Programming Activity – Resistive Sensors and Servos

The goal is to make sure that everybody knows how to connect a resistive sensor (potentiometer and light sensors are examples), how to calibrate those sensors by writing the values to the serial monitor, how to map those values to a sensible range of values, and how to use those values to update an output (like an LED or a servo).

Overview

There are five steps in today's exercise. They lead up to a circuit that moves a servo in response to changing light levels. Along the way we'll use example programs that are already in the Arduino software suite, but we'll modify them slightly along the way. The five steps are:

Step1: Connect a potentiometer (knob) to the analog input of the Arduino, and use it to control the flashing rate of an LED.

Step2: Replace the potentiometer with another resistive sensor – the cds light sensor. Now the changing light level will modify the flashing rate.

Step3: Use the "serial monitor" to see what values are being returned by the light sensor. By writing down the values you can calibrate the light sensor.

Step4: Use the "map" function in the Arduino library to take the range of values that are returned by the sensor, and re-map those values to a different range of values that is a more useful range.

Step5: Replace the flashing LED with a moving servo. The servo requires a slightly different way of describing things in the code using the Servo library and "member functions" in the code. The calibrated light sensor is used to control how far the servo moves.

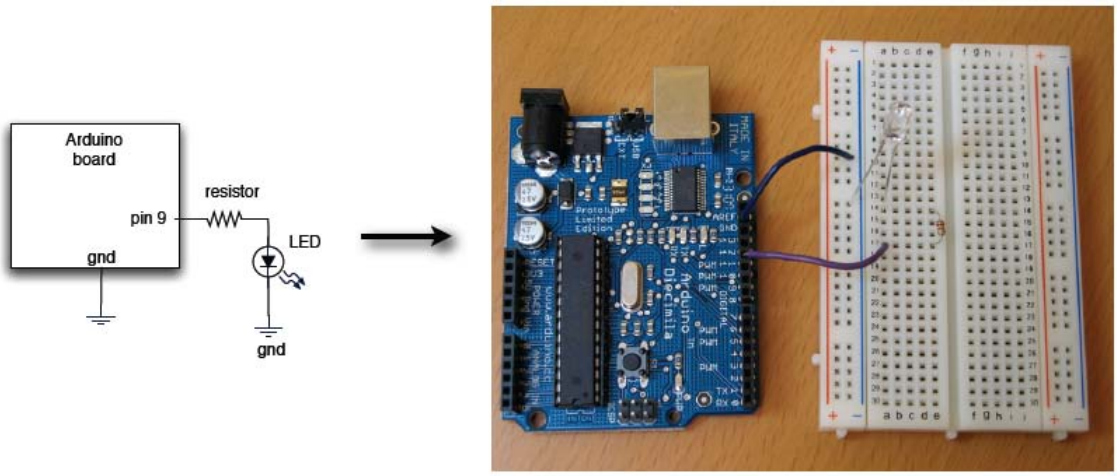
Now you should have enough tools to work on Assignment 3. For this assignment you should use some sort of sensing (switches, pots, light sensors) to control some sort of output (LEDs, servos). The work should have some relationship to drawing or mark making. What that relationship is, is up to you.

Details

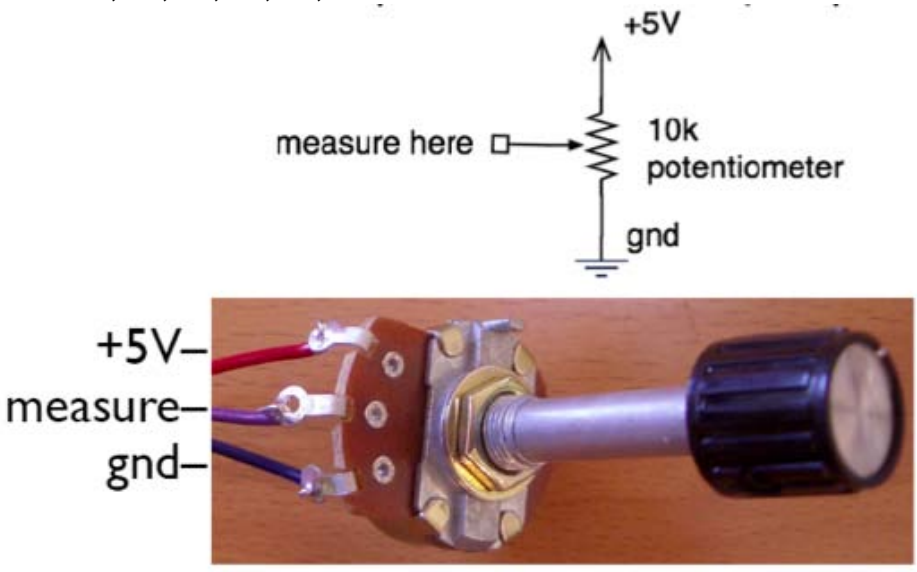
Step1: connect a potentiometer (pot, or knob) to an analog input of the Arduino and learn to use it to control the flashing rate of an LED.

- Use the following components: an LED, and a 220Ω-330Ω resistor for the LED, and a potentiometer.
- Connect the LED and current-limiting resistor to one of the digital outputs of the Arduino. This figure shows using pin 9 as an output pin. That's a fine choice, but you can choose a different pin if you like. Choose one that is a

"PWM" pin though. (Actually, the schematic shows pin9, but I think the photo uses pin 11!)

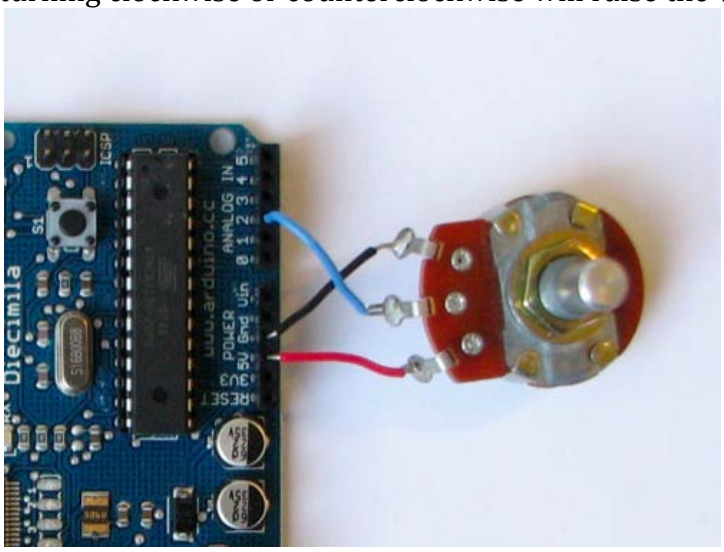


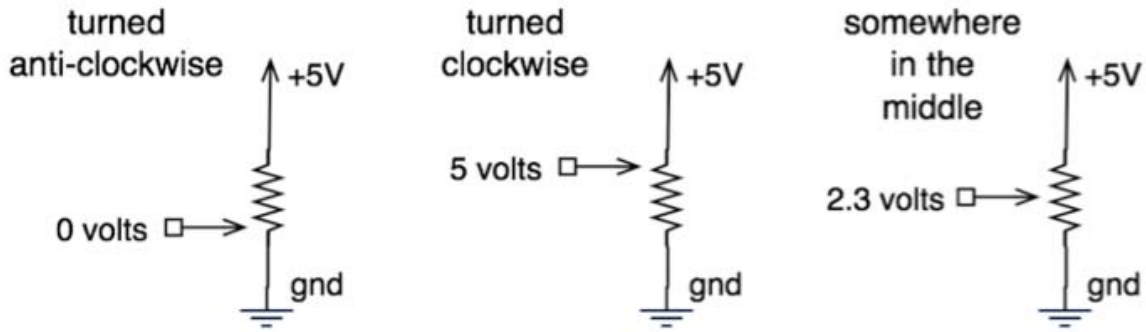
- Connect the potentiometer to an analog input of the Arduino. A potentiometer is a resistor that has a third terminal that can "tap" the resistor at different spots. You usually change where the tap hits the resistor by turning a knob. There are linear "slider" pots too. If you put the endpoints of the pot at 5v and gnd, then when you measure at the tap, you get some analog voltage between 5v and 0v. The analog inputs of the Arduino are on the side of the board nearest the processor chip. There are 6 analog inputs named A0, A1, A2, A3, A4, and A5.



- Our pots are also 10k pots, but they're smaller and made of plastic. The terminals are in the same arrangement though: the measurement point is in the middle, and 5v and gnd are on the outside. It doesn't matter which one you connect to 5v and which one to gnd. The only difference is whether

turning clockwise or counterclockwise will raise the voltage.





- In the Arduino programming environment, load the **Examples→Analog→AnalogInput** program. Modify it to use the LED pin and the analog pin you chose to use. Note that the pot value is read using the `analogRead(<pinnumber>);` function. This returns a numeric value between 0 and 1023. The value that's read from the pot is used to modify the delay of the flashing LED. You can change the value of the delay (change the number that's returned from the pot) by turning the knob on the pot.

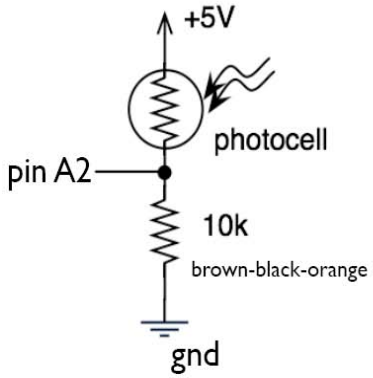
- This is a general technique that uses a pot/knob to return a range of numbers that can be used to modify the behavior of some aspect of the program. Because of the way that the analog voltages are sampled, the numbers you get from a pot are always between 0 and 1023, no matter what size pot you use. 10kΩ is a good general pot size for these types of circuits.

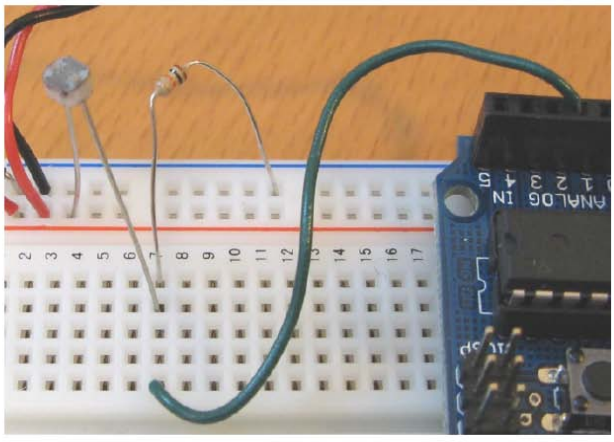
Step2: connect a light sensor to an analog input of the Arduino and have it control the flashing delay of an LED instead of the pot. This swaps out the pot for another circuit component that acts like a variable resistor. This is a cds light sensor (cds stands for the material it's made of – Cadmium Sulfide). A cds sensor is photoresistive – the resistance between the terminals changes as the light intensity changes. The brighter the light, the lower the resistance.

- Use the following components: A light sensor and a resistor between 330Ω and 10kΩ. Light sensors look like flat-topped components with a squiggly line on the top. The squiggly line is the photoresistor.

- Now connect the light sensor to one of the analog pins of the Arduino. The light sensor (like many resistive sensors) needs to be connected with a series resistor just like the LED. In this case you may be able to use a larger valued resistor like a 10kΩ resistor. In general, if the light sensor you have is small (¼" or less) you'll probably want to use a small resistor like 330Ω or 470Ω. If it's larger like ¼" to ½: across on the top, you can use a larger resistor up to 10kΩ. The resistor value isn't critical. It just changes slightly the range of values you'll get back from the sensor (see Step3). The circuit in the figure uses the A2 analog pin. You can pick any of the analog pins you like.

Photocell Circuit





- Now rerun the **AnalogInput** program. Shading the light sensor will change the flashing rate of the LED by changing the resistance of the photocell, and thus changing the value that is read by the `analogRead(<pinnumber>);` function.

- But, the range of flashing rates is probably not as wide as with the pot. This is because the range of resistances is not as wide with the photocell as with the pot, so the numbers you read back are not all the way to 0 and not all the way to 1023.

- This is a great way to change the behavior of your program based on an environmental condition like the light levels. There are many resistive sensors that do the same thing – change their resistance based on some environmental condition - like temperature, Co2 concentration, pressure, strain (bending), humidity, etc.

Step3: Use the serial library to see what range of values you're getting from the light sensor. This is one way of "calibrating" your sensor – figuring out what the range of values is that you're getting from the sensor. You might be able to figure this out with details about the component, but it's often easier just to use this calibration procedure to see what values you're actually getting.

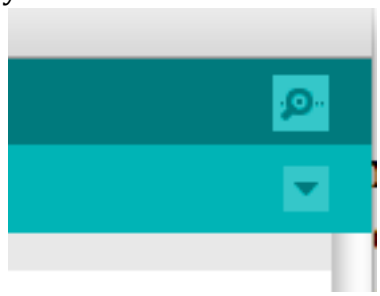
- Because you probably don't know what the resistance values you're getting out of your light sensor, it's hard to predict what values you'll get out of the `analogRead(<pinnumber>);` function.

- So, one calibration technique is just to print out the values you're getting. If you do this every 1/10 of a second or so, and print the number, you'll be able to see what values you can expect as you shade and unshade the sensor.

- To print values, we use the "Serial" library in the Arduino programming language. This library lets you print things to a window on the Mac (or PC) that you're connected to through the USB cable. The Serial library has three parts:
    - Initialize the library with `Serial.begin(<baudrate>);` This sets the speed that characters are sent. Baud rates are restricted to certain numbers: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. The larger the number, the faster individual characters are sent to the computer. In practice, 9600 is fine for our purposes, and is a standard baud rate. Others are fine too.
    - You can send values to the Mac/PC window (called the "Serial Monitor") using the `Serial.print(<value>);` or

`Serial.println(<value>);` function. The first one prints the value to the serial monitor, the second prints the value and then starts a new line. The value can be a variable (in which case the value in that variable is printed), or a number or a string in quotes in which case that number or string is printed.

- You open the serial monitor on the Mac/PC using the icon in the upper right of the Arduino programming environment. You need to make sure that the serial monitor uses the same baud rate that you used in the `Serial.begin(<baudrate>);` function.



- Now open the **Examples→Basics→AnalogReadSerial** program. This is a very simple program that just reads a value from analog pin A0 (you can change this if you have your light sensor connected to different analog pin), and prints the value to the serial monitor. I like to add some delay to the main loop. I'd add `delay(100);` to the loop so that you only update things every 1/10 of a second (100msec).

- You could also add other Serial.print() statements to format the printed values in a little nicer way. Try:
  Serial.print("Value from the sensor is ");
  Serial.println(sensorValue);
  delay(100);

- As you shade and unshade the light sensor, you'll see the range of values that you're likely to see in this environment. Of course, you'll see different values if you took this outside where it's brighter, or in a darker room. Make a note of the likely highest and likely lowest value you'll get in our room.

Step 4: Use the "map" function to remap the values you're getting from the light sensor to something that makes sense. The map function takes a range of expected values (the low and high values you wrote down in step3), and maps them to a new range. For examples, if you expect (based on the calibration) to get values that range from 390 to 846, but you want to have the value you get when it's dark be around 100 and the value you get when it's bright to be around 1000, you could use map to do this.

- The map function is:
  `map(value, inLow, inHigh, outLow, outHigh);`

- In the previous example, the map call would be:
  `map(sensorValue, 390, 846, 100, 1000);`

- This process is technically called "interpolation". The numbers in the original range are interpolated so that they fall within the new range, but in the same relative position in the range. That is, a number that is around 1/3 of the way through the original range, will be about 1/3 of the way through the new range. But, that will fall on a different number because the range is different.

- This `map` function returns a new value that is in the new range, so you need to set a variable to the value returned by this function. You can use the same value as the one you're mapping if you like:
  `sensorValue = map(sensorValue, 40, 235, 0, 255);`

- Load the program **Examples→Analog→AnalogInOutSerial**. This program reads the value from an analog input (like your light sensor), maps the value that you read in to the range 0-255, then prints out the original value, and the mapped value. It also sets the LED to a brightness value based on this mapped value.

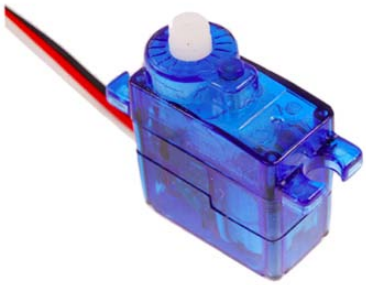
- Change the `map` function so that is uses the values you measured in Step3 as the `inLow` and `inHigh` values. Leave the `outLow` and `outHigh` values as 0 and 255. It's good to be a little conservative in the "in" values. That is, if you measured 40 and 235 as the values in Step3, use 35 and 245 as the "in" values in the map function. This will give up a little bit on the output range, but you won't go outside that range either.

- The reason the output mapped value is 0-255 is that this is the range that the `analogWrite` function will accept. This is the function that "dims" the LED by changing the pulse width modulation (PWM).

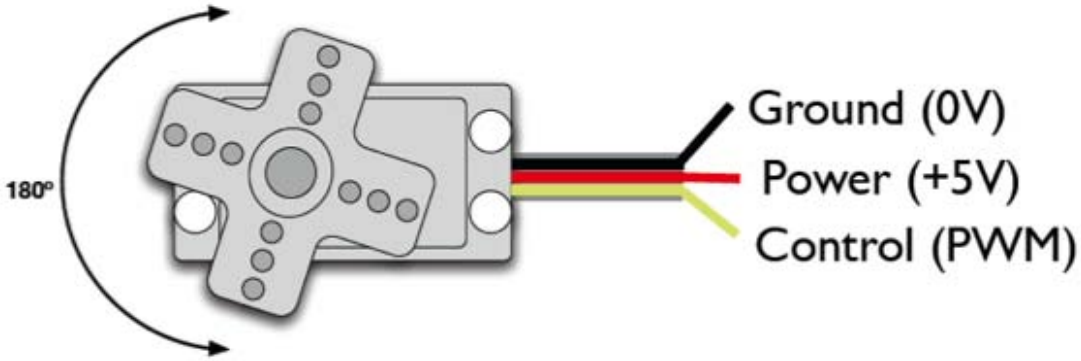
- If you still have your LED wired up, you'll see the LED get brighter and dimmer as you shade your light sensor.

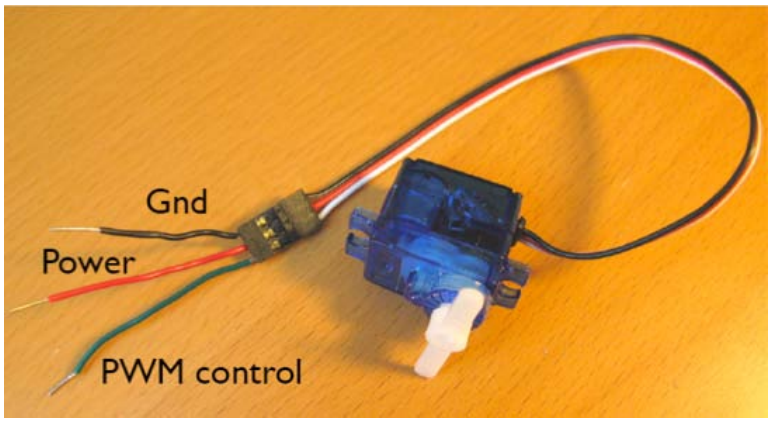
Step5: Learn to make a servo move - use the input from the light sensor to change the value of the servo position. A servo is a little motor that moves (rotates) in response to a change in the input signal. The input signal is a PWM signal, so you'd think you could use only PWM pins. But, the servo library code does something clever so that you can use any digital output pin you like.

- Servos look like little square box with wires coming out of one end, and a motor shaft that turns. Servos usually have a restricted range of motion of around 0-180 degrees.



- Connect your servo to your Arduino (use the breadboard). The three wires of the servo are power, ground, and signal. The colors aren't always consistent, but the ground is usually the darkest color, and on one edge of the cable. The power is in the middle, and the signal wire is on the other end, and is usually the lightest color.





- Connect the power to 5v, the ground to gnd, and the signal (PWM control) to one of the digital pins of the Arduino.

- In the Arduino programming environment, load the **Examples→Servo→Knob** program. This program will take an analog input from one of the analog pins, and use that value to move a servo. A few things to note about this program:
    - The servo library needs to be "declared" so that Arduino knows that you're using it: `#include<Servo.h>`
    - Each servo that you use needs to be defined as a "servo object" using the following statement: `Servo <servoname>;` In the Knob program there is only a single servo named "`myservo`." You can use any names you like - use a new name for each servo.
    - Each servo object will have a different name – and each servo object will have "member functions" associated with it. The most important member functions are "`attach`" and "`write`".
    - Member functions are called by putting the name of the function after the object name separated by a dot. Like `myservo.attach(9);` This takes the servo named "myservo" and attaches it to digital pin 9.
    - Each servo needs to be "attached" to a digital output pin using the "`.attach`" member function that is associated with each servo object.
    - Now in the `loop()` code you can change the servo's position to a new angle between 0 and 179 degrees using the .write member function:
      ```
      myservo.write(23); // move the servo to 23 degrees
      ```

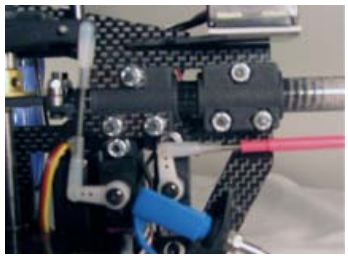
- Note that the knob program is written to imply that there's a pot (knob) providing the analog input, but we can use the light sensor as just another type of analog sensor. Make sure to update the Knob program to use the analog pin that your light sensor is connected to! In my case, that is:
  ```
  int potpin = A2;
  ```

- In this case, you probably want to modify the “map” function for the calibration values from Step3. That is, something like:
```
val = map(val, 40, 230, 0, 179);
```

- The output range of 0 to 179 is the full range of a servo – 0 degrees to 179 degrees (180 degrees counting from 0).

- Once the light sensor and servo are connected, you can run the “knob” program and watch the servo change position based on how much light is falling on the light sensor.

- Servos are a very convenient way to get things to move under program control. You can do simple rotational motion by connecting directly to the rotating hub of the servo, or linear(ish) motion by connecting a wire to a “horn” that is attached to the hub.

Step6: *Assignment #3* – use some environmental inputs (light sensor, pot, switches) and some output devices (LEDs, servos), and a program on the Arduino to make something interesting. It should relate in some way to drawing or mark making. You may use any other materials you like in constructing your piece.