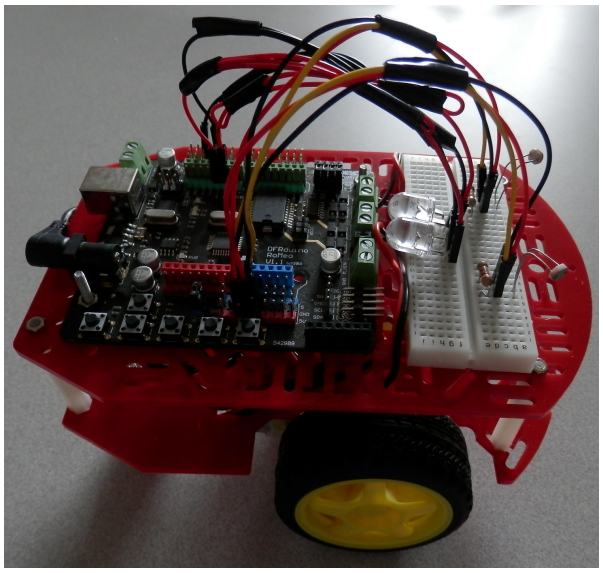
Physics Cross-Curricular Lab -- Robotics and Programming

Prepared by Lance Goodridge and Joshua Veltri



Introduction:

Robotics is a branch of technology that involves the design, development, and operation of autonomous mechanical, electrical, and software systems. The software is the logic that controls the mechanical and electrical components based on input from various sensors. Autonomous and semi-autonomous mechanical and electrical systems are useful for a variety of applications, from everyday devices such as cars and smartphones to more specialized systems such as defense/weapons systems and "smart" prosthetics. Robotics and programming are among the fastest growing and best paying fields in science and technology.

Purpose:

The purpose of this lab is to serve as an introduction to the basics of engineering design, programming, and robotics. Through exposure to these concepts, the hope is to expand interest in engineering and technology, and to help students to decide whether an engineering or programming career could be right for them.

Goals:

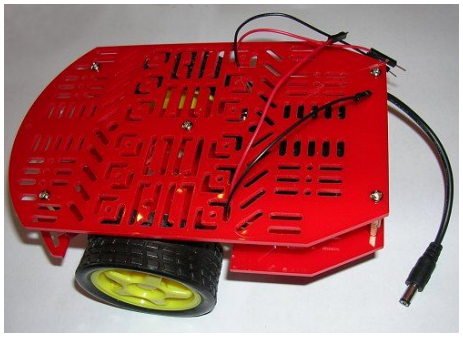
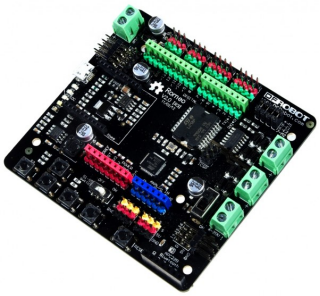
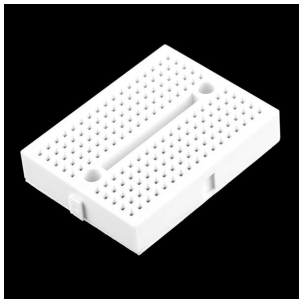
- Build and understand the mechanical and electrical systems of a simple robot, including components such as DC motors, Arduino boards, breadboards, LEDs, resistors, and photoresistors.
- Develop a basic understanding of programming, including some very basic syntax, the flow of an Arduino program, how to read and write signals to electrical devices, and how to manipulate variables and user-defined constants to vary the output sent to those devices.

Part I: Assembling the Robot Frame

Goal:

- Assemble the robot frame, including mounting the arduino board and the motors and wheels

Required Materials:

Robot Frame (1x)		Arduino Board (Romeo) (1x)
		
DC motor (2x)		Wheels (2x)
		
Omniwheel (1x)		Breadboard (1x)
		

Required Tools:

- Screwdriver
- Solder and Soldering Iron
- Needle-nose Pliers

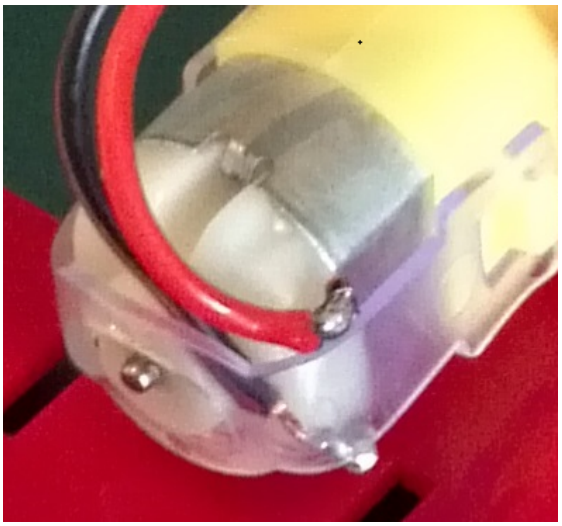
1. If there are none are already attached, solder approximately 6 inch wire leads to each terminal of the DC electric motors.

    a. Strip about ½ inch of the plastic insulation off of the end of the wire lead.

    b. Twist the end of the wire so that the now-exposed individual wire strands are all held together.

    c. Feed the end of the wire through the hole in the terminal on the motor. Bend the wire back around so that it is securely attached to the terminal.

    d. Using a soldering iron, carefully melt a little solder and surround the exposed wire and the terminal; allow the solder a minute to resolidify before handling the motor again. A well-soldered connection should look something like this:

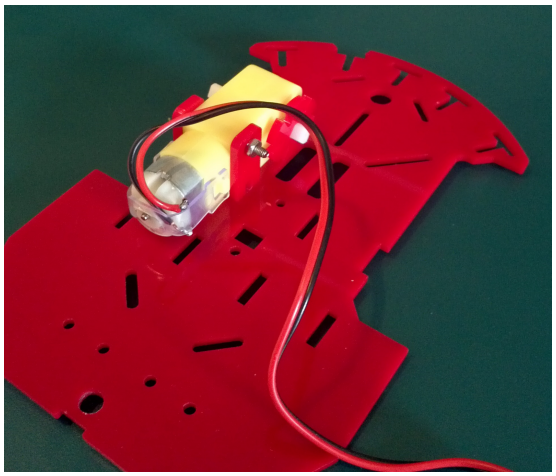


2. Attach motors and wheels to bottom of frame

    a. Take two of the four small plastic tabs. Place one up through the slot in the middle of the bottom piece of the frame. Hold the other in the nearby notch on the edge of the frame.

    b. Place one of the motors between the two tabs. Take one of the long bolts and insert it through the hole in one of the plastic tabs, through the hole that passes through the motor housing, and out the hole on the other plastic tab.

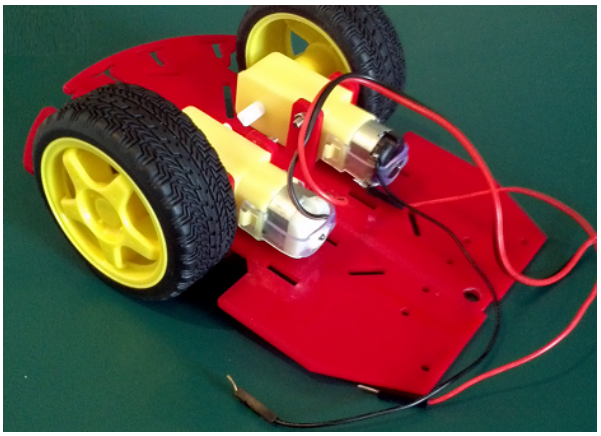
    c. Tightly wind a nut on the end to hold everything in place permanently. The correct placement of the tabs and alignment of the motor is shown in the picture below.



    d. Repeat with the second motor on the other side of the lower frame.

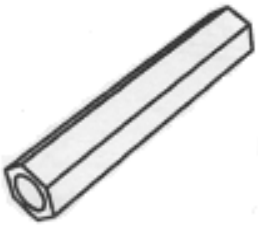
    e. Press the wheels onto the axles of the newly mounted motors.

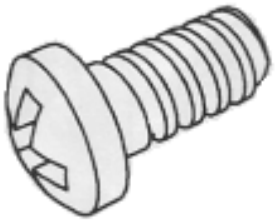
placement of the tabs. The frame should now look something like this:



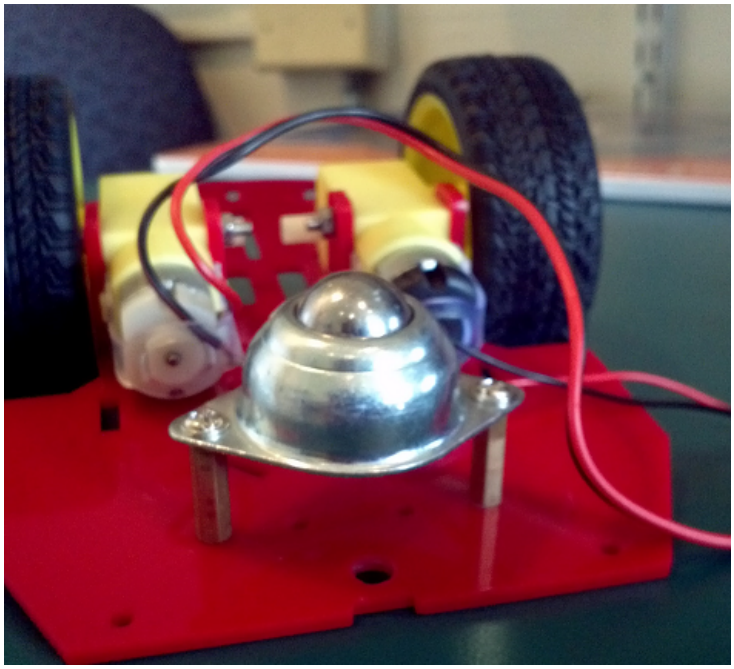
3. Mount the omniwheel

    a. Take two of the long spacers and four of the short round-headed screws.





Attach the two spacers to the frame and the omniwheel to the spacers as shown below.

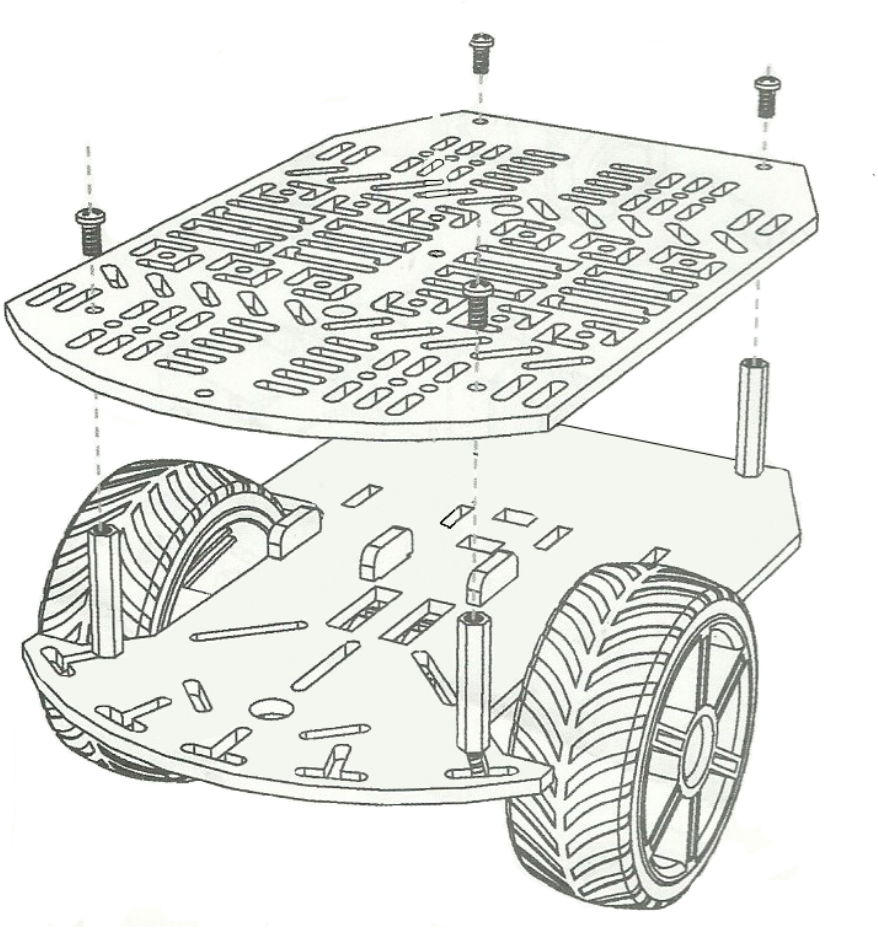


4. Add top layer of frame

    a. Flip the robot so that omniwheel is on the table/ground

    b. Attach the long spacers in each corner of the robot, facing upwards, using the short, round-headed bolts (same parts as in step 3).

    c. Place the top layer of the frame on top of the spacer, and bolt it to the spacers with short, round-headed bolts (see picture below).



5. Mount Arduino Board and Power Supply

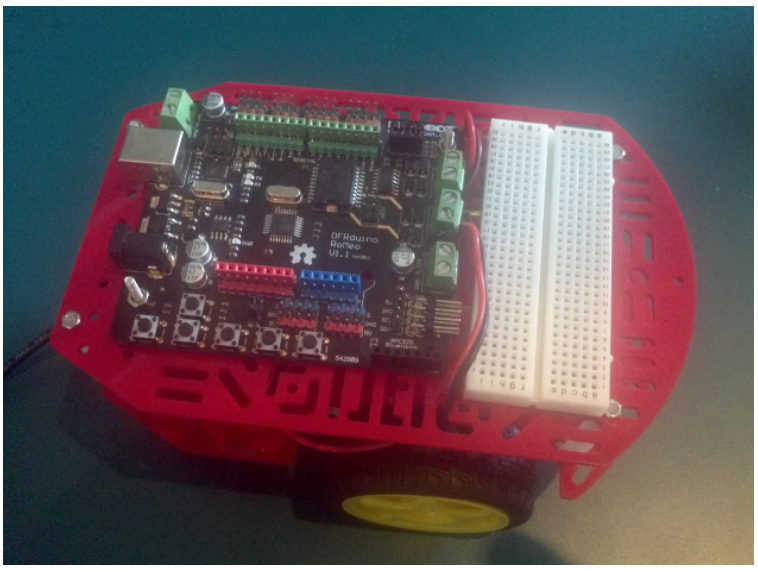
   Note: The front of the robot will be the side with the two black wheels attached to the motors. The back will be the end with the omniwheel.

   a. At the back of the top piece of the frame (above the omniwheel), mount the arduino board. The power and USB jacks should be pointing directly off of the back of the frame. Secure the board down with at least two small bolts and their corresponding nuts.

   b. Connect the wires attached to the left motor into the M1 green connector on the Arduino board. You'll need to strip 1/4 to 1/2 an inch of the plastic insulation off of the end of each wire. Then take your flat headed screwdriver and loosen the screws on the top of the green connector, opening the gate. Insert the wires into the gate, and tighten the screw, securing the wires in place. Repeat with the right side motor, connecting the it to the green M2 connector.

   c. Secure the breadboard on the top part of the robot, in front of the arduino board. Your breadboard will probably be a bit smaller than the one pictured below, and can likely be bolted to the frame.

   d. Using two strips of sticky-backed velcro, secure the battery mount on the bottom of the top piece of the robot frame (underneath the arduino board). Be sure the power plug can reach the connector on the board



Part II: Testing the Arduino Board

Goals:

- Install and ready Arduino for use on your computer.
- Learn how to upload a program to the Arduino board.
- Learn how to use the Serial Monitor and Arduino IDE

Background:

A program is a set of instructions given to a machine for execution. Whereas the hardware of a machine can be referred to as its body, the software can be thought of as the 'brain' of the machine: it takes in the data received by the body and tells it how to respond. Programming is used today to operate almost anything technological you see today, from the simple google search on your computer, to the cutting edge robots waiting to be released. For robots in particular, who must engage and respond to their environment in real time, software is especially important. The robot must be programmed to read its inputs, and output appropriate responses, or it will likely get into undesirable situations, e.g. driving off a cliff. The board we're using, the Arduino, is a microcontroller designed to easily do just that; we can program our Arduino to read data from our sensors, make appropriate decisions, then write those to our output devices. For our very first program, we're going to upload a *very* simple piece of software to print something to the screen.

Step One: Installing Arduino on your computer

1. If Arduino is not already installed on your system, download it from http://arduino.cc/en/Main/Software and install it.
2. Using the USB cable, connect the arduino board to your computer. If you're on Windows and this is the first time someone's done so on your computer, Windows will try to install the appropriate drivers, but will fail. You'll need to go into "Control Panel" → "Device Manager", then select "Arduino UNO" under "Other Devices." Right-click "Arduino UNO", and select "Update Driver Software" → "Browse My Computer". Find the location of the arduino drivers folder, which should look something like:

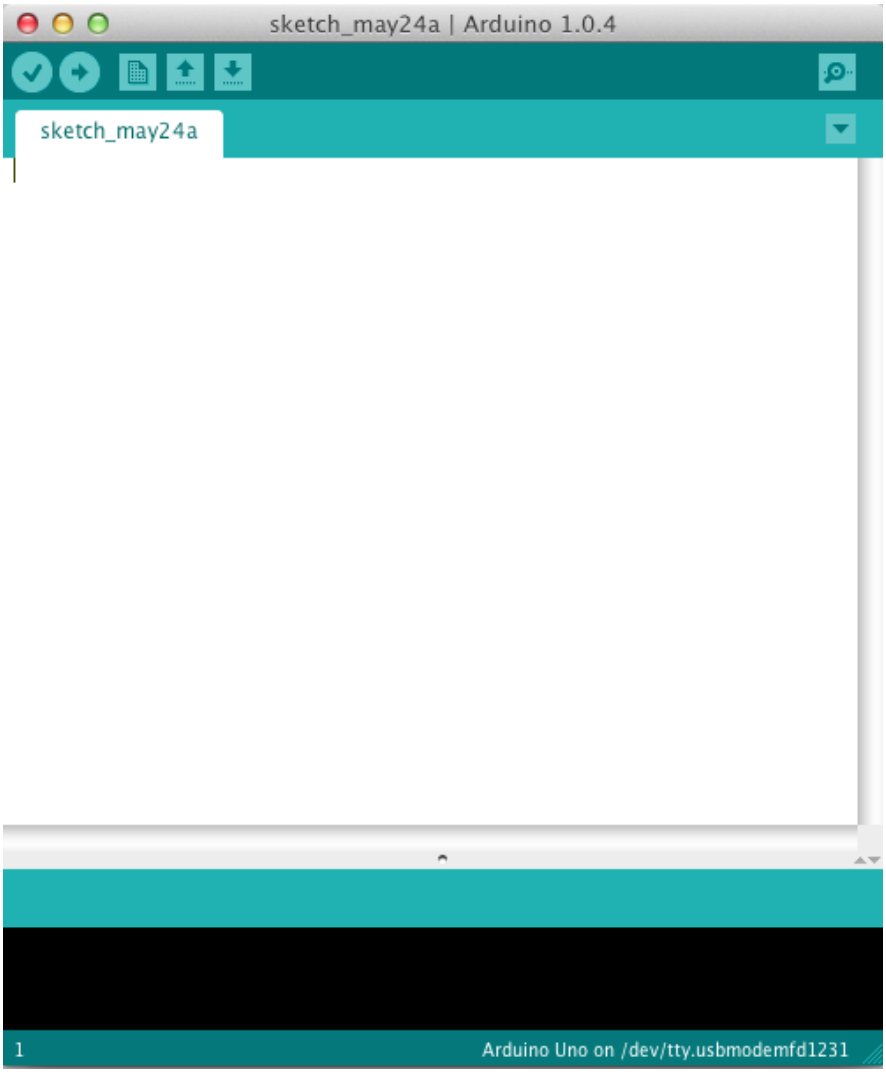
   C:\Program Files (x86)\arduino-1.5.2\drivers
   or
   C:\Program Files\arduino-1.5.2\drivers

   Note that the numbers following "arduino-" in the folder name (in this case, 1.5.2) may be different, depending on which version of the software you've installed. Click "Next" and allow Windows to install the drivers.

   See http://arduino.cc/en/Guide/HomePage if you have a Mac, or if you have any trouble with the above instructions.

Step Two: Uploading a program to the Arduino board

1. Open up arduino.exe to open the Arduino IDE. This where you will write your programs. (Arduino calls them “sketches”). It should look something like this:



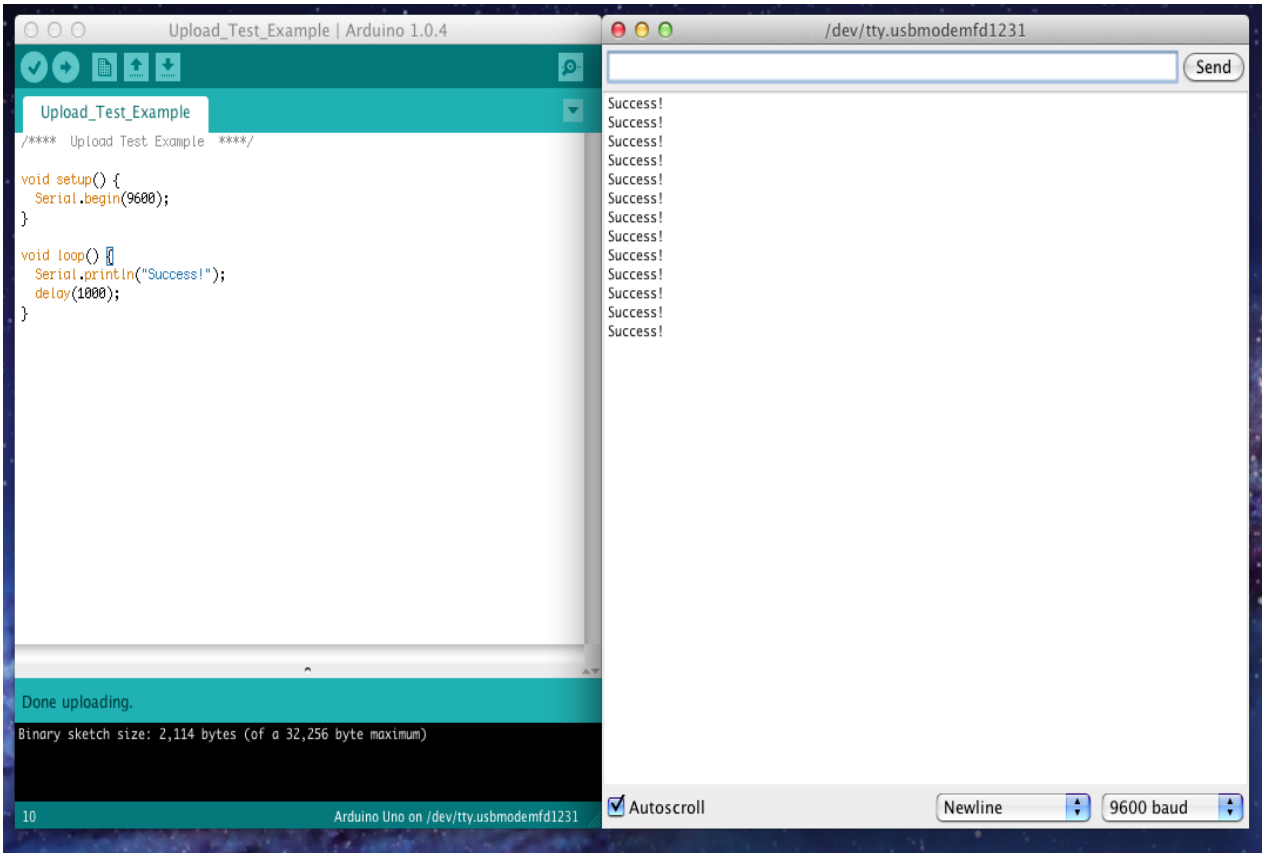
2. Copy and paste the following code into the space:

```
/****  Upload Test Example  ****/

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Success!");
  delay(1000);
}
```

3. First, click the check mark in the top left corner of the IDE. This 'verifies' your program to make sure there aren't any syntax or board-crashing errors. If it finishes verifying, and the message near the bottom says "Done Compiling", you may move on to the next step. If not, ensure you have properly copied the code into your IDE.
4. Now, connect the Arduino to your computer using the USB cable. A small power LED on the board should turn on.
5. Go to Tools → Board, and ensure that "Arduino Uno" is selected. Note: The Arduino board we're using is the RoMeo, which a variation of the Uno.
6. Go to Tools → Serial Port and select the correct USB port. If you are on a PC, it will probably look something like "COM #". If you are on a mac, it will probably look something like "/dev/tty.usbmodemfd1231". If either of these appear, select it. If not, it is likely an issue with your drivers. Try re-downloading + installing them, or go to http://arduino.cc/en/Guide/HomePage for troubleshooting help.
7. Finally, click the arrow next to "Verify" in the top left corner of the IDE. This will upload your program to the Arduino Board. When the message in the bottom left says "Done Uploading", the program will begin to run on your computer.

Step Three: Using the Serial Monitor

1. Go to your toolbar and go to Tools → Serial Monitor. A second window should appear on your screen.
2. Go the bottom of the Serial Monitor and make sure that "Autoscroll" is checked, the first selector is set to "Newline", and the baud rate is set to 9600 baud (the same as what was written in Serial.begin().
3. You should see a list of "Success!" begin to appear on the Serial Monitor. If all has gone well, it should look something like this:



Step Four: Saving your program

1. If the previous step was successful, you may close the Serial Monitor and unplug your Arduino board.
2. Go to your toolbar and select File → Save As to save this program. Choose where you would like to save the file (creating a new Arduino folder may be a good idea), then give your file a name, such as "My First Program", and click Save.
3. If you ever wish to open this program again, re-enter the Arduino IDE, select File → Open, and select the program.

Part III: Assembling the LED and Breadboard

Goals:

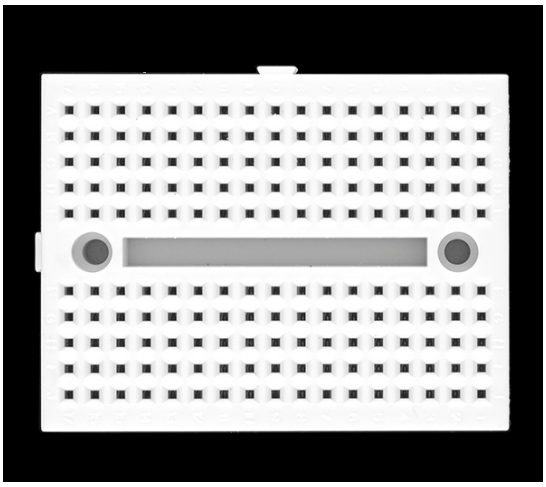
- Mount LED onto the robot
- Become familiar with the way a breadboard works and is wired

Background:

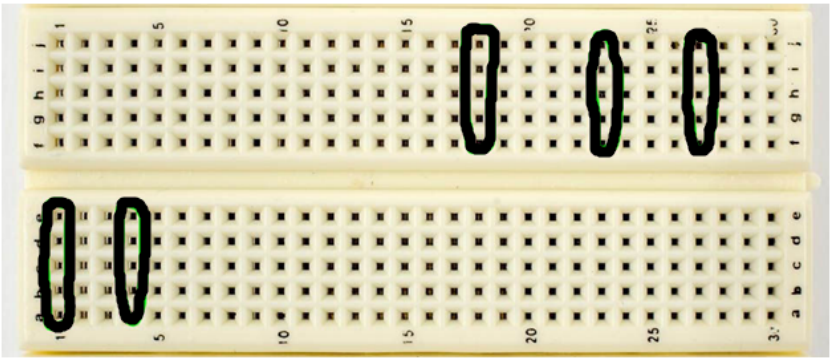
For this part of the lab, we will be using a breadboard to mount and wire in Superbright LEDs to the Arduino board.

An LED (Light Emitting Diode), is a small light source widely used across engineering platforms. The default color is blue, but LEDs for all wavelengths of light, including ultraviolet and infrared, are produced. The two leads can either be plugged in directly, or wired into the Arduino. The short lead is called the anode, and is the positive end of the LED. This typically gets plugged into Ground. The long lead is called the cathode, and is the negative side of the LED. This typically gets plugged into your power supply. For a constant light, you would plug it into either the 3.3V or 5.5 V pins, since those provide steady voltage. Since we wish to control the LEDs with software, however, we will plug them into one of the analog pins, so we can control when, and how bright we want the light to be.

LEDs are typically grouped together to produce more visible light for civil applications. They are often used for street lights, car lights, signs, lamps, and many other indicators and light sources. They are also commonly used for decorations, such as Christmas lights.In comparison to incandescent light bulbs, LEDs are more quicker, smaller, and more efficient, but are typically more expensive than their counterparts.



A breadboard is an electrical construction base mainly used for prototyping circuits and other wiring designs. The biggest advantage of the breadboard is that it does not require soldering to make connections, therefore, it is quick, easy to use, and recycleable. Each of the pins in a vertical column are internally connected - they are essentially soldered together for you. Thus, we can treat every pin in the same numbered column as though they were all touching. The pins are not horizontally connected, however. This is shown in the diagram below, where the pins in each circle are connected to each other, but to no other pins:

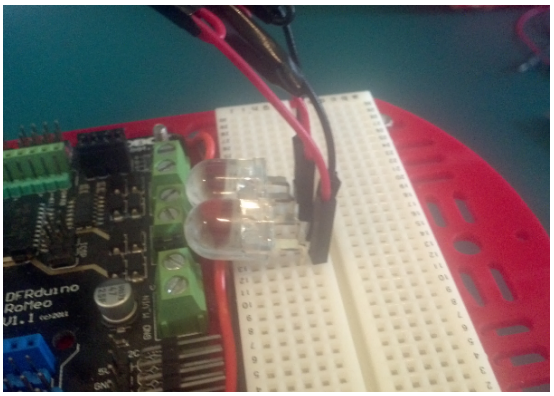


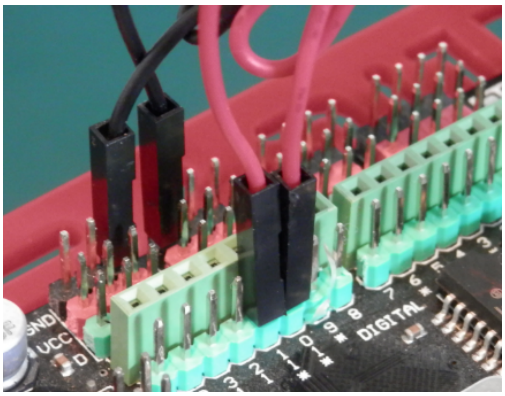
We can connect pins from different columns together by using jumper wires. “Jumping” a pin in column 1 to a pin in column 3 for example, will connect the all of the pins in both of those columns together. We will be using a breadboard to assemble the photoresistor apparatus and to mount the LEDs. Although a real engineering lab might involve the use of something more permanent, such as a circuit board, the quick, and solder-less capabilities of the breadboard make it perfect for this project.

Mounting the LEDS:

1. If the leads on each superbright LED are not already bent, bend them as shown in them as shown in the image below:

2. Near the middle of the breadboard, insert the LEDs as shown below. Note which one of the leads on each LED is longer than the other.
3. Take one red male/female jumper and one black male/female jumper. Place the male end of red wire in a pin on the breadboard connected to the long lead on one of the LEDs and the female end on the red pin (labeled VCC) of digital port 10 on the arduino board.
4. Then take the black jumper wire, insert the male end into a pin connected to the short lead on the LED, and the female end on the black pin (labeled GND) of digital port 10.
5. Repeat with the second LED, wiring it in the same manner to digital port 11.





Part IV: Controlling an LED

Goals:

- Become familiar with the basics of programming
- Upload a program to blink an LED
- Simulate an analog output using PWM
- Upload a program to fade an LED
- Learn how to manipulate values to control how a program works

Background:

Now that the LEDs have been mounted, we will control them with programs. First, we will turn them on and off using digital writes, then, we will simulate analog outputs using PWM. It is not necessary to understand all of the software, but a brief look at the programming primer is recommended if you get stuck, or are interested in learning it.

Step One: Make an LED blink

1. Open the Arduino IDE and plug in your Arduino board.
2. Copy and paste the following code into the environment:

```
/****  Digital Blink LED Example  ****/

#define LED_PIN 10       //Which pin the LED is plugged into
#define BLINK_TIME 1000  //How long each "blink" takes (in milliseconds)

//Readies the board and software
void setup() {
  Serial.begin(9600);       //Prepare Board to receive data
  pinMode(LED_PIN, OUTPUT);  //Prepare the pin to send output values
}

//This loops continuously
void loop() {
  blinkLED(LED_PIN, BLINK_TIME);  //Blink LED
}

//"Blinks" the LED
void blinkLED(int pin, int blinkTime) {
  digitalWrite(pin, HIGH);  //Turn LED on
  delay(blinkTime / 2);     //Wait (leave on for half of the total blink time)
  digitalWrite(pin, LOW);   //Turn LED off
  delay(blinkTime / 2);     //Wait (leave off for half of the blink time)
}
```

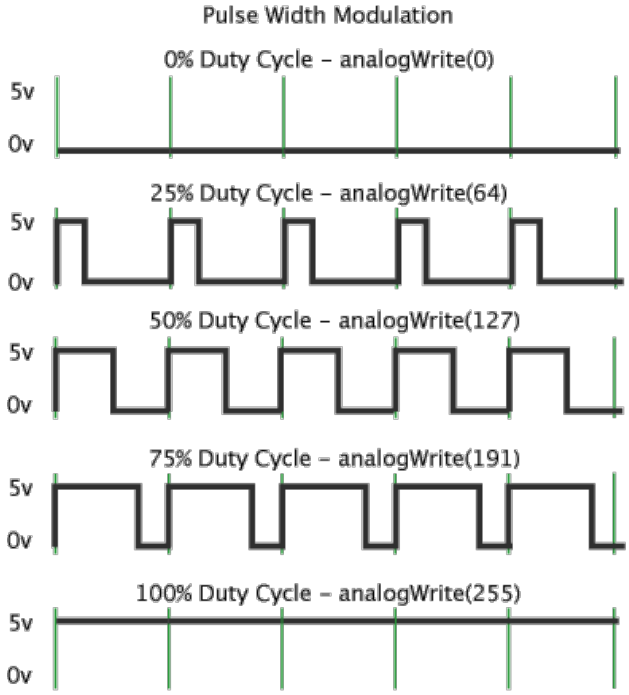
Note: The words after the "//" and between the "/* */" are called comments. They are ignored by the program, and are used to explain what is going on (to other humans).

3. Verify the program to make sure it has been copied correctly, then upload if there are no errors. If all goes well, you should see the LED plugged into pin 10 blink on and off, in one second intervals.

Step Two: Customizing your LED blink program

1. Now that you have your LEDs blinking, it's time to make the program your own. Return to your Arduino IDE.
2. The BLINK_TIME constant determines how long each blink takes. If BLINK_TIME is set to 1000, as it is by default, each blink is a second long (because 1000 milliseconds = 1 second), so the LED stays on for 500 milliseconds, and off for 500 milliseconds. Change the BLINK_TIME to 3000 and re-upload it. Note what happens.
3. Try to make the LED blink the way you want it to by trying several different values.

We now know how to turn an LED on and off. But what if we want it to be only 50% on? Or 25%? Our motors won't be of much use if we can only run them at full power or nothing. Unfortunately, with digital pins, the only signals we can send are on (HIGH) and off (LOW) values. This is where PWM comes in.



Analog signals can be approximated with digital signals using Pulse Width Modulation (PWM). By turning a digital signal on and off extremely fast, an analog value is approached. For example, in the picture (above, below, etc.), each green line represents a cycle which is run every two milliseconds. By leaving the digital signal on 25% of each cycle, the perceived output approaches an analog value of 64, or 25% of 255, the maximum analog output.

This can be applied for any desired output percentage. Calling an analogWrite on certain digital pins will actually invoke PWM. With this, you use the digital pin exactly like you would an analog, by writing values from 0 to 255; the board will output the correct PWM signal for you.

Step Three: Fade an LED

1. Open the Arduino IDE and plug in your Arduino board
2. Copy and paste the following code into your sketch area

```
/****  Analog Fade LED Example  ****/
#define LED_PIN 10      //Which pin the LED is plugged into
#define FADE_TIME 5000  //How long each fade takes (in milliseconds)
```

```
int brightness;     //The current brightness of the LED
int fadeDirection;  //Determines whether the LED is getting brighter or darker

//Readies the board and software
void setup() {
  Serial.begin(9600);        //Prepare Board to receive data
  pinMode(LED_PIN, OUTPUT);  //Prepare the pin to send values
  brightness = 0;            //Set initial brightness to 0 (off)
  fadeDirection = 1;         //Set initial fade direction to +1 (getting brighter)
}

//This loops continuously
void loop() {
  fadeLED(LED_PIN, FADE_TIME);  //Fade LED
}

//"Fades" the LED
void fadeLED(int pin, int fadeTime) {
  brightness += fadeDirection;               //Increase or decrease brightness, according to fade
direction
  if (brightness >= 255) fadeDirection = -1;  //If max brightness (255) reached, start getting
darker
  if (brightness <= 0) fadeDirection = 1;     //If minimum brightness (0) reached, start getting
brighter
  analogWrite(pin, brightness);              //Set the LED at the new brightness
  delay(fadeTime / 255 / 2);                 //Wait
}
```

3. Click Verify, then Upload if there are no errors. If the LED then begins to fade on and off, everything has gone smoothly

Step Four: Changing your fade program

1. Return to the Arduino IDE
2. This time, the FADE_TIME constant determines how long each “fade” takes, change this value to your liking and see how it affects the LED.
3. Now for some exercises. Can you:
    a. Write a program that fades the LED in three and a half second intervals?
    b. Write a program that fades the other LED (the one plugged into pin 11)?

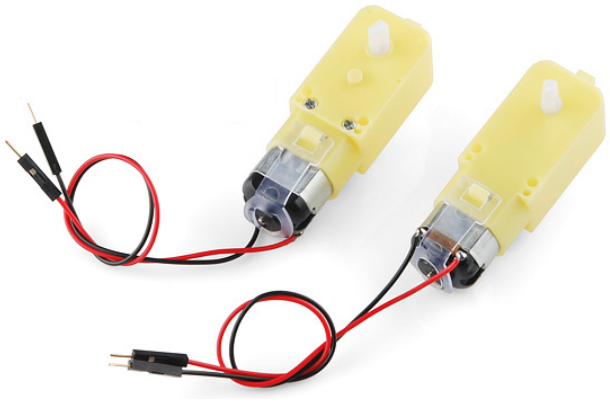
Part V: Controlling the Motors

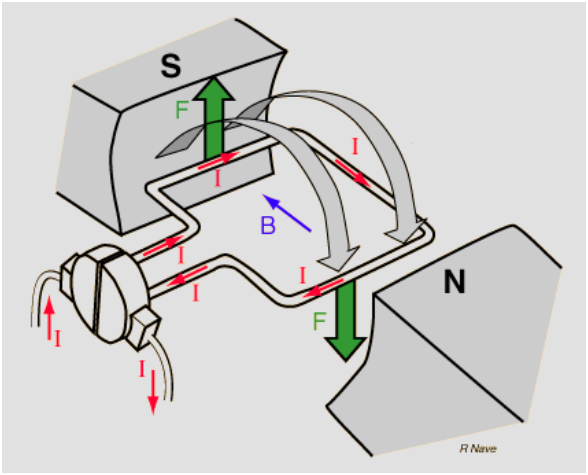
Goals:

- Operate a single motor using the Arduino board
- Operate both motors using the Arduino board
- Write programs to make the robot move in specific ways

Background:

Now that we've mastered writing to an LED, let's look at a more applicable output: a motor.





Direct Current (DC) Electric motors are fairly simple devices that convert electrical energy into mechanical energy. However, fully understanding them requires a basic understanding of electricity and magnetism - you'll talk more about electric motors later this year. For now, we'll just take a very simplified look at how a DC motor works.

The most basic DC motor consists of a current-carrying wire loop (that is, a wire loop carrying moving charges) next to a magnet. Moving charges are affected by magnetic fields, so when the current in the wire is flowing, the moving charges in the wire experience a force. This force on the wire makes the wire loop spin on its axis. On most DC motors, if the positive and negative wires are swapped at the power supply, it will reverse directions because the direction of the force on the wire loop depends on the direction of the current.

A typical electric motor has an efficiency of 70-85% -- this means that 70-85% of the electrical energy input into a motor is output by the motor as rotational motion; the other 15-30% is lost as heat and sound. Electric motors are used in everything from hybrid cars to power tools to analog clocks.

We'll be using the motors to control the movement of the robot. The motors will run as long as we give it two commands in the software: 1. Which direction to run, and 2. What percentage of power to run it at.

Step One: Controlling a single motor

1. Open the Arduino IDE and plug in your board. Also, make sure you are holding the robot up, or have it propped up on something when you run these programs - you don't want your robot to drive off the table!
2. Copy and paste the following code into your sketch area:

```
/****  Single Motor Example  ****/

#define FORWARD HIGH  //Tells motors to go forward
#define REVERSE LOW   //Tells motors to go in reverse

#define START_BUTTON A7  //This pin controls which button will start the program

#define MOTOR_DIRECTION_PIN 4  //This pin controls the motor direction
#define MOTOR_POWER_PIN 5      //This pin controls the motor power

int motorDirection = FORWARD;  //Controls whether the motor runs forward or in reverse
int motorPower = 125;          //Controls how fast the motor runs (from 0 - 255)

void setup() {
  Serial.begin(9600);
  pinMode(MOTOR_DIRECTION_PIN, OUTPUT);
  pinMode(MOTOR_POWER_PIN, OUTPUT);
  waitUntilButtonIsPressed();
}

void loop() {
  runMotor(motorDirection, motorPower);
}

/* Runs the motor in the direction given,
 * at the specified power */
void runMotor(int direction, int power) {
  digitalWrite(MOTOR_DIRECTION_PIN, direction);
  analogWrite(MOTOR_POWER_PIN, power);
}

/* Waits until the start button is pressed
 * to continue */
void waitUntilButtonIsPressed() {
  while (analogRead(START_BUTTON) > 500) {
    delay(100);
```

```
  }
}
```

Note: For safety, we added a function that requires you to press the S2 button before the program begins to loop. For this lab, the robot will not operate until the button marked S2 is pressed. Also note, you can press the button marked either RST or RESET to stop the robot at any time.

3. Verify and Upload the program to your board. When it is finished, press the S2 button to start the robot. You should see one wheel (the left one) moving forward at about half its maximum output. To accomplish this, all we did was digital write a direction (FORWARD) to the direction pin, then analog write a power (125) to the power pin, which approximated a 125 value with PWM.
4. Now, change the motorDirection variable from FORWARD to REVERSE. Verify and Re-upload your program. You should now see the wheel spin in the opposite direction.
5. Next, change the motorPower variable to another number (between 0 - 255). You should see the wheel speed up, or slow down depending on which number you chose.
6. Change the motorDirection and motorPower variables until you are comfortable making the motor do what you want it to.

Step Two: Controlling both motors

1. We're going to control both motors next. Save the single motor file somewhere and open up a new Arduino sketch.
2. Copy and paste the following code into your sketch area:

```
/****  Double Motor Example  ****/

#define FORWARD HIGH
#define REVERSE LOW

#define START_BUTTON A7  //This pin controls which button will start the program

#define LEFT_MOTOR_DIR_PIN 4     //This pin controls the left motor direction
#define LEFT_MOTOR_POWER_PIN 5   //This pin controls the left motor power
#define RIGHT_MOTOR_DIR_PIN 7    //This pin controls the right motor direction
#define RIGHT_MOTOR_POWER_PIN 6  //This pin controls the right motor power

int leftMotorDirection = FORWARD;   //Controls whether the left motor runs forward or in
reverse
int leftMotorPower = 125;           //Controls how fast the left motor runs (from 0 - 255)
int rightMotorDirection = FORWARD;  //Controls whether the right motor runs forward or in
reverse
int rightMotorPower = 125;          //Controls how fast the right motor runs (from 0 - 255)
```

```
void setup() {
  Serial.begin(9600);

  pinMode(LEFT_MOTOR_DIR_PIN, OUTPUT);
  pinMode(LEFT_MOTOR_POWER_PIN, OUTPUT);
  pinMode(RIGHT_MOTOR_DIR_PIN, OUTPUT);
  pinMode(RIGHT_MOTOR_POWER_PIN, OUTPUT);

  waitUntilButtonIsPressed();
}

void loop() {
  //Run each of the motors
  runLeftMotor(leftMotorDirection, leftMotorPower);
  runRightMotor(rightMotorDirection, rightMotorPower);
}

/* Runs the left motor in the direction
 * given, at the specified power */
void runLeftMotor(int direction, int power){
  digitalWrite(LEFT_MOTOR_DIR_PIN, direction);
  analogWrite(LEFT_MOTOR_POWER_PIN, power);
}

/* Runs the right motor in the direction
 * given, at the specified power */
void runRightMotor(int direction, int power){
  digitalWrite(RIGHT_MOTOR_DIR_PIN, direction);
  analogWrite(RIGHT_MOTOR_POWER_PIN, power);
}

/* Waits until the start button is pressed
 * to continue */
void waitUntilButtonIsPressed() {
  while (analogRead(START_BUTTON) > 500) {
    delay(100);
  }
}
```

3. Verify and Upload the program to your board. When it is finished, press the S2 button to start the robot. You should see both wheels moving forward at about half power. If one, or both wheels are moving in reverse, the motor polarities have been mixed up. Identify the wheel(s) not moving in the right direction, and switch its red and black wires on the board's motor connectors.

4. You should notice that the code to control each motor is identical to the last program, only with different variable and function names. Change the rightMotorPower to different analog value, change leftMotorDirection to REVERSE, and re-upload the sketch. You should notice a change in speed from the right wheel, and the left wheel now spins in reverse.
5. Play with each of the direction and power variables as much as you like.

Step Three: Driving the robot in a straight line

1. We're now ready to give the robot specific movement commands. Open up your double motor program if it isn't open already.
2. Set both motor directions to FORWARD and to a medium power (between 150 and 200). Make sure the left and right motor powers are the same. Upload this code to the Arduino board.
3. Unplug the USB cable and set your robot on the floor. Plug in the battery and press the S2 button to start the robot. It should drive in a (reasonably) straight line. It's normal for the robot to curve a little bit, there is typically some difference between two motors at the same power. If your robot is turning significantly to either side, however, you may have a bad motor. See your instructor about replacing it.
4. Change the program so that it now drives backwards in a straight line. Test your results

Step Four: Swiveling the robot

1. You can make the robot swivel by making one wheel go forward, and the other go in reverse. Set both motors to a medium power, set the left motor direction to FORWARD, and the right motor direction to REVERSE.
2. Upload and test this program. Notice which way the robot spins?
3. Now, write a program that can make the robot swivel in the other direction. Test this code.

Step Five: Turning the robot

1. You can turn the robot by having one wheel move faster than the other in the same direction. Set both motors to FORWARD, set the left wheel to a low power (100 - 150), and set the right wheel to a high power (200 - 250).
2. Upload and test this program. Your robot should be turning significantly to the left.
3. Bring the left wheel up to a medium power (150 - 200) and re-test it. Your robot should still be turning, but less sharply now. The degree of your curve depends on the difference between your left and right wheel powers.
4. Now write and upload sketches that make the robot:
    a. Turn sharply to the right
    b. Turn softly to the left in reverse
    c. Stand still

5. When you have successfully tested the above programs, proceed to the next part of the lab.

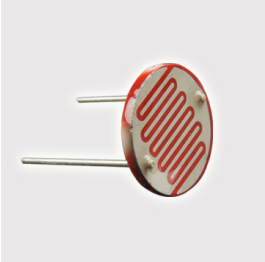
Part VI: Mounting and Reading a Photoresistor

Goals:

- Attach the two photoresistors to the robot
- Become familiarized with reading inputs from the Arduino board
- Detect the amount of light present using a photoresistor
- Use the data gathered by the photoresistor to produce a simple output

Background

We've covered the output devices we'll be using for this project. Now lets take a look at our input device: a photoresistor, or light sensor.



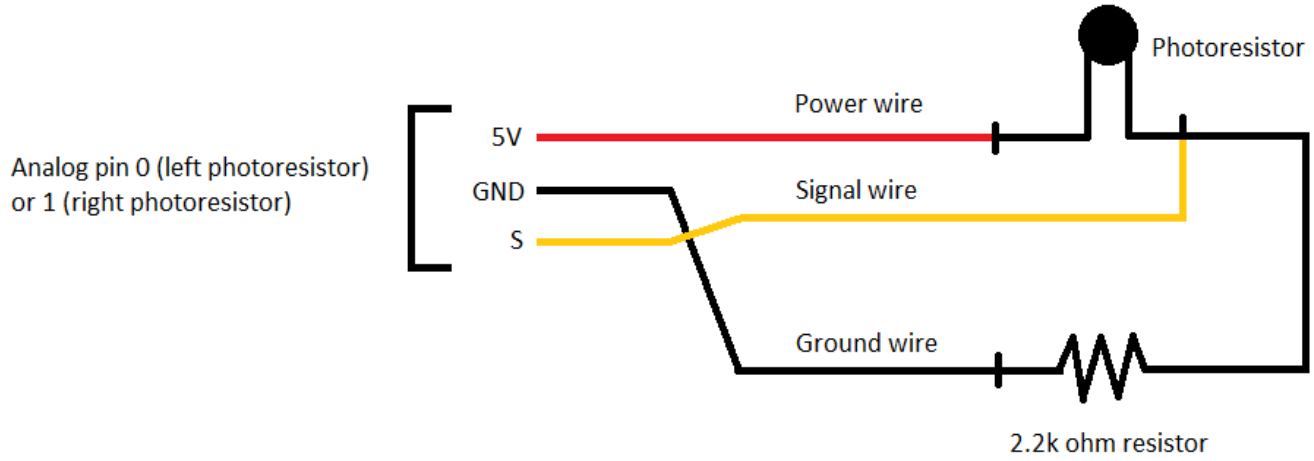
A light sensor is capable of detecting a light source and determining how strong and far away that light source is. The sensors we will be using are made to detect visible light, like your eyes, but there are several other sensors that are used to detect other kinds of light. For example, an IR (infrared) sensor can be used to detect infrared light, and an UV (ultraviolet) sensor can detect ultraviolet light sources, as their names suggest. Most light sensors are photocells, which are sensors that react to the presence of light or electromagnetic energy. Photocells come in many shapes and forms, such as solar panels that use light to generate electricity, to cryogenic detectors, which are extremely sensitive electromagnetic radiation. This particular light sensor is a photoresistor, a type of photocell which changes its resistance according to the amount of light it detects. Photoresistors are used for anything that needs to adjust to changes in light autonomously. Street lights and nightlights typically use photoresistors to determine when to turn themselves on. A camera also typically uses a photoresistor to determine if the flash is needed or not, among other things.

On robots, photoresistors, and other kinds of light sensors, are used to give the robot "vision", and help enable it to function autonomously. The light sensor can either be used independently or in combination with other sensors to perform a certain application. For example, a light sensor and a proximity sensor could be used to build a robot that avoids collisions (with the proximity sensor), and getting stuck under furniture (with a light sensor placed on top of the robot).For this project, we will be using two light sensors, positioned to give the robot a full 180 degree field of photosensitivity. The robot will be then be able to detect a

light source, such as a flashlight or an LED, and determine how far away it is, based on the source's intensity. The software will then be able to use that data to either move toward the light or away from it, depending on the user's configuration.

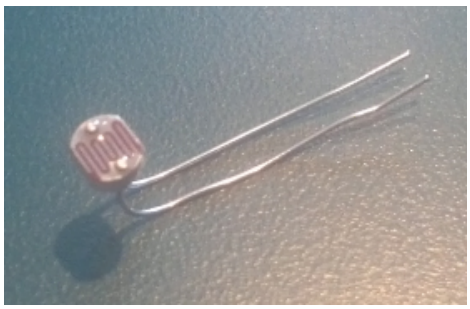
Step One: Attaching the photoresistors

1. Familiarize yourself with the circuit required to run and read a signal from the photoresistor. You'll be setting this circuit up on your breadboard: one leg connects to the 5V power supply, while the other connects to your signal wire, which itself is connected to the resistor / ground and your signal pin (which is either Analog In 0 or 1, depending on which side the sensor is on).

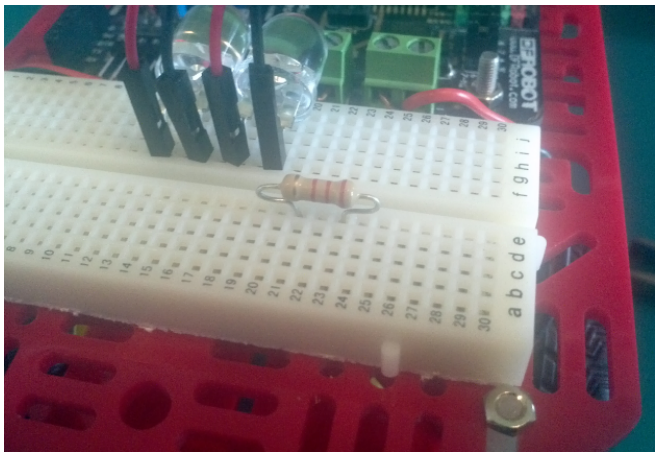


2. Using your needle-nose pliers, bend the ends of the 2.2k ohm resistors as shown below:

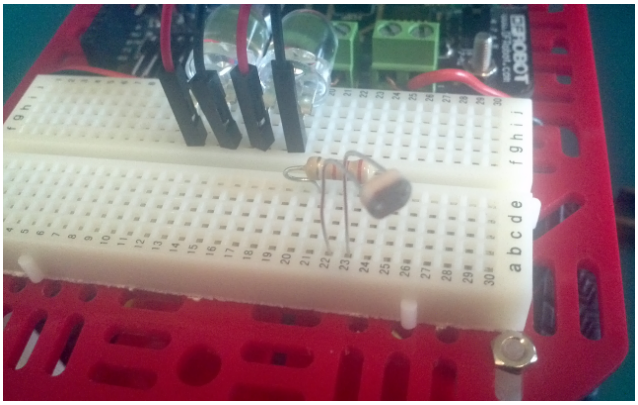
3. Carefully bend the leads of the photoresistors as shown below:



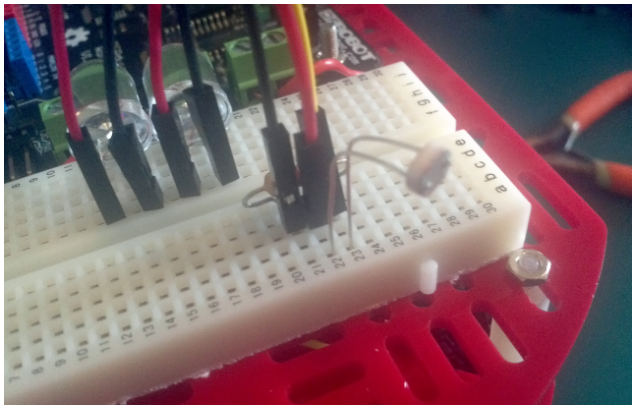
4. Insert the resistor into the breadboard as shown below:



5. Insert the photoresistor into the breadboard as shown below:

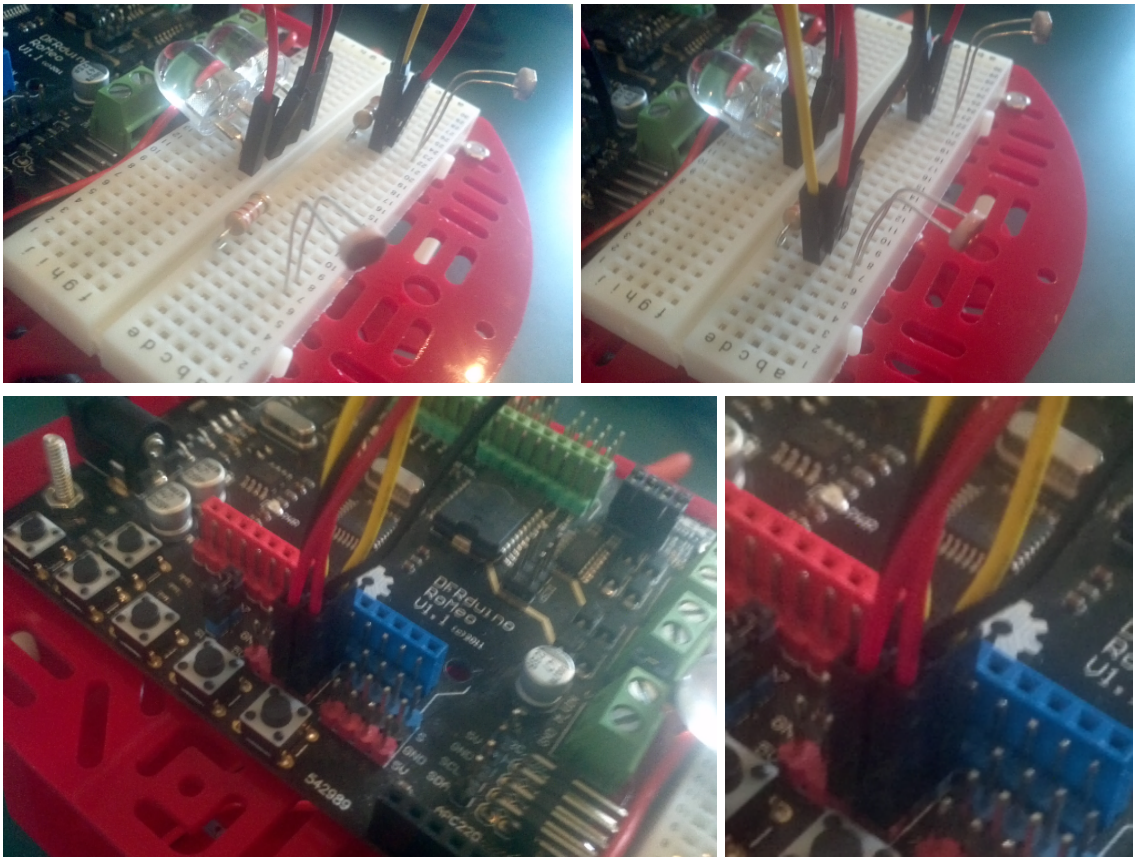


6. Insert the appropriate jumper wires into the breadboard as shown below. It should be set up to match the diagram from part (a). If you've placed the resistor and photoresistor as shown in steps (c) and (d), this means that power should connect in the middle, with the signal wire connecting on the right and the ground wire connecting on the left (if viewed from the front, as in the image below).



7. Connect the power jumper wire to the pin labeled “VCC” on analog port 0, connect the ground jumper wire to the pin labeled “GND” on analog port 0, and connect the signal jumper wire to the pin labeled “S” on analog port 0.

8. Using the second resistor and photoresistor, repeat steps (a) through (f) on the right side of the breadboard, and connect the appropriate jumper wires to analog port 1. See below:



Step Two: Reading from the photoresistor

1. Open up Arduino and plug in your board
2. Copy and Paste the following code into your sketch area

```
/****  Reading the Photoresistors Example  ****/

//Define Photoresistor Pins
#define LEFT_PHOTORESISTOR_PIN A0   //The left light sensor pin
#define RIGHT_PHOTORESISTOR_PIN A1  //The right light sensor pin

//Variables
int leftLight;   //How much light the left sensor sees
int rightLight;  //How much light the right sensor sees
```

```
void setup() {
  Serial.begin(9600);

  //Remember: We need to set the light sensors as inputs
  pinMode(LEFT_PHOTORESISTOR_PIN, INPUT);
  pinMode(RIGHT_PHOTORESISTOR_PIN, INPUT);
}

void loop() {
  //Read the photoresistors
  leftLight = analogRead(LEFT_PHOTORESISTOR_PIN);
  rightLight = analogRead(RIGHT_PHOTORESISTOR_PIN);

  //Report the data
  sendDataToSerialMonitor();

  //Wait a little
  delay(500);
}

//Sends the sensor data to the Serial Monitor
void sendDataToSerialMonitor() {
  Serial.print("Left Light: ");
  Serial.print(leftLight);
  Serial.print(" , Right Light: ");
  Serial.println(rightLight);
}
```

3. Verify and Upload your program to the board. Go to Tools => Serial Monitor to view the Serial Monitor.
4. You should see a list of values from the left and right light sensors. The photoresistors return a value between 0 - 1023, with 0 meaning it can't detect any light, and 1023 representing its maximum light capacity. Try to change these values: bring the robot near a window or cover one of them with your hands - how do the values change?

Step Three: Mimicking a light source

1. Save this program and open up a new Arduino sketch
2. Copy and paste the below code into your sketch area:

```
/****  Light Mimicker Example  ****/

//Define Photoresistor Pins
```

```
#define LEFT_PHOTORESISTOR_PIN A0
#define RIGHT_PHOTORESISTOR_PIN A1

//Define LED pins
#define LEFT_LED_PIN 10
#define RIGHT_LED_PIN 11

//Constants
#define AMBIENT_LIGHT 400

//Variables
int leftLight;
int rightLight;
int leftLightAnalog;
int rightLightAnalog;

void setup() {
  Serial.begin(9600);

  //Set light sensor pins as inputs
  pinMode(LEFT_PHOTORESISTOR_PIN, INPUT);
  pinMode(RIGHT_PHOTORESISTOR_PIN, INPUT);

  //Set LED pins as outputs
  pinMode(LEFT_LED_PIN, OUTPUT);
  pinMode(RIGHT_LED_PIN, OUTPUT);
}

void loop() {
  //Read the photoresistors
  leftLight = analogRead(LEFT_PHOTORESISTOR_PIN);
  rightLight = analogRead(RIGHT_PHOTORESISTOR_PIN);

  //Convert sensor values to analog values
  leftLightAnalog = convertToAnalog(leftLight);
  rightLightAnalog = convertToAnalog(rightLight);

  //Output to LEDs
  analogWrite(LEFT_LED_PIN, leftLightAnalog);
  analogWrite(RIGHT_LED_PIN, rightLightAnalog);
}

/* Convert the number you get from the sensor
 * (0 - 1023) into an analog value the LED can
 * use (0 - 255) */
int convertToAnalog(int sensorValue) {
```

```
  int constrainedValue = constrain(sensorValue, AMBIENT_LIGHT, 800);
  return map(constrainedValue, AMBIENT_LIGHT, 800, 0, 255);
}
```

3. Verify and Upload this code to the board. This program simply takes the value the photoresistor reads, converts to a 0 - 255 number, then analog writes it to the corresponding LED. Your LEDs should now “mimic” what the light sensors see, getting brighter as a light source approaches, and darker as the amount of visible light decreases
4. The AMBIENT_LIGHT constant will need to be changed according to the amount of ambient light in the classroom. If there is a lot of ambient light where you are, change it to a high value (between 300 - 400). If there is little ambient light where you are, change it to a low value (between 100 - 200).
5. Once again, change the amount of light reaching the photoresistors yourselves. The LEDs should change accordingly.
6. Ask a friend to load their LED blink or fade software unto their machine. Bring their robot close enough for the photoresistors to detect their LEDs. Do your LEDs ‘mimic’ theirs? Note: If you are receiving inadequate results, try reducing the amount of ambient light in the classroom if at all possible. Consult your instructor about this.

Part VII: Putting it all Together

Goals:

- Use the skills you've learned to construct a photophobic / photophilic robot
- Manipulate the software to customize your robot
- Conduct a classroom Braitenberg AI Experiment

Background:

We now have all the components we need to complete the final robot. At the end of this lab, you will have a robot that either drives toward, or away from light sources. To accomplish this, we will be reading data from the photosensors, performing calculations on that data, and writing values to the motors. We will also turn the two LEDs on so that the robots may follow or run away from each other. Once everyone in the class has finished their robot, you will partake in a classroom wide experiment exploring the perceived behavior of simple, artificially intelligent machines.

Step One: Making the photophilic robot

1. Open the Arduino IDE and plug in your board
2. Copy and paste the following code into your sketch area:

```
/****  Braitenberg Machine Example  ****/

//Define Photoresistor Pins
#define LEFT_PHOTORESISTOR_PIN A0
#define RIGHT_PHOTORESISTOR_PIN A1

//Define LED pins
#define LEFT_LED_PIN 10
#define RIGHT_LED_PIN 11

//Define Motor Pins
#define LEFT_MOTOR_DIR_PIN 4
#define LEFT_MOTOR_POWER_PIN 5
#define RIGHT_MOTOR_DIR_PIN 7
#define RIGHT_MOTOR_POWER_PIN 6

//Define Motor directions
#define FORWARD HIGH
#define REVERSE LOW

//Define start button
#define START_BUTTON A7
```

```
//Constants
#define IS_PHOTOPHOBIC true    //Determines whether robot follows or runs away from
light
#define LIGHT_SENSITIVITY 50    //How sensitive the robot is to light
#define MOTOR_SENSITIVITY 50    //How much the robot turns in response to light

#define MINIMUM_BASE_MOTOR_POWER 125    //Motor Power when light source is
farthest
#define MAXIMUM_BASE_MOTOR_POWER 175    //Motor Power when light source is
closest

//Variables
int leftLightValue;     //How much light the left sensor sees
int rightLightValue;    //How much light the right sensor sees
int leftMotorPower;     //Determines power output to the left wheel
int rightMotorPower;    //Determines power output to the right wheel

void setup() {
  //Start communication with Serial Monitor
  Serial.begin(9600);

  //Set input pins
  pinMode(LEFT_PHOTORESISTOR_PIN, INPUT);
  pinMode(RIGHT_PHOTORESISTOR_PIN, INPUT);

  //Set output pins
  pinMode(LEFT_LED_PIN, OUTPUT);
  pinMode(RIGHT_LED_PIN, OUTPUT);
  pinMode(LEFT_MOTOR_DIR_PIN, OUTPUT);
  pinMode(RIGHT_MOTOR_DIR_PIN, OUTPUT);
  pinMode(LEFT_MOTOR_POWER_PIN, OUTPUT);
  pinMode(RIGHT_MOTOR_POWER_PIN, OUTPUT);

  //Set the initial value of variables
  initVars();

  //Turn LEDs on
  digitalWrite(LEFT_LED_PIN, HIGH);
  digitalWrite(RIGHT_LED_PIN, HIGH);

  //Wait until start button is pressed
  waitUntilButtonIsPressed();
}

void loop() {
```

```
//Read Photoresistor Values
leftLightValue = analogRead(LEFT_PHOTORESISTOR_PIN);
rightLightValue = analogRead(RIGHT_PHOTORESISTOR_PIN);

/* Calculate minimum light for movement using the
 * light sensitivity constant. */
int minimumLightForMovement = 400 - map(LIGHT_SENSITIVITY, 0, 100, 0, 400);

/* If the amount of light for both sensors is too low:
 * If its photophobic, stop completely.
 * If its photophilic, swivel and search for light sources. */
if (leftLightValue < minimumLightForMovement &&
rightLightValue < minimumLightForMovement) {
  if (IS_PHOTOPHOBIC) {
    //Stop the robot
    runLeftMotor(FORWARD, 0);
    runRightMotor(FORWARD, 0);
  } else {
    //Swivel the robot
    runLeftMotor(FORWARD, 200);
    runRightMotor(REVERSE, 200);
  }
}

/* Otherwise, there is a notable light
 * source, and we should go follow it */
else {
  /* Determine and assign base motor power: More
   * total light means the robot should move faster */
  int totalLight = leftLightValue + rightLightValue;
  totalLight = min(totalLight, 800);
  int baseMotorPower = map(totalLight, 0, 800,
  MINIMUM_BASE_MOTOR_POWER, MAXIMUM_BASE_MOTOR_POWER);

  /* Caluclate motor difference value using motor sensitivity
   * constant and the difference between the left and right
   * light sensors */
  int difference = leftLightValue - rightLightValue;
  int motorPowerDifference = convertToAnalog(difference) *
  MOTOR_SENSITIVITY / 50;

  //Calculate the final motor powers
  if (IS_PHOTOPHOBIC) {
    leftMotorPower = boundMotorPowers(baseMotorPower + motorPowerDifference);
    rightMotorPower = boundMotorPowers(baseMotorPower - motorPowerDifference);
  } else {
```

```
    leftMotorPower = boundMotorPowers(baseMotorPower - motorPowerDifference);
    rightMotorPower = boundMotorPowers(baseMotorPower + motorPowerDifference);
   }

   //Write values to the motors
   runLeftMotor(FORWARD, leftMotorPower);
   runRightMotor(FORWARD, rightMotorPower);

   //Report values to the Serial Monitor
   sendDataToSerialMonitor();
  }

  //Brief delay
  delay(50);
}

//Set the initial value of variables
void initVars() {
  leftLightValue = 0;
  rightLightValue = 0;
  leftMotorPower = 0;
  rightMotorPower = 0;
}

/* Runs the left motor in the direction
 * given, at the specified power */
void runLeftMotor(int direction, int power){
  digitalWrite(LEFT_MOTOR_DIR_PIN, direction);
  analogWrite(LEFT_MOTOR_POWER_PIN, power);
}

/* Runs the right motor in the direction
 * given, at the specified power */
void runRightMotor(int direction, int power){
  digitalWrite(RIGHT_MOTOR_DIR_PIN, direction);
  analogWrite(RIGHT_MOTOR_POWER_PIN, power);
}

//Keep Motor Powers within bounds
int boundMotorPowers(int value) {
  return constrain(value, 100, 255);
}

/* Convert the number you get from the sensor
 * (0 - 1023) into an analog value the LED can
 * use (0 - 255) */
```

```
int convertToAnalog(int sensorValue) {
  return map(sensorValue, 0, 1023, 0, 255);
}

//Sends out values to the Serial Monitor
void sendDataToSerialMonitor() {
  Serial.print("Left Light: ");
  Serial.print(leftLightValue);
  Serial.print(" , Right Light: ");
  Serial.print(rightLightValue);
  Serial.print(" , Left Motor: ");
  Serial.print(leftMotorPower);
  Serial.print(" , Right Motor: ");
  Serial.println(rightMotorPower);
}

/* Waits until the start button is
 * pressed to continue */
void waitUntilButtonIsPressed() {
  while (analogRead(START_BUTTON) > 500) {
    delay(100);
  }
}
```

3. Examine the program. Most of the functions and variables should look familiar. Don't worry if you can't understand the entire thing: there are several things you haven't seen before also. Refer to the comments if you don't know what something does.
4. Upload the code to your board, unplug your robot, and set it down on the ground. Acquire some sort of light source (if someone in your group has a smartphone, they likely have a flashlight app or something similar. That will be sufficient).
5. Press the S2 button on your robot then hold the light source a small distance away (one or two feet) away from the robot. You should be able to lead it around the room by moving the flashlight where you want it to go (so long as the light can still read one of the photoresistors). Notice that the robot speeds up and turns harder the closer the light source is.
6. Return to your program and plug your robot back in.
7. Find the IS_PHOTOPHOBIC constant. You can alter these to change how your robot responds. The IS_PHOTOPHOBIC constant changes whether your robot runs from light, or follows it. When it is set to false, it is a photophilic robot, and drives toward the light as it does now. When it is set to true, it is a photophobic robot, and turns away from light.
8. Change IS_PHOTOPHOBIC to true and re-upload your program. When you test your robot again, you should find that it now turns away from your flashlight, and tries to find a dark place to stop.

9. Return to your program and change IS_PHOTOPHOBIC back to false. Now find the LIGHT_SENSITIVITY constant just under it. This controls how sensitive your robot is to light sources. It can be set to value from 0 - 100, where 0 is the least sensitive, and 100 is the most sensitive. In a room where there is a lot of ambient light, this value will need to be set lower, so your robot can distinguish actual light sources from the standing light. In areas where there is little to no ambient light, this value will need to be set higher, so your robot can more easily detect the light sources.
10. Set your LIGHT_SENSITIVITY very low (between 0 - 10) and test your robot. You will find that your robot only detects distinct light sources that are very close to it - it will not settle for any ambient light as a light source.
11. Set your LIGHT_SENSITIVITY very high (between 90-100) and test your robot. You should find that your robot now follows any slimmer of light, even perhaps light reflecting off a wall!
12. Return to your program and set the LIGHT_SENSITIVITY constant to your liking. Now find your MOTOR_SENSITIVITY constant. This can also be set between 0 - 100, only this controls how hard your robot tries to turn in order to follow a light source. On tile or similar floors, this constant may be set relatively low, since it is fairly easy to turn. On carpets, however, this value will need to be set higher, since the material makes it more difficult for the robot to turn. This will also need to be adjusted according to how strong your motors are. Don't set it too high, though, or your robot will spin out every time it attempts to turn. Set it too low, and it won't even turn at all.
13. Adjust the MOTOR_SENSITIVITY constant until you find a value that works for your robot.
14. Now, explore! Test your robot in different environments, with different amounts of light, and see how your robot performs. Does your robot successfully find the brightest light source if its photophilic? If it its photophobic, does it consistently find a dark place to hide? Adjust the constants until it does and you are comfortable with them.

Step Two: The Braitenberg experiment

1. When everyone in the class has constructed their robot, and is ready to proceed, it is time to perform the classic Braitenberg AI experiment.
2. Select one group in the class to have a photophobic robot. Everyone else should set their robot to be photophilic.
3. Your instructor will direct you to a closed off space with little to no ambient light. Have everyone randomly set their robots on the ground and switch on their robots.
4. Observe. How do the robots behave? Does something appear to occur between the photophilic robots and the lone photophobic robot?