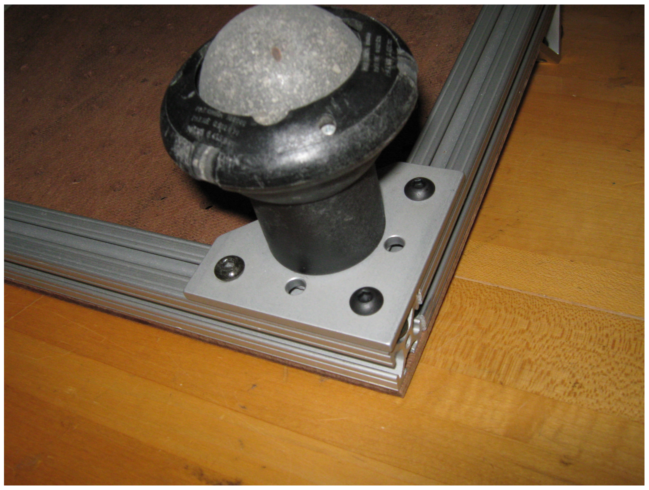

Figure 10: Tighten the casters onto the two rear corners.

is facing the back of the robot, towards you, as shown in Figure 8. As you reach the angle brackets attached to the 15" beam, slide the side beams onto it as well.

6. Place one slider in the top groove of each side beam. Place two more in each of the outside side grooves, for a total of three sliders per beam.
7. Slide the last 15" 80/20 bar onto the two back sliders, as in Figure 9. Put two sliders in the top groove.
8. Place the casters in the back corners, as in Figure 10. Attach with 3 1/4-20 bolts each, and tighten them down.
9. You can now tighten all of the 1/4-20 bolts on the chassis, except for those on the angle brackets intended for the roll bar.
10. Flip over the chassis, and attach the rollbar to the angle braces. It should look like that shown in Figure 11. Move the rollbar and braces as far forward as possible in the pegboard slots, and tighten all the remaining bolts.
11. Mount a motor to each side, using the two sliders placed there previously. Mount them directly underneath the rollbar, as in Figure 12. Try to ensure that your robot's motor shafts are collinear (why?).
12. Attach the wheels to the motors.
13. Mount your μOrc and battery to the robot chassis. You may mount them wherever you like, but consider the effect of the battery's weight on the robot's driving performance. Leave room for the laptop behind the rollbar.

Deliverables: Post to the wiki a brief discussion of your assembly process, especially steps that went particularly smoothly, were particularly difficult, or required substantial rework.

3 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.

1. Consult the wiring of the light sensor on the exemplar, and Figure 13.

   Write down the formula for the voltage divider which gives V_{out} in terms of $V_{in} = 5\text{V}$, 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?

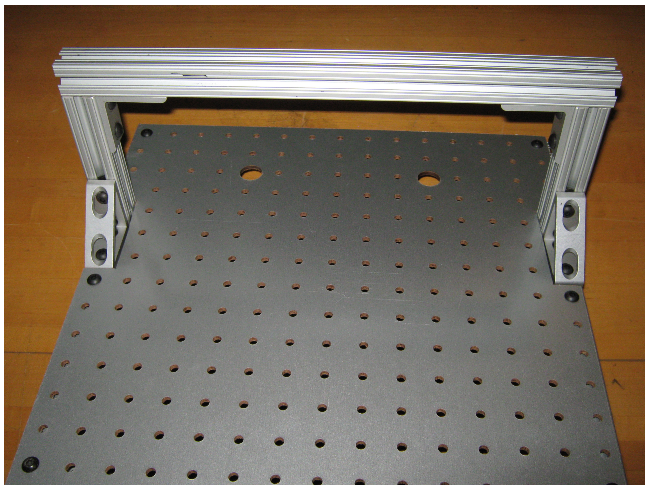

Figure 11: Attach the rollbar to the angle brackets on the top of the robot.

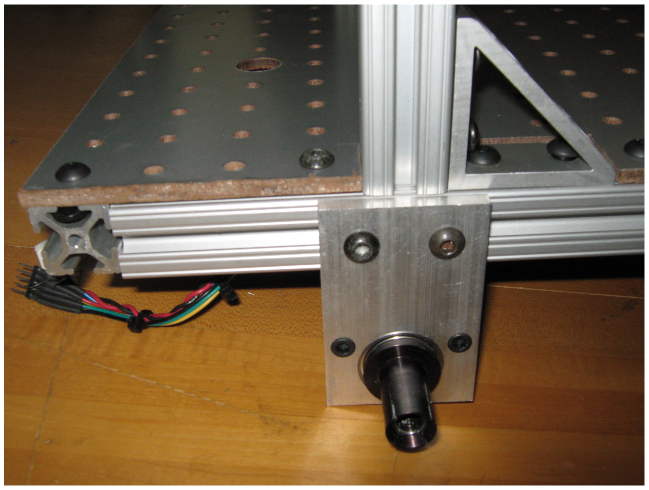

Figure 12: Attach the motors to the sides of the robot, directly underneath the rollbar.

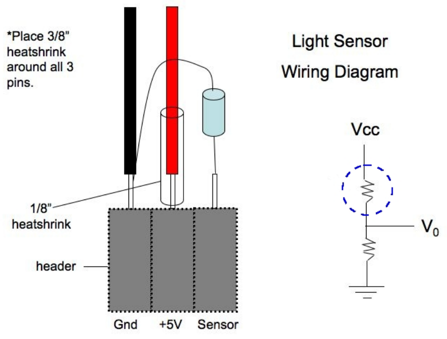
*Place 3/8" heatshrink around all 3 pins.
1/8" heatshrink
header
Gnd
+5V
Sensor
Light Sensor Wiring Diagram
Vcc
V_0

Figure 13: Wiring diagram of the light sensor.

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. A staff member will demonstrate how to build and mount each light sensor on the robot. For each sensor, place the signal pin of the connector you make into header 0 or 1 of the AD ports on the μORCboard, exactly as on 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.

4 Populate your Working Area with Staff Code

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

NOTE: The *first* team member to log in should do steps **1, 2, 3**. All team members must assist the first team member on step **4**. Each subsequent member should do steps **1, 2, and 5**.

1. Have each team member login to the Sun workstation assigned to your team, using his/her athena user name and default password (which is the same as the username). Change your individual password using the command

   ```
   passwd
   ```

2. Now each team member should log in and check out a working copy of your group's repository into his/her home directory using the following steps. Notice that all members now have a working copy of the group repository, at the same revision level (version number).

   (a) Open a terminal and make sure that you are in your home directory by typing:

   `cd ~` (or just `cd`)

   (b) Check out a working copy of your group's Subversion repository with the following command:

   ```
   svn checkout https://svn.csail.mit.edu/rss-2010/group(Your group number) ./RSS-I-group
   ```

   You will be asked for your password to access *your svn repository*. Use the default password (your athena user name). Note that this password is not the same as the password for your workstation account.

3. When the *first* team member logs in, s/he should add the Braitenberg source to your group repository by executing the following three steps:

   (a) Go to your working copy of the repository by

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

   (b) Pull the Braitenberg source into your working copy of the 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)`"`

   If you do not specify `-m` on the command line you will be brought to a screen where you can add a commit message by typing `i`. You will then be able to insert text. To quit the program you need to press `ESC` and you will now be typing at the bottom of the window. Enter `:wq` meaning write and quit. (You are in the editor `vi`; you can learn more commands online if you wish.)

4. Also, you have to reset your default password for the repository. When the first member is logged-in, *all members* should do the following:

   ```
   htpasswd ~/RSS-I-group/passwdGroup-(Your group number) username
   ```

   The username should be the same as your workstation account. Enter your chosen password (you will be asked twice). Commit your working copy after all members update their passwords. **NOTE:** this does not immediately change the password. From the next lab, you will be asked for the new password when you access your repository.

5. Now that you have populated your group repository with the Braitenberg source, each team member should log in to and update his or her own working copy of your group's repository by changing to the `~/RSS-I-group/` directory and running:

   ```
   svn up
   ```

5 Java Source

In this lab, you will write Java source code to create a reactive robot controller. Refer to the Java tutorial at `http://java.sun.com/docs/books/tutorial/index.html` as needed. If you are unfamiliar with Java, we recommend these sections: Getting Started; Learning the Java Language; Essential Java Classes; and Collections.

Before continuing, familiarize yourself with the Java source in `Braitenberg/src/`. Read and understand each class along with its individual members and methods.

You will edit and add code to some of the class files in `Braitenberg/src/`. We have written method templates within classes for you to use. We have placed `// StudentCode` comments throughout the code to mark locations in which we ask you (below) to complete the implementation of various supplied methods. This handout indicates explicitly which methods you will modify.

Since the lab requires motor control code which we will not visit until the next lab, we provide with two `jar` files `uORCInterface.jar` `motorControl.jar` that include pre-compiled classes for motor control. These classes enable you to set the angular velocity of each robot wheel. You can import the necessary methods by including the following in your `java` source:

```
import MotorControlSolution.*;
```

These classes are provided for comprehension and execution only.

We use the `Apache` "`ant`" build tool for compilation. You can compile and run the code from `Braitenberg/src/` with the following steps:

1. Compile by typing `ant` in the `Braitenberg/` directory. View the `build.xml` file to see the configuration information `ant` uses.
2. Plug the ethernet cable into the μOrc board. After a few seconds, the wired network icon should appear at the upper right of your desktop, along with the message `Connection Established`.
3. Run your code by typing `ant run` while in the `Braitenberg/` directory.

While you are required to be familiar with command-line Java utilities, you may also compile, run and debug your code within `Eclipse`, the IDE provided for RSS use. The staff will give a brief tutorial on `Eclipse` during the Lab.

For editing, use `Eclipse` or `emacs` or `vim` or `gedit`. (There are other useful applications on your Sun workstation, including a web browser (`mozilla`), a plotting utility `gnuplot`, a pdf viewer (`evince`), and a postscript viewer (`gv`) for viewing plots.) Use the workstations only for RSS-related tasks.

6 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.

Warning: the next portion of the lab involves strong, moving motors. Test your robot on the wooden stand (where its wheels will spin freely) before letting it move on the ground. To avoid injury, keep all clothing, jewelry and hair away from the motor shafts!

7 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.

8 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**.