

MTRE 2610 Engineering Algorithms and Visualization – Dr. Kevin McFall

Laboratory – Arduino Basics

Introduction

The goal of this laboratory exercise is to understand the basic operation of the Arduino Uno and use it to read input from switches and potentiometers to control LEDs and DC motors.

Arduino Uno

The Arduino Uno, pictured in Figure 1, is a microcontroller allowing for easy reading and writing of voltage values. This offers the ability to quickly develop applications incorporating sensors offering information about the surroundings and manipulation of the environment through the control of actuators such as motors. The Arduino is programmed using a special integrated development environment (IDE) on a computer by connecting to the Arduino with a USB cable. When getting started, be sure Arduino Uno is selected under the IDE menu Tools – Board, and that the correct COM port is chosen under Tools – Port after the Arduino is connected to the computer by USB. Analog sensors, such as the potentiometer used in this exercise, are connected to the Arduino on one of the analog pins, and the controlling signal for the motor is sent via a digital pin.

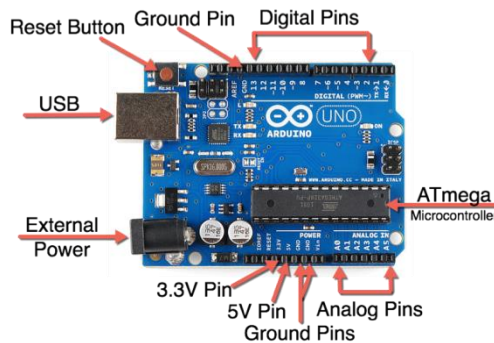



Figure 1: Arduino Uno microcontroller with its various components labelled.

All Arduino programs must contain two functions, `setup` which runs once when the Arduino is powered up, and `loop` which repeats continuously while the Arduino is powered. The syntax for these functions is as follows:

```
void setup() {  
    // put your setup code here, to run once  
}  
void loop() {  
    // put your main code here, to run repeatedly  
}
```

The Arduino programming language is a superset of C++, such as enclosing a function's commands in `{ }` curly braces, ending commands with a `;` semi-colon, and writing single-line comments with the `//` double slash. To upload a program from the IDE to the Arduino, click on the upload icon  in the IDE.

Digital input and output

A digital signal is one that can only take one of two values, often designated as true and false. Digital signals are represented by different voltages, which are 5 and 0 V on an Arduino. In reality, signals are never truly digital since the actual voltage in a signal is never exactly 5 V; rather, voltages close to 5 V are considered true (or high) and values close to 0 V are considered false (or low).

Every Arduino board contains numerous digital pins, which can be used as either inputs or outputs. In order to work correctly, each pin must be declared as either an input or an output using the `pinMode` command, usually placed in the `setup` loop. Digital inputs will read an external digital signal such as that coming from a switch with the `digitalRead` command. On the other hand, a digital pin configured to be an output will enforce a value of either 5 or 0 V on the pin using `digitalWrite`, which can be used to drive electronic components such as LEDs and transistors.

Analog input and output

Analog signals can vary continuously in voltage rather than taking two discrete values like digital signals. For example, a light sensor can generate a voltage which is directly proportional to the amount of light incident upon it. Arduino boards read analog voltages between 0 and 5 V with `analogRead` when connected to an analog input pin.

Unfortunately, an Arduino cannot produce a variable analog output voltage. However, using pulse width modulation (PWM), an analog device can be “fooled” into interpreting a digital signal as an analog one. The digital signal will always be either 0 or 5 V, but the *time* it is high can be varied. If a signal oscillates equally between 0 and 5 V at a sufficiently fast rate, a motor or other device will not notice the fluctuations but rather interpret the signal as 2.5 V. The value of the “analog” voltage is controlled by changing the duty cycle, which is the ratio of the pulse width and signal period, appearing in Figure 2. A duty cycle of 100% therefore generates a pure 5 V DC signal.

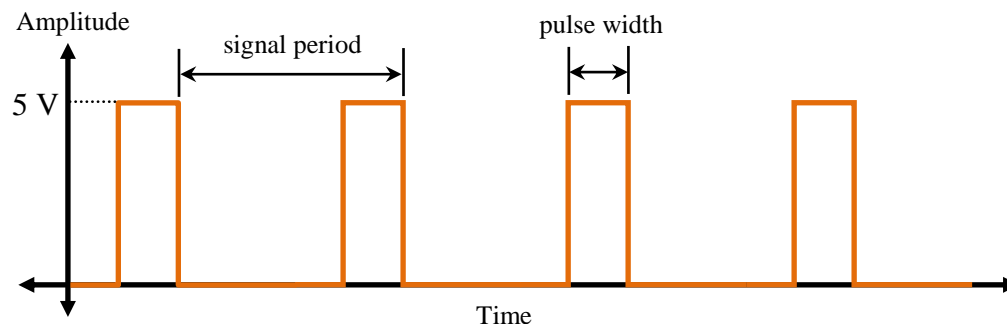


Figure 2: Depiction of pulse width modulation signal.

Arduino boards do not have analog output pins since they cannot produce a true analog output. Instead a PWM signal is generated on a digital output pin. Pins capable of producing a PWM signal are marked with a ~ on the board. A PWM signal can be generated in two ways, manually during a digital pin on and off (so-called “bit-banging”) and using the Arduino command `analogWrite`. An analog signal can be simulated manually by writing a digital pin high, using the `delay` command to wait the desired pulse width, writing the pin low, and then delaying the remaining time in the signal period. This method benefits from simple adjustment of the signal period but suffers from pausing operation of the `loop` function for the duration of the signal period. Using `analogWrite` is simpler than the manual method, but is difficult to modify the signal period from the default 490 or 980 Hz frequency (depending on the pin used).

Breadboards

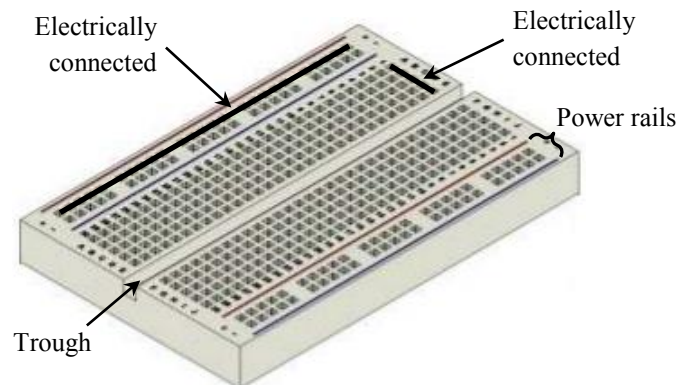


Figure 3: Breadboard illustration including which holes are electrically short-circuited.

Although breadboards vary in size and shape, the principle of operation is the same; they are simply array of holes housing tiny electrical contacts. The two primary sections of a breadboard are red and blue marked rows of contacts called power rails, and columns of 5 contacts separated by a trough as appears in Figure 3. The contacts on a breadboard are designed to accept the pins of integrated circuits (ICs), other electronic parts, and small-diameter jumper wires. ICs, switches, and other devices manufactured as dual inline packages (DIPs) are placed on the breadboard straddling the center trough. Some larger boards have several troughs, and any one may be used. The pins on alternate sides of a DIP device insert into holes on either side of the trough. Note that holes on one side are electrically insulated from holes on the other. Also, each of the five holes in any column are tied together internally but insulated from all other columns. Interconnections between various columns, and the power rails, are made using short pieces of jumper wire. Since most devices require connections to power and ground, power rails spanning the entire length of the breadboard are convenient. Note that on some larger breadboards, the power rail rows may be divided into two or more separate buses.

Switches (digital inputs)

A typical analog circuit diagram appears in the left side of Figure 4, which includes a 5 V battery, resistor, and ground. Since digital circuits are not as concerned with circuit loops, they often appear in the simplified form on the right of Figure 4. The analog method emphasizes current traveling around loops whereas the digital model focuses on current following paths from power to ground.

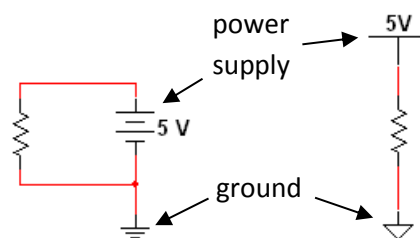


Figure 4: Equivalent circuits using analog (left) and digital (right) symbols.

The two primary types of switches are toggles (like a typical light switch) and the pushbutton switches (like the home button on an iOS device). By moving either type of switch, an open or closed circuit is realized across it. Pushbuttons contain an internal spring returning them to a default position when released. Some pushbuttons are open by default, termed “normally open”, while others are “normally closed” where the spring maintains a closed circuit when not pressed. Toggle switches have no springs and require action to change the state from closed to open and vice versa. Since the Arduino can sense voltage but not resistance, switches must be connected to power for proper operation. As in Figure 5, the path between power and ground include a resistor and switch in series. When

the switch is open, current has no path to ground and so the resistor experiences no voltage drop. This means the voltage at Y will be 5 V and 0 V for the left and right sides of Figure 5, respectively. Alternately, with the switch closed voltage at Y is 0 V and 5 V for the left and right, respectively because Y is short-circuited to either ground or power. The resistor is termed pull-up when connected to power and pull-down when connected to ground. The different combinations of pull-up/down and normally open/closed allow designing the output Y to be high or low in the unpushed position for either type of switch. Toggle switches are connected in the same manner, with either pull-up or pull-down resistors, but obviously do not have a default position. The exact resistance value used does not affect the output Y , but larger values are preferred to limit current and thus power dissipation through the resistor.

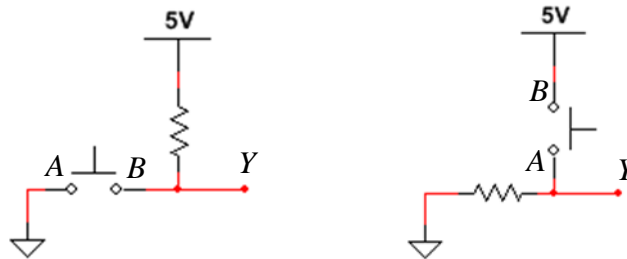


Figure 5: Circuit diagram for connecting a normally open pushbutton switch both with pull up (left) and pull down (right) resistors.

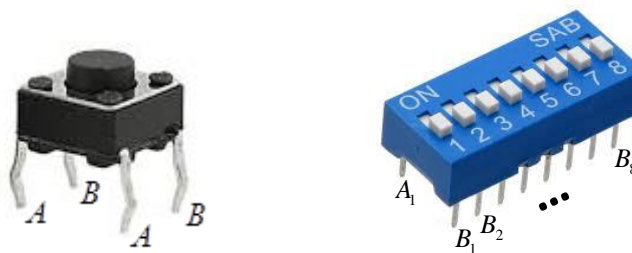


Figure 6: Image of a pushbutton (left) and toggle switch (right) indicating terminals A and B from Figure 5.

The pushbutton switch in Figure 6 (left) is marked with terminals A and B representing both sides of the switch from Figure 5. Be sure that A and B are not inserted into the same 5 hole column on the breadboard. Since Y and B are the same terminal in Figure 5, the switch output Y is connected to the Arduino input using a jumper wire connected to the same 5 hole column as B . Note that both B terminals (and both A terminals) are short-circuited to each other, offering multiple places to connect to the switch. The toggle switches in Figure 6 (right) are designed to straddle a breadboard trough, and provide 8 independent switches. The switch contacts are closed in the ON position. The switch output Y can be connected with a jumper wire to a digital input pin to be read by an Arduino.

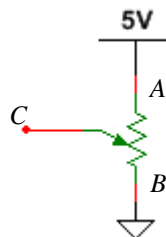


Figure 7: Circuit diagram for a potentiometer.



Figure 8: The potentiometer supplied in the lab kits.

Potentiometers (analog inputs)

The output of a potentiometer can be controlled by moving a dial that varies the resistance between the output C and power supply, thus varying the output voltage. Potentiometer circuits are drawn as in Figure 7 where the potentiometer is connected to power at A , ground at B , and the output is terminal C . These terminals are marked in

Figure 8 on the physical potentiometer pins. The Arduino power and ground are connected to pins *A* and *B*, respectively, and terminal *C* connects to an analog input on the Arduino.

Light emitting diodes (digital outputs)

Light emitting diodes (LEDs) offer a simple method for displaying digital values. The light emitted by LEDs depends on the current passing through them. Because they have very little internal resistance, connecting directly to power and ground will cause them to burn out due to large currents. Therefore, a resistor must be included in series with the LED as illustrated in Figure 9. The value of the resistor will affect the current, and therefore LED brightness; resistors in the range of several hundred Ohms offer the best performance for 5 V power supplies. LEDs have two different length leads. Current must flow into the long lead and out the short lead. This requires that the long lead be connected on the power side rather than the ground side.

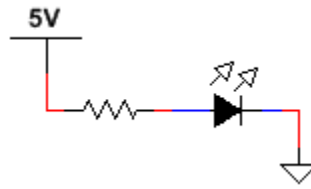


Figure 9: LED connected to power including a current-limiting resistor.

DC motors (analog outputs)

A DC motor is an analog device which turns at speeds dependent on the voltage applied across its two leads. Unfortunately, DC motors present two problems for an Arduino. The first is that Arduinos cannot output a true analog output, although a PWM signal with sufficiently high frequency can be used to control motor speed.

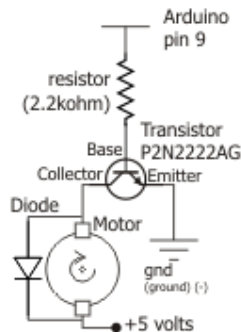


Figure 10: Circuit diagram for connecting the DC motor

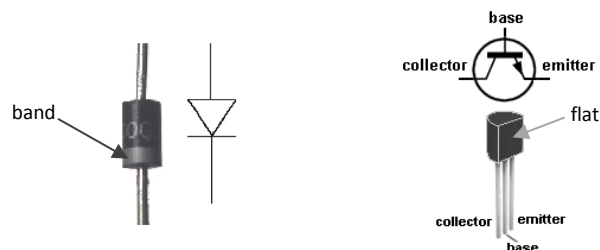


Figure 11: Orientation of diode silver band (left) and transistor flat side (right) relative to their circuit symbols.

The second problem driving DC motors is that they generally require more current than can be supplied by an Arduino digital output pin; digital pins can supply 40 mA while the 5 V pin can supply 200 mA¹. Currents around 200 mA are sufficient to drive small DC motors, but without the PWM capabilities of the digital output pins, speed could not be varied. To resolve this, the digital pin can be used to instead drive a transistor to pass current to the motor from the 5 V power pin as in Figure 10. A transistor essentially acts as a voltage controlled switch, allowing current to flow through the DC motor according to the PWM duty cycle applied to its base.

Notice in Figure 10 that a resistor and diode are also connected in addition to the transistor. The purpose of the transistor is to open or close a channel for current to flow from the collector to the emitter, but some current passes through the base as well. Without the base resistor, this current could become large since current is high through circuits with little or no resistance. Under normal operation, current will flow from the 5 V power source through the motor, into the collector, and out the emitter to ground. However, DC motors are inductive devices and current can flow through them in the opposite direction for short periods of time when power is turned on or off. If current attempts to flow backward through the motor, the diode offers a lower resistance path that will protect the motor.

Alternative DC motor circuitry

Driving high current devices from low current microcontroller output pins is relatively common, not only with motors but solenoids as well. The effort in wiring the electronic components in Figure 10 can be avoided by using a connecting wire with the electronics built-in as appears in Figure 12. The same Arduino sketch can be used whether connected as in Figure 10 or 12, the only difference is in the complexity of the necessary wiring.

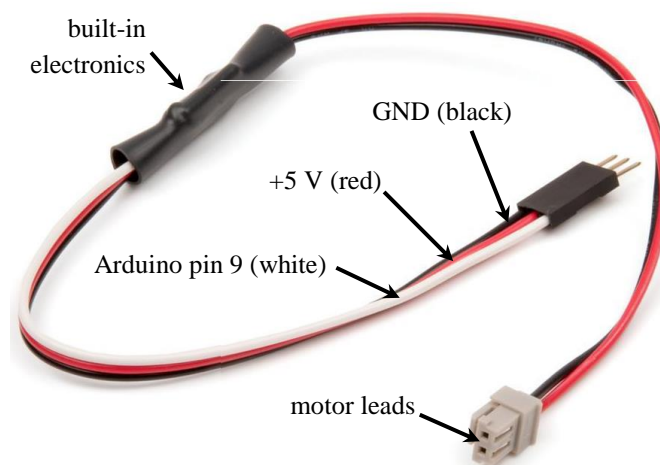


Figure 12: Motor connector including built-in transistor circuitry.

Laboratory exercise procedure

The Arduino website offers many [practical examples](https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations)² of how to get started with basic devices. First, follow the tutorial for [Blink](https://www.arduino.cc/en/Tutorial/Blink)³. Notice that the Arduino has a built in LED connected to pin 13 which blinks in unison with the externally connected LED. Leave the LED connected to the Arduino and follow the [Button](https://www.arduino.cc/en/Tutorial/Button)⁴ tutorial. Modify the sketch to make the LED blink when holding the switch down; the LED should remain off when the button is not pressed. Leave both switch and LED connected as they will be required later. Leave the button and LED subcircuits

¹ <http://playground.arduino.cc/Main/ArduinoPinCurrentLimitations>

² <https://www.arduino.cc/en/Tutorial/HomePage>

³ <https://www.arduino.cc/en/Tutorial/Blink>

⁴ <https://www.arduino.cc/en/Tutorial/Button>

in place, and complete the [DC motor tutorial](#)⁵ as an example of how an Arduino can drive DC motors. Refer to Figure 11 for diode and transistor orientation. Be sure to demonstrate motor operation using the circuitry in Figure 10 for full credit. OPTIONAL: afterwards, circuitry from Figure 10 can be replaced with the connecting wire in Figure 12. With the button, LED, and motor subcircuits in place, separately wire the potentiometer by following the [AnalogReadSerial](#)⁶ tutorial.

Now take the four separate subcircuits for LED, switch, potentiometer, and motor and connect them individually to the Arduino to construct a system that

1. Waits to begin until a pushbutton switch is pressed
2. The DC motor runs for 5 s at a speed controlled by a potentiometer
3. The LED is off when the motor is running and on when it is not
4. The process begins again after the motor stops
5. The motor speed is adjustable during the 5 s the motor is on

Grading rubric

1. 25 points: Successful operation of blink tutorial
2. 25 points: LED blinks while holding button down and is off when button is released
3. 25 points: Successful operation of the DC motor tutorial using Figure 10
4. 25 points: Operation of the final five step process

⁵ <http://www.oomlout.com/oom.php/products/ardx/circ-03>

⁶ <https://www.arduino.cc/en/Tutorial/AnalogReadSerial>