Arduino Workshop: Lab Materials

Dario Schor, Troy Denton

Contents

Exercise 1 - Getting started with Arduino

For this workshop, it is recommended that you use version 1.0.5 of the Arduino IDE.

Introduction to Arduino hardware

The Arduino is an open-spruce platform and a development environment for writing software. Although there are many other microntrollers that offer similar functionality, the Arduino is popular because it abstracts the functionality into an easy-to-use package. There are many types of Arduinos that offer different features useful for your project.

This workshop uses the Arduino UNO r3. This is the third revision of the original platform using an ATmega328 processor operating at 5 V. The processor operates at 16 MHz and has 32 KB or flash memory, 2 KB of SRAM memory, and a 1 KB EEPROM memory. The board provides access to 14 digital input/output pins and 6 analog pins that are sufficient for simple projects. The parts of the Arduino are shown in Fig. 1

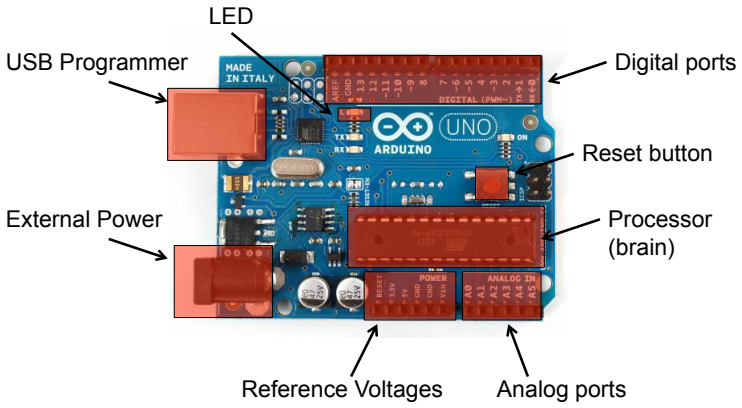

Figure 1: Parts of an Arduino UNO

Digital ports - Ports for controlling input/outputs. Some of these pins can be configured for pulse-width modulation. I/O pins can drive up to 40 mA.

Reset button - Resets the processor (may be located next to the USB Programmer in some boards)

Processor - ATmega328 microprocessor.

Analog ports - Analog ports for reading/writing continuous values.

Reference voltages - Reference voltage used for interfacing devices.

External power - Connection to plug in a 7-12V power supply.

USB programmer - Input to program the processor from a computer or provide power from a USB adapter. Can also be used to communicate with a computer through a serial interface.

LED - User programmable LED controlled through I/O pin 13.

Installing the Arduino IDE

Windows

There is an installer and a zip file option for Windows. The zip file is recommended as it allows you to run everything from the location you specify. Simply download and unzip the file from `http://arduino.cc/en/Main/Software`.

MacOSX

Download the zip file from `http://arduino.cc/en/Main/Software` and copy the Arduino software to your Applications directory.

Linux

For installation on Linux, head over to http://arduino.cc. Under the 'Download' section, there is a version for Linux (32 or 64 bit) compressed in a .tgz file. See the instructions below for basic installation

Listing 1: Installation on Linux-based systems

```
mkdir $HOME/Software
mv $HOME/Downloads/arduino-1.0.5-linux64.tgz $HOME/Software
cd $HOME/Software
tar xvzf arduino-1.0.5-linux64.tgz
./arduino
```

Using the Arduino IDE

The Arduino IDE is shown in Fig 2. The programs in Arduino are sometimes called "sketches" and are shown in the middle region with the white background. The black background region at the bottom is for compiler messages.

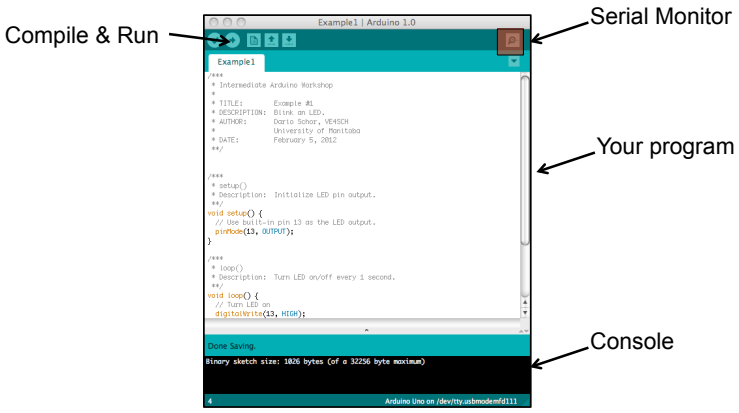

Figure 2: Arduino IDE

The key buttons in the interface are:

Checkmark - Check the code for syntax errors.

Right arrow - Download program to processor.

Magnifying glass - Open a serial monitor to send/receive data from the processor.

Other important menus to note are:

File→Examples - List of examples provided with the IDE. Great way to learn about Arduinos.

Tools→Board - Allows you to select what processor you are using. Most people here are working with the Arduino UNO, however the same IDE (and in some cases the same software) can be used for other processors.

Tools→Serial Port - Allows you to select the serial port (USB port) you are using to program your Arduino.

Exercise 2 - Blinky

Using on-board LED

Download and run "Example2.ino". You will see the on-board LED blink at 1 Hz.

Modify the program to blink the message "COOKIES" in morse code. The LED should be ON for 500 ms for a dash, and 200 ms for a dot. The spacing between symbols should be 200 ms and 600 ms between words. Fill in the blank for the sendLetter(), dot(), and dash() functions of the lab.

Adding an LED to the Arduino

Once your code is working, add an LED connected between PIN 13 and GND. Nothing should change, but now your program will control the external LED.

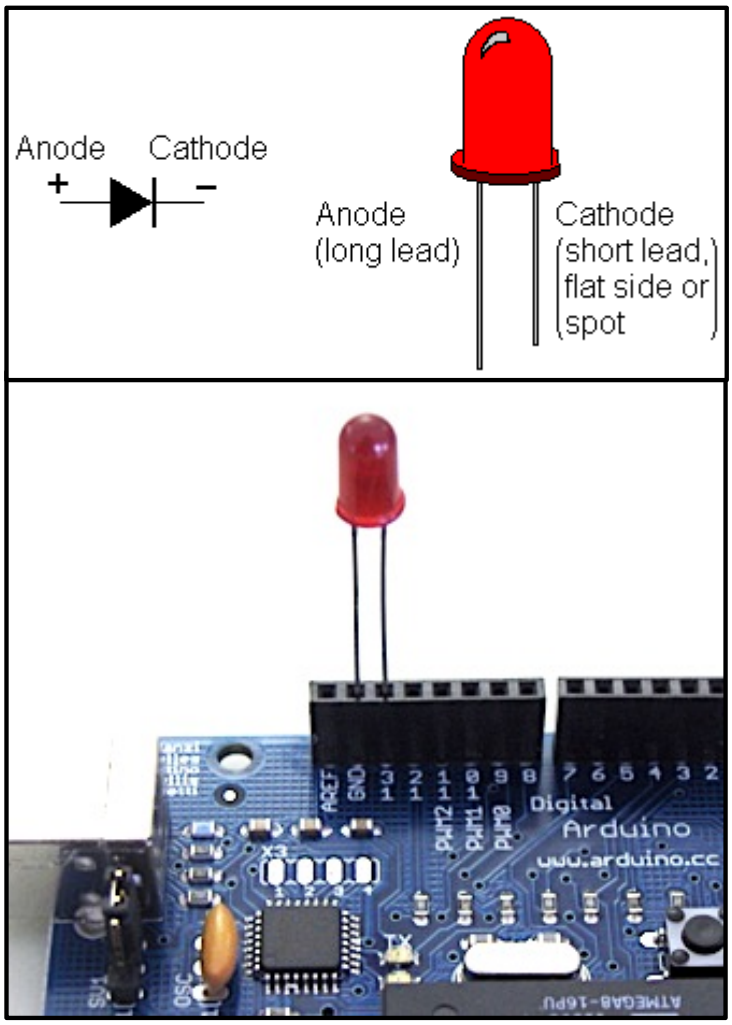

Figure 3: Connecting an LED to pin 13

Exercise 3 - LCD Shield

The Liquid Crystal Display (LCD) used in this lab has two rows of 16 characters. To control it, we can use the LCD library from Arduino found in `http://arduino.cc/en/Reference/LiquidCrystal`. This is a generic library that allows you to connect multiple types of LCDs, send text, scroll text, and more.

Writing to the display

Example 3a shows how to configure this LCD shield and write text to the screen. This will print "IEEE Workshop" on the first line and then blink the messages, "Troy!" and "I want cookies!" on the second line.

The key commands from the library are:

lcd.clear() - Clear the LCD.

lcd.begin(COLS,ROWS) - Initialize the LCD and specify the number of rows and columns for the particular model being used.

lcd.setCursor(COL,ROW) - Set the cursor to the position specified by the arguments COL and ROW.

lcd.print(MSG) - Print the given message to the LCD starting at the position of the cursor. If the message is too long it truncates it and ignores the rest.

Your task is to use the LCD library to write "IEEE Workshop" and "I want cookies!" to the two lines on the LCD. Then, in the loop, scroll the text all the way left (until you can't see anything), all the way right (until it is outside the screen), and loop again.

Button interactions

In this Arduino shield, all the buttons are read through a single analog pin. Each button has a different resistance associated with it and therefore produces a different value when pressed. This is not ideal for some applications where users can press multiple buttons simultaneously, but it is convenient for our example because it does not use many analog input pins.

Example 3b shows how to read the buttons and display a message depending on which button was pressed.

Example 3c shows how to create custom characters to display on the LCD.

Your task is to combine examples 3b and 3c. Place pacman at position (0, 0) to start the program. Use the buttons left, right, top, and bottom to move pacman across the screen. Make sure that pacman does not leave the screen dimensions.

Serial pass through from IDE to LCD

Example 3d shows how you can pass data from the IDE to the LCD. Run the example, open the serial monitor, and type some data into the input field. The data will be displayed on the LCD on the top row. This is a useful way to send commands to the Arduino from the console.

Your task is to repeat the pacman lab but using the IDE to enter 'u'-UP, 'd'-DOWN, 'r'-RIGHT, and 'l'-LEFT.

Exercise 4 - Optical Sensor

Reading with analogRead()

Arduino abstracts analog reads into functions that are independent on the type of converter. When a value is read using analogRead(PIN), it returns a value between 0 and 1023 which maps to 0 V to 5 V respectively.

Example 4 uses an optical sensor. The sensor has an LED on the side marked with the letter "S". The anode (+) is the top pin and can be connected to pin 13 on the board. The cathode can be connected to ground. This allows you to turn on/off the sensor using an I/O pin. The other side acts like a phototransistor. The top pin is the analog value and the bottom is ground. When you connect this circuit and open the serial monitor, you will see the values change when blocking the sensor (i.e., putting a piece of paper to block the light).

Your tasks is to write a program that turns on an external LED when the sensor is blocked.

Reading with digitalRead()

Alternatively, you can read the sensor as a logic level input. For many applications, this will suffice - we only need to know if something is either blocking, or not blocking sensor. Try using the *digitalRead()* command with the optical sensor - refer to the code in Listing 2 for an example of setting this up!

Listing 2: Reading an Optical Sensor with digitalRead

```
const int PIN_SENSOR=2;  //Replace with your sensor pin as necessary

pinMode(PIN_SENSOR,INPUT);

int state = digitalRead(PIN_SENSOR);
```

Timing analysis

The difference between an *analogRead()* and *digitalRead()* may seem trivial, but to illustrate the difference in hardware requirements, try playing with the code in 3

Listing 3: Timing analysis base code

```


unsigned long t1, t2, tdelta;

const int PIN_SENSOR=2;

void setup()
{

 Serial.begin(9600);

}

void loop()
```

```
{

  int i;
  pinMode(PIN_SENSOR,INPUT);
  t1 = millis();    //millis returns the number of milliseconds since the arduino turned on
  for(i=0;i<1000;i++)
    int dummy = digitalRead(PIN_SENSOR);
  t2 = millis();
  tdelta = t2 - t1;
  Serial.print("Time for 1000 digitalRead's is approximately: ");
  Serial.print(tdelta);
  Serial.println(" mS!");


  t1 = millis();    //millis returns the number of milliseconds since the arduino turned on

  for(i=0;i<1000;i++)
    int dummy = analogRead(PIN_SENSOR);
  t2 = millis();
  tdelta = t2 - t1;
  Serial.print("Time for 1000 analogRead's is approximately: ");
  Serial.print(tdelta);
  Serial.println(" mS!");


}
```

Exercise 5 - Regular Servo Motor

Description of Servo Motors

Servo Motors are a unique type of motor that accept a positional command. The Servo Motor (heretofore *Servo*) uses internal circuitry to rotate its shaft to the position specified - it will maintain that position up to a specified amount of torque.

This precision adjustment of Servo Motors makes it useful in many hobbyist and industrial applications.

Connecting to a Servo Motor

The Servo only requires three connections - *Power*, *Ground*, and *Pulse*. Connect the regular servo motor's pulse pin to *D3* on the arduino.

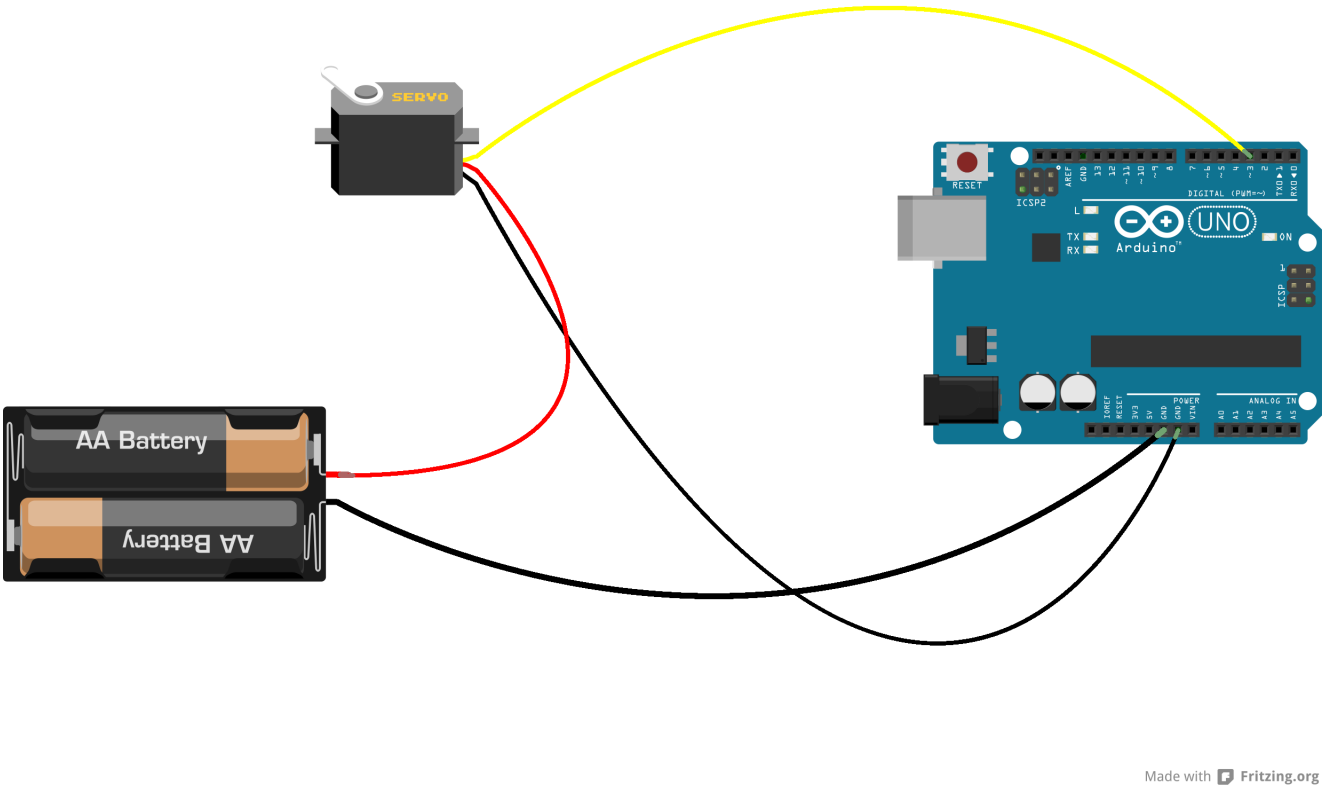

Figure 4: Connection for Regular Servo motor. Battery represents 5VDC bench supply!

When you wish to connect the Servo motor through the LCD shield, refer to Figure 5 for the connection mapping.

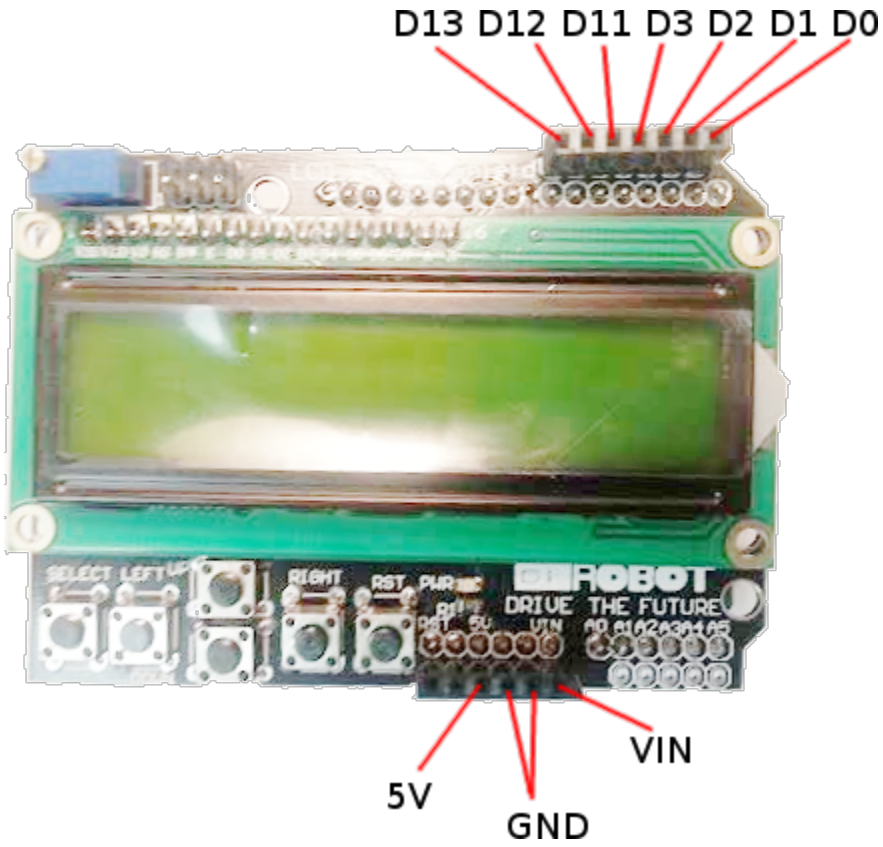

Figure 5: Arduino connections mapped through LCD shield

Sending commands to the Servo

Servo commands are sent as precise digital pulses.

From Wikipedia (`http://en.wikipedia.org/wiki/Servo_control`):

"*The servo expects to see a pulse every 20 ms, however this can vary within a wide range that differs from servo to servo. The length of the pulse will determine how far the motor turns. For example, a 1.5 ms pulse will make the motor turn to the 90 degree position (neutral position).*"

For our purposes today, we do not need to know exact pulse widths - the Arduino has a Servo library that takes care of that for us.

```
Servo myServo; //declare a Servo object
myServo.attach(servoPin);     //servoPin is the pin used to send pulses
myServo.write(90);  //move the servo to the 90 degree (neutral) position
```

Control position with left/right buttons

In this section, use the 'Left' and 'Right' keys on the LCD interface to increase/decrease the rotational position of the servo motor. See *lab5.ino* for a starting point!

Control step size with up/down buttons

In addition to the controls for the 'Left' and 'Right' keys, use the 'Up' and 'Down' buttons on the LCD interface to increase the step size.

Exercise 6 - Continuous Servo Motor

In the hobbyist RC market, the servo protocol is widely used - some remote control receivers output standard servo pulses for all channels, for example. In some instances, hobbyists want to control a standard motor from these servo commands. As you have likely found, regular servos have a limited rotation range. Hobbyists began modifying 'standard' Servos so that they could do continuous rotation. This got popular enough that 'continuous rotation' servos became commercially available. These typically do not have any speed control - only forwards, backwards, or stop. In our lab apparatus, we use a continuous rotation servo to control the conveyor belt. For this example, connect the continuous servo's pulse pin to *D11* on the arduino. Refer to Figure 6 for a connection diagram.

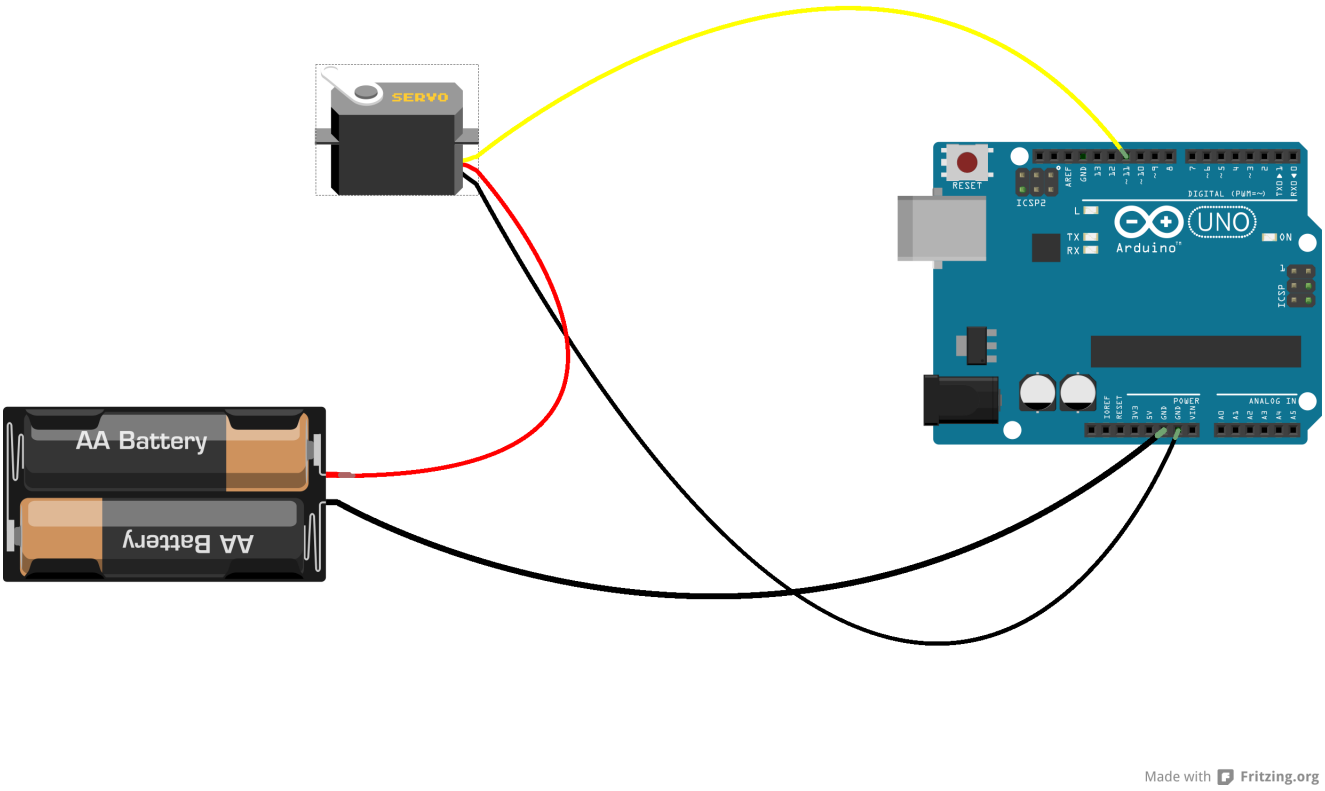

Figure 6: Connection for Continuous Rotation Servo motor. Battery represents 5VDC bench supply. Note: refer to Figure 5 to connect through the LCD shield

Control via Serial Port

For part 1, control your continuous rotation servo via serial command. This is a common technique to control arduino-based hardware from a PC - it is easy to write programs that communicate via USB serial port. Refer to Listing 4 for an example on how to read serial output from a PC, and check out *lab6.ino* for a starting point for the following exercises!

Listing 4: serialEvent example for obtaining PC output

```
// serialEvent() is called after every iteration of 'loop', when there is serial data
// available
void serialEvent()
{
  while(Serial.available())  // Serial.available() tells us if there are characters
                             // to process in the receive buffer
  {
    char inChar = Serial.read();  // Serial.read() returns the first character in
                                  // the serial receive buffer

    //  Act on serial data here
  }

}
```

Control via LCD interface

Move the continuous rotation servo via LCD interface. The controls should work as so:

- holding Left or Right should make the servo rotate clockwise or counterclockwise, respectively
- the motor should not move when no keys are pressed

See *lab6.ino* for a starting point!

Control via optical sensor

Move the continuous rotation servo until the optical sensor becomes obstructed. Hint - re-use your code from exercise 4! See *lab6.ino* for a starting point!

Exercise 7 - System Integration - Manual Control Mode

In this lab, we will begin interfacing to the lab apparatus.

The user should be able to:

- Use the left/right buttons to move the conveyor belt
- Use the up/down buttons to move the lifter assembly
- Use the "Select" button to push the extruder

This will allow you to determine timing constants for the various operations required for the full integration in exercise 8. This portion of the lab exercise will require you to come up and test on the lone lab apparatus - please be courteous and allow for other participants to test their designs as well!

The following connections will be made already, you will simply bring up your programmed arduino. These are already established in the starting code in *lab7.ino*:

- Regular Servo on pin D3 - this controls the lifter assembly
- Continuous Rotation servo on pin D11 - this controls the conveyor belt
- LCD connections remain unchanged from previous examples
- A motor controller has been added to control the cookie dough extrusion motor. This has been abstracted for you in the example code - simply call *pushExtruder()*, *pullExtruder()*, and *stopExtruder()* as necessary!

Exercise 8 - System Integration - Automatic Mode

In automatic mode, the user should be able to push the "Select" button and print *at least two* cookies on the tray. A state machine approach is recommended, and provided partially in the example code, *lab8.ino*.

The arduino should check the optical sensor on pin 2 - If there is no tray underneath the extruder, do not try to push out any dough! This is accomplished as shown in Listing 5.

Listing 5: Reading the optical sensor

```
//input pin for the sensor
const int PIN_TRAY_SENSOR=2;

//set pin as input
pinMode(PIN_TRAY_SENSOR,INPUT);

//to check the sensor...
if (digitalRead(PIN_TRAY_SENSOR)==LOW)
{
  Serial.println("Tray is covering the sensor");
}
```

Super bonus awesome points: Let the user select the number of cookies to print via the LCD interface... but keep a safety check to ensure the extruder only actuates when there is a tray underneath it!