Lab 12 – Connecting Processing and Arduino

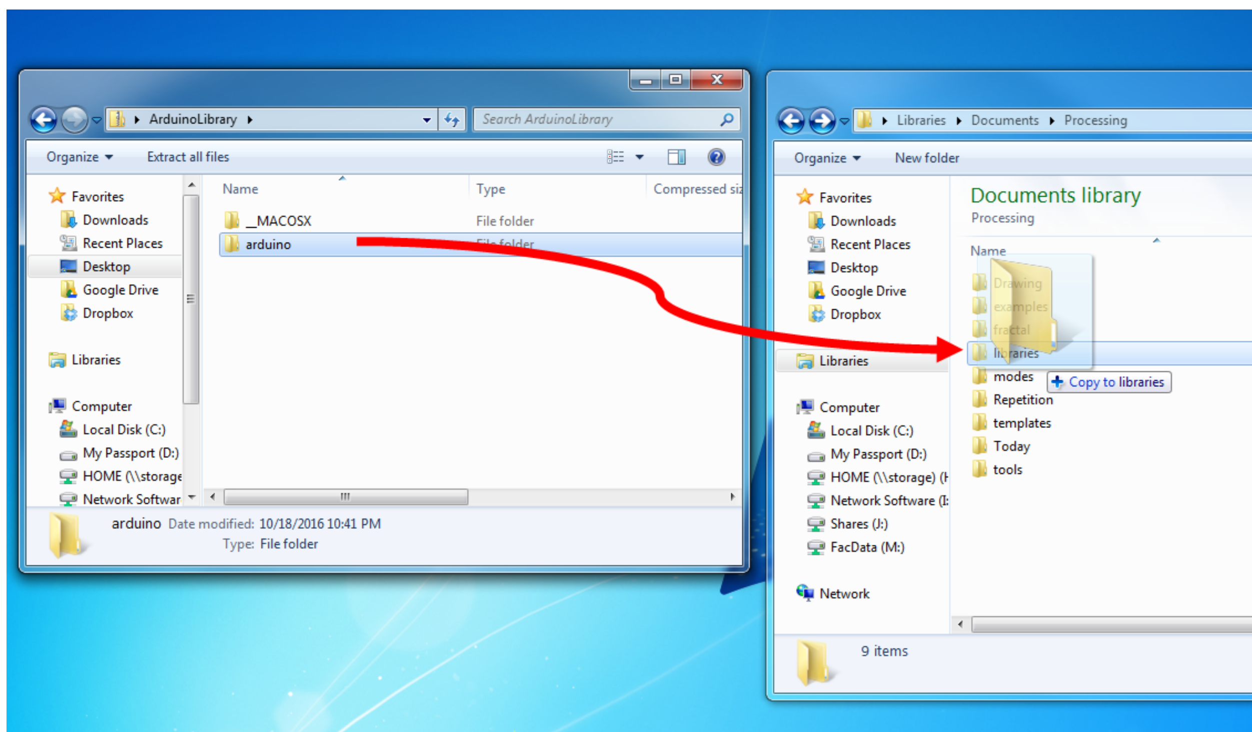
Overview

In the previous lab we have examined how to connect various sensors to the Arduino using Scratch. While Scratch enables us to make simple Arduino programs, it is limited to only communicate with the Arduino itself. To expand the range of possible Arduino interactions, we will access the Arduino directly from a Processing sketch. This will enable you to incorporate animation, sound, and other visual elements with physical input or output.

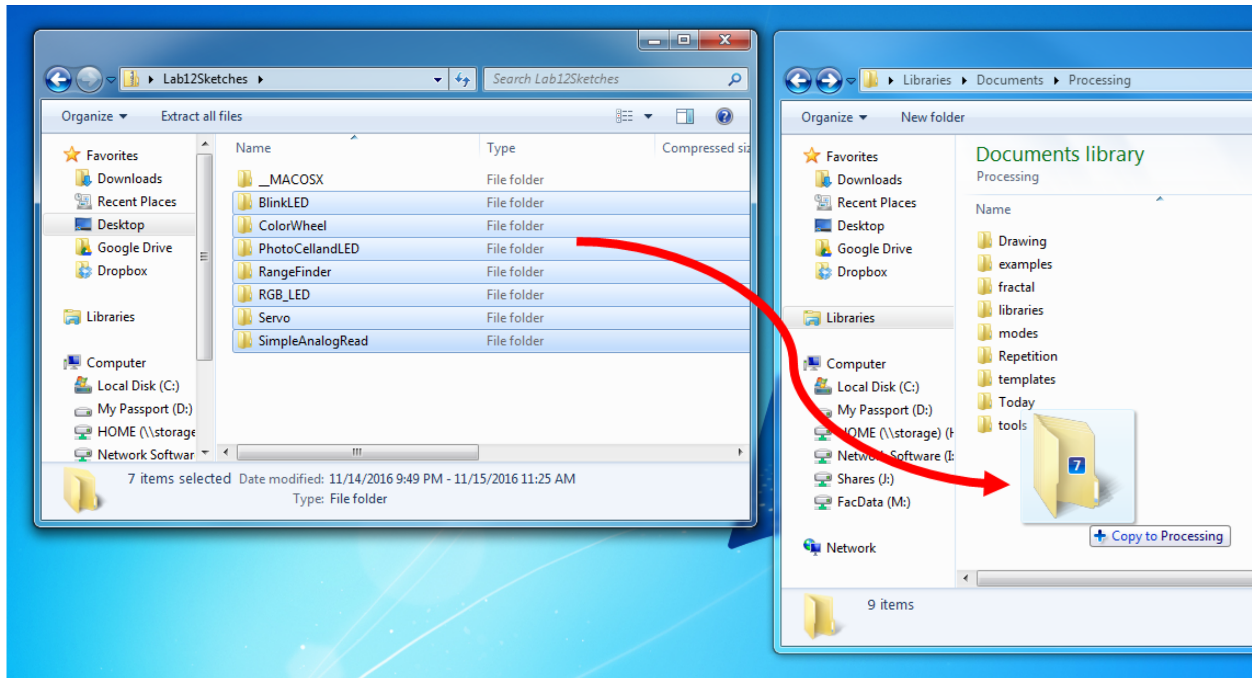
Setup

Download the Arduino library (link) for Processing and the Lab 12 sketches (link).

To install the Arduino library, open the ArduinoLibrary.zip file and drag the Arduino folder into your **Libraries** folder within Processing:



To install the sketches for today, open the Lab12Sketches.zip folder and drag all of the contents into your **Processing sketchbook**.



Part 1: Connecting to Your Arduino

When using the Arduino library in Processing you can directly communicate with the Arduino by turning on/off pins and requesting different values. However, before you perform these activities you must first establish a connection to the Arduino.

Load up sketch “Blink LED”. Examine the setup() loop of the sketch as shown in Figure 1.

```
Arduino arduino;
void setup()
{
  //print out a list of Arduino serial ports
  String[] ports = Arduino.list();
  for(int i=0;i<ports.length;i++)
  {
    println("Port "+i+":\t"+ports[i]);
  }

  //setup your arduino as port enumerated as 3
  arduino = new Arduino(this, Arduino.list()[3], 57600);
```

Figure 1: Create an Arduino object and determine which port is connected

The first line *Arduino arduino* creates an Arduino object that we will talk to. The subsequent lines print out the different serial ports on the computer. You must select the serial port that your computer is actually connected. This can be challenging as the output is not human-

readaes an Arduino object that we will talk to. The subsequent lines print out the different serial ports on the computer. You must select the serial port that your computer is actually connected. The code will print out the list of available serial ports in the terminal below your sketch. Figure 2 shows the list of ports available on my MAC. Yours will look different.

```
Port 0:   /dev/cu.Bluetooth-Incoming-Port
Port 1:   /dev/tty.Bluetooth-Incoming-Port
```

Figure 2: List of available serial ports

This can be challenging as the output is not human-readable. In general, you want to select the "largest" port value. In this instance I would select Port 1. As you can see in the code below, I selected port 3 on an earlier install. You will change the value 3 to whatever value is good for your Arduino.

```
//setup your arduino as port enumerated as 3
arduino = new Arduino(this, Arduino.list()[3], 57600);
```

Figure 3: Selecting port 3 for my Arduino - yours will be different

Part 2: Blinking an LED

To use a pin on the Arduino that pin must be declared as either an INPUT or an OUTPUT. Input pins bring information into the Arduino. Output pins produce some information. Most of your pins will be output pins.

In this sketch will blink the onboard LED, much like you did in Scratch. To begin, we need to set pin 13, which is connected to the LED, as an output. The line below in setup() performs this operation.

```
//declare Pin 13 as an Output
arduino.pinMode(13, Arduino.OUTPUT);
```

Once we have set pin 13 to be an output, it can be used in draw() to blink the LED. Examine the code below and note the similarities with your Scratch code. Instead of blocks, we have individual lines of code.

```
//turn on Pin 13
arduino.digitalWrite(13, Arduino.HIGH);

//wait 1000ms (1 second)
delay(1000);

//turn off Pin 13
arduino.digitalWrite(13, Arduino.LOW);

//wait 1000ms (1 second)
delay(1000);
```

Figure 4: Blinking an LED in Processing and in Scratch

Part 3: Reading in Data
Many of the parts in we have used require an Analog Input to read in data- the photocell, pressure sensor…etc. Accessing this information is very similar in Processing. Load up the sketch called “SimpleAnalogRead”.

In SimpleAnalogRead the setup loop should be the same as the others. If the serial port is incorrect, please change the port value as you did in the beginning of the lab.

Examine the DRAW() loop shown below. This code reads in a value from Analog Input 0 and stores that value in a variable called *value*. Value has a *type* called *int* which simply means that it stores positive or negative integers.

```
//read in the sensor value from Analog Input 0
int value = arduino.analogRead(0);
```

Figure 5: Reading in a value from Analog Input 0

An equivalent Scratch version of this code would look Figure 6. Instead of using a single block to read in the sensor and assign it, we are using a single statement in Processing.

Figure 6: Scratch equivalent of reading in and storing a value

Once the value has been read in, the Arduino can print out that value and display it in the console. The code also includes a small delay to ensure we do not receive a wall of text in the console. The instruction *println* stands for “Print Line” and will print out different variables in the console.

```
//read in the sensor value from Analog Input 0
int value = arduino.analogRead(0);

//print out the sensor value to the Processing console
println(value);

//delay a short time so we don't get a wall of text
delay(100);
```

Figure 7: Full loop requesting data from Analog Input 0 and printing that value to the console

When running, you should see output similar to the values below that show data coming through the Arduino and into the console.

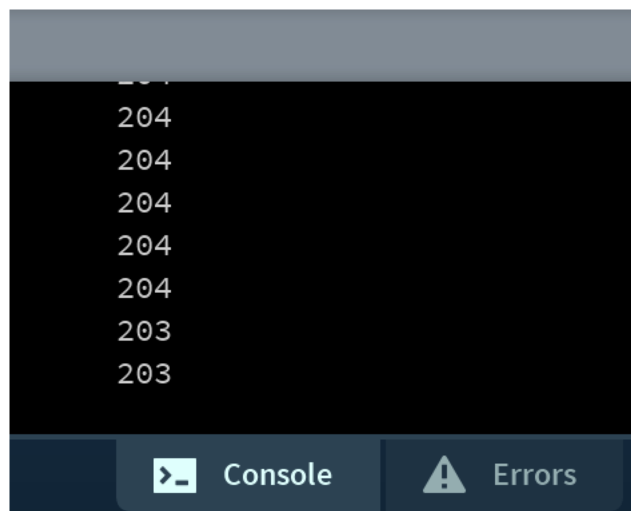


Figure 8: Printing data to the console

Part 4: Controlling a Servo

In Scratch we used special motor blocks to control the servo. In Processing we will write to the servo directly to control its operation. In general, sending a value of **0** to the servo will cause it to move counter clockwise and sending a value of **180** will cause it to move clockwise. A value in the middle will cause it to stop. Currently, I used **92** to make the servo stop.

Load up the sketch called "Servo" and examine the DRAW() loop. The SETUP loop should be the same as the previous sketches.

Just above the DRAW loop I used a variable to hold the pin that the servo is connected to. We can technically use a digital pin on the Arduino, however I have restricted this to pin 3 which also worked in Scratch.

```
int servoPin=3;
void draw()
```

Figure 9: Variable to hold the pin number of the servo

Having setup the pin, we can now cause the motor to rotate different directions and then stop. Notice the different values that are used to cause the motor to spin different directions. Try out different value to get different results.

```
//set motor 0 to spin counter-clockwise
arduino.servoWrite(servoPin, 0);

//wait 1 second
delay(1000);

//set motor to 180 to spin clockwise
arduino.servoWrite(servoPin, 180);

//wait 1 second
delay(1000);

//set the servo to 92 to stop. This value should be 90 but our motors
//are slight different. If 92 doesn't stop the motor, try a value nearby
arduino.servoWrite(servoPin, 92);

//wait 1 second
delay(1000);
```

Figure 10: Causing the motor to spin different directions using the servoWrite instruction

- Try change the servo values between 0 and 180 to see different behaviors

Part 5: Connecting Processing and Arduino – Range Finder

Now that you've had an introduction to working with an Arduino in Processing, it is now time to connect the two. Load up the sketch called RangeFinder and wire the Range Finder to your Arduino.

The setup() loop should be the same as before but the draw() loop is different. In this loop we begin to mix Arduino and Processing instructions. Like before, we read in a value from the sensor, but then use that value to make a simple animation.

```
//read in the sensor value from Analog Input 0
int value = arduino.analogRead(0);

//use a magic number to calculate how far away the object is
float inches = value * 0.4982;

//draw a circle on the screen based upon that distance.
fill(204,102,0);
ellipse(width/2, height/2, 2*inches, 2*inches);
```

Figure 11: Use the sensor value to draw an ellipse on the screen

Figure 11 shows code that reads the range finder result into a variable called "value" and then coverts that value into another variable called "inches". The new variable is the actual distance measured by the range finder. Using that distance, an ellipse is drawn on the screen and the distance is printed below it (code not shown in Figure 11).

- Try out different fill colors for your sketch
- Aim your range finder at different objects and see how "noisy" it is

Part 5: Connecting Processing and Arduino – RGB LED

Using your breadboard and the RGB LED, wire up your LED to the following pins:

- Red to Arduino Pin 3
- Green to Arduino Pin 5
- Blue to Arduino Pin 6
- wire up the long LED pin to Ground.

Once the wiring is complete, load up the sketch called "RGB_LED". This sketch uses many output pins and so there are many declarations of the outputs in the SETUP loop. As a short cut, I have made variables that contain the pin number for each

```
int redPin = 3;
int greenPin = 5;
int bluePin = 6;
```

```
//make the red, green, and blue pins outputs
arduino.pinMode(redPin, Arduino.OUTPUT);
arduino.pinMode(greenPin, Arduino.OUTPUT);
arduino.pinMode(bluePin, Arduino.OUTPUT);
```

Examine the code in the DRAW function shown below. This code sets a value for each LED color. Just like in Scratch, we use the Analog Write function to set the intensity of the LED. Go to www.colorpicker.cm and try manually creating different values and colors.

```
arduino.analogWrite(redPin, 219);

arduino.analogWrite(bluePin, 86);

arduino.analogWrite(greenPin, 20);
```

Figure 12: Manually setting the color of each LED (red, green, blue)

- Try out different color combinations to explore the LED

Part 6: Connecting Processing and Arduino – LED and the Mouse

In Part 5 we found that it is possible to manually control the LED color. While this is interesting, it is quite boring to select a new color each time. Load up the sketch called "ColorWheel".

In this sketch, Processing will display a color wheel that provides all variations of RGB colors. Keep your LED installed on your Arduino and watch what happens when you click a particular color. You should notice that your LED becomes the colors that you clicked on the screen.

Processing captures your mouse click and then determines what color you selected. Examine the mouseMoved function to see how this works. Processing first captures your mouse click by finding the mouseX and mouseY values and grabs the color of that pixel. Then, it breaks apart the pixel into its different color values.

```
void mouseClicked()
{
 //get the color of the pixel that you selected
  color c = img.get(mouseX, mouseY);

  //extract the three different colors
  float r = red(c);
  float g = green(c);
  float b = blue(c);
```

Figure 13: Pulling the color from an image

After each color has been extracted, those individual colors are sent to the LED. The blue and green values are decreased slightly as they are very bright and overwhelm the red color.

```
//write these three colors to the Arduino
arduino.analogWrite(redPin, (int)r);
arduino.analogWrite(bluePin, (int)b-20);
arduino.analogWrite(greenPin, (int)g-20);
```

Figure 14: Sending the colors to the LED

- Change the function from mouseClicked() to mouseMoved() and see how your sketch behavior changes

Part 5: Connecting Processing and Arduino – Inputs and Outputs
The final part of this lab is to connect an input device (the photocell) and an output device (the onboard LED) together. To do this, wire up the photocell as shown in the previous lab documentation. Load up the sketch called "PhotoCellandLED".

The first two lines of the sketch should be familiar as they are directly from the Analog Input example. These two lines read in and print the sensor value.

```
//read in the value of the sensor attached to
//analog input 0
int value = arduino.analogRead(0);

println(value);
```

However, the following lines show something different: an IF-ELSE statement.

```
//if the value is greater than 300, turn on the LED
if(value < 300)
{
    //it's dark! Turn on the LED!
   arduino.digitalWrite(13,Arduino.HIGH);
}

//otherwise, turn off the LED
else
{
  //don't need a light here!
  arduino.digitalWrite(13, Arduino.LOW);
}
```

Figure 15: IF-ELSE statements in Processing

An IF-ELSE statement is simply a way to make your program do one thing or another, but not both. In this code, the program tests whether the sensor data stored in "value" is less than 300. If so, the LED is turned on. Otherwise the LED is turned off.

- Change the "<" sign to a ">" sign and see the difference in behavior
- Change the "<" sign to "==" (two equals). What happens to your code when == is used? Note: the == operator tests for equality.