Lab 7

Goals

This lab introduces students to an inexpensive open-source cross-platform microcontroller called the Arduino. It is a digital, programmable device that can take inputs from sensors or switches and control a variety of output devices including lights and motors through digital outputs.

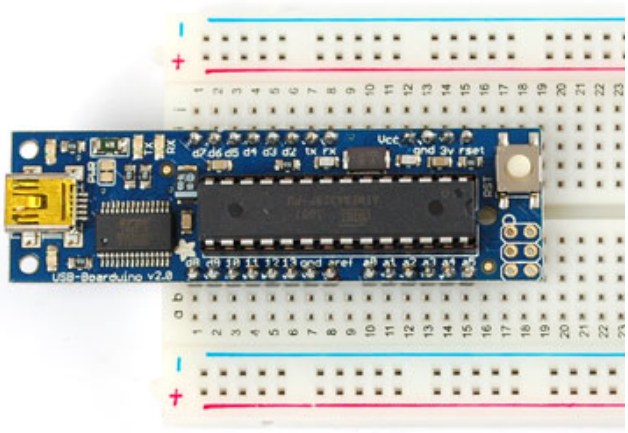
This lab will also introduce students to:

- Programming a microcontroller to execute a simple sequence of digital outputs (7-1: Stoplight Controller)
- Programming a microcontroller to respond to a digital input (7-2: Stoplight controller with a digital sensor)
- Using a switch with a "pulldown resistor" as a digital input (7-2)
- Using a transistor as a switch to control power to a motor (7-3: Controlling a motor with microcontroller)
- Controlling power delivered to a motor using "Pulse-Width Modulation" (PWM, 7-3)
- Reading an analog sensor value using a microcontroller (7-3)

Background: Arduino Microcontroller

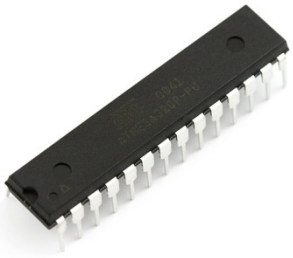
Microcontrollers are digital programmable devices that are designed for measurement and control and are often embedded in instruments. They should not be confused with microprocessors, which are much more complicated, fast and expensive programmable devices because they need to run a complicated operating system and support video displays. Microcontroller chips, unlike microprocessors, usually have built-in analog-to-digital (ADC), digital-to-analog converters (DAC) and PWM digital outputs so they are ready to use for measurement and control.

History. The goal of the Arduino initiative is to disseminate inexpensive, open-source microcontroller boards together with an easy-to-use software programming environment. It was first introduced in 2005 by founders Massimo Banzi and David Cuartielles from Ivrea, Italy.



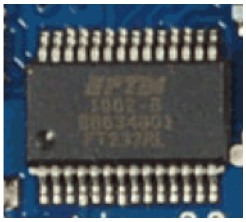
You will be using an Arduino variant called a "**Boarduino**" from Adafruit Industries that is designed to directly plug into a breadboard for prototyping and instructional applications. This powerful little device cost $25 as a kit.

Arduino / Boarduino Hardware



Microcontroller. The microcontroller chip is an Atmel ATmega328P, which is an 8-bit microcontroller with 32KB flash memory, 2KB SRAM, 1024B EEPROM, an 8-channel/10-bit A/D converter and achieves throughputs approaching 16 MIPS (Million Instructions Per Second) when run with a 16 MHz oscillator.

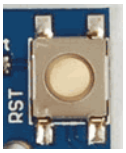
The chip is clocked with a 16 MHz ceramic **resonator**.



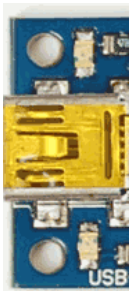
Serial Converter. Programs are downloaded from a computer to the Arduino using a standard USB cable. If desired, data can also be exchanged between the Arduino and a computer using USB. A FT232RL serial converter on the Arduino converts the USB signal to the serial standard used by the microcontroller (RS232). There are two LEDs labeled **RX** and **TX** that blink when data is received or transmitted.

Power. The USB input also conveniently delivers 5V power to the Boarduino. However, it is also possible to power Arduinos from a small external battery. Standard Arduinos can work off of any voltage between 7 – 17 V because they have an on-board 5V voltage regulator. A standard 9V battery works well. Therefore the connection to the computer is only needed to download software and then the Arduino can function independently of a computer, e.g., embedded in an instrument. (Note: Our Boarduino is not designed to use external battery power.)

Fuse. The USB Boarduino has a 500 mA fuse that will trip if too much current is drawn in order to protect the host computer.



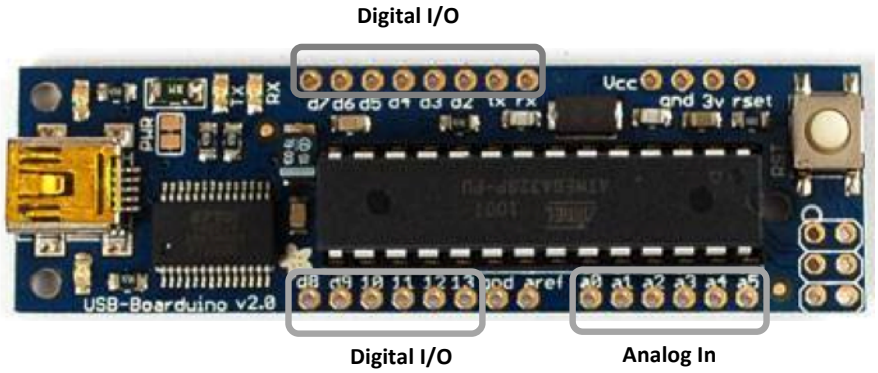
RESET button. Pressing this button resets the hardware and software; the board also has software auto-reset capability, so resetting is automatic when you download a new program



LEDs. There are two LEDs on either side of the USB port. The green one indicates power is on and the red one is connected to pin d**13** just like classic Arduinos. You can use the pin 13 LED for debugging. It will also blink during boot loading and right before the boot loader starts.

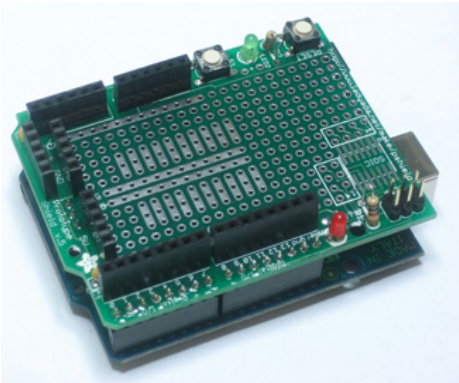
I/O (Inputs/Outputs).

- There are 14 digital pins that can be programmed to either read in digital data (e.g., read a switch) or output digital data (e.g., turn on a light). Two of the digital lines (d0 and d1) are dedicated to run the **TX** and **RX** LEDs described previously. Six of the digital pins (d3, d5, d6, d9, d10, d11) can put out Pulse-Width-Modulation (PWM) digital signals to allow variable power to be applied to a load (e.g., control the speed of a motor).



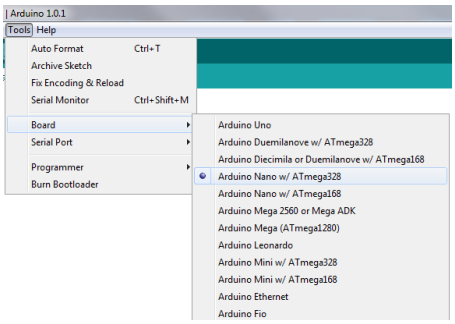
- Six 10-bit Analog-to-Digital Converter (ADC) inputs are available to read in analog signals (e.g., read a temperature sensor).

- The Boarduino allows easy access to these I/O pins because the board directly plugs into a breadboard. Standard Arduino boards are a bit larger and have "header" sockets on top of the board, rather than header pins below the board, to make I/O connections. The standard form factor of Arduinos makes them easily extendable with add-on boards that plug in to the standard Arduino board. These add-on boards are called "**shields**" and are widely available from third-party vendors. For example, a "Proto-shield" (right) is a prototype board that allows you to solder in any circuit of your choice, e.g., add a light and temperature sensor. Other available shields include a data-logging shield with a socket to plug in an SD card, a touchscreen display shield, a motor and server controller shield, a GPS shield and many others.



Arduino Software. The software environment for the Arduino is one of its best features because it is designed to be easy enough for use by non-engineers. Programs written for the Arduino are called **"sketches"**. Sketches are written on a computer (Windows or MacOS X or Linux) running the Arduino Integrated Development Environment (IDE) and then downloaded. Sketches are written in a form of C / C++ with a software library that makes most common I/O operations very simple.

Open the Sketch Pad and choose the correct board from the Tools Menu:



Two common functions are written by the user:

- **setup()** runs once at the start of the sketch to initialize settings, e.g., define whether a digital pin is an output or an input
- **loop()** is a function that will run indefinitely until the board is turned off.

For example, the following program turns the LED on the board (connected to d13) on and off each second.

```
#define LED_PIN 13

void setup () {
  pinMode (LED_PIN, OUTPUT); // enable pin 13 for digital output
}

void loop () {
  digitalWrite (LED_PIN, HIGH); // turn on the LED
  delay (1000); // wait one second (1000 milliseconds)
  digitalWrite (LED_PIN, LOW); // turn off the LED
  delay (1000); // wait one second
 }
```

As in the C language, every line to be executed ends with a semicolon (;) and every control structure begins and ends with brackets ({ }). Comments are inserted following two forward slashes (//). The *void* before a function name indicates that the function does not return a value.

The complete language reference with links that describe every command can be found at: http://arduino.cc/en/Reference/HomePage

Exercise 7-1: Arduino Stoplight Controller

Now that you have implemented a stop-light controller the "hard way" in Lab 6, you will see how modern stop-light controllers are implemented the "easy way" using a microcontroller.

Hardware Setup:

1. Mount the Boarduino onto the breadboard, straddling the pins on either side of one of the slots on a board.
2. Connect "gnd" to one of the buses on the breadboard. This is your ground.

N-S LED Connections

3. Connect anodes of LEDs Red, Yellow and Green to pins d5, d6, d7 respectively on the Boarduino.
4. Connect the cathode of each of the three LEDs to a current-limiting resistor (~270 Ω) and then connect the other end of the resistor to ground.

E-W LED Connections

5. Repeat the above steps using pins d11, d12, d13 for LEDs Red, Yellow and Green respectively.
6. Connect a USB cable from your computer to the miniB jack on the Boarduino.
7. Open the Arduino Software. Paste the program in the Sketch Window. Click Verify ✓and then Upload⇨.
8. Demonstrate to the Instructor or a TA that your stoplight controller is working correctly.

Software: Besides being much easier to wire up, use of the programmable Arduino gives you much more flexibility on the timing of each of the lights. For example, let's assume the N-S street has much more traffic than the E-W street, then the green light for N-S should be much longer than the green light for E-W. Try out the following sketch:

```
#define N_S_red 5              // assigning digital pins to each light
#define N_S_yellow 6
#define N_S_green 7

#define E_W_red 11
#define E_W_yellow 12
#define E_W_green 13

#define yellow 1000            // yellow light (both N-S and E-W) = 1s (1000ms)
#define NS_g_time 8000         // N-S green for 8s
#define EW_g_time 2000         // E-W green for 2s

void setup () {
```

```
  pinmode (N_S_red, OUTPUT);
  pinmode (N_S_yellow, OUTPUT);
  pinmode (N_S_green, OUTPUT);
  pinmode (E_W_red, OUTPUT);
  pinmode (E_W_yellow, OUTPUT);
  pinmode (E_W_green, OUTPUT);
}

void loop () {

  digitalWrite (N_S_red, LOW);          // State 1: N_S green, E_W red
  digitalWrite (N_S_yellow, LOW);
  digitalWrite (N_S_green, HIGH);

  digitalWrite (E_W_red, HIGH);
  digitalWrite (E_W_yellow, LOW);
  digitalWrite (E_W_green, LOW);
  delay (NS_g_time);

  digitalWrite (N_S_green, LOW);        // State 2: N_S yellow, E_W red
  digitalWrite (N_S_yellow, HIGH);
  delay (yellow);

  digitalWrite (N_S_yellow, LOW);       // State 3: N_S red, E_W green
  digitalWrite (N_S_red, HIGH);
  digitalWrite (E_W_red, LOW);
  digitalWrite (E_W_green, HIGH);
  delay (EW_g_time);

  digitalWrite (E_W_green, LOW);        // State 4: N_S red, E_W yellow
  digitalWrite (E_W_yellow, HIGH);
  delay (yellow);
}
```

Exercise 7-2: Stoplight Controller With Digital Sensor

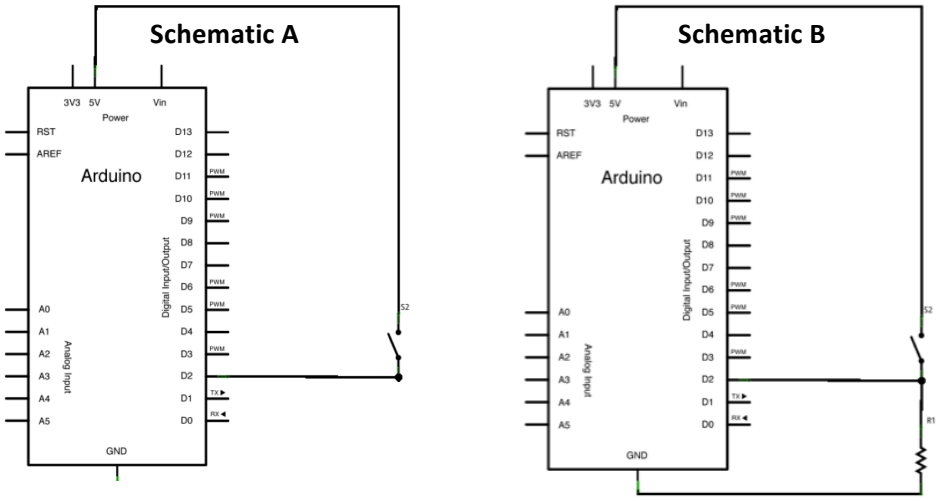
Sometimes the best approach when one of the streets has very little traffic is to have the light turn green only when a vehicle is actually waiting. Therefore a sensor (switch) is installed in front of the stoplight that responds to the weight of the vehicle. When the switch is closed there is a delay and then the lights cycle.

Theory: Reading the digital state of a switch with a "pull-down" resistor.

The Arduino digital pins can be configured to read digital values. If the voltage applied to the pin is greater than ~3V the input will be considered HIGH, otherwise it is LOW. As shown in **Schematic A**, if one end of the switch is connected to +5 V (labeled "Vcc" on the Boarduino)

and the other end to the digital pin (d2), then when the switch is closed the digital input d2 will be read as HIGH.

However, consider the case when the switch is open. Will the digital input be LOW? Actually, the input will be "floating" with an undefined voltage relative to ground. Should we then connect the digital input to ground? This won't work either because the +5V power supply will be directly connected to ground (short circuited) when the switch is closed.



The solution is to use a "pull-down" resistor as illustrated in **Schematic B**. Now when the switch is closed the digital input is directly connected to +5V and will read a HIGH value, yet the resistor will limit the current between +5V and ground. When the switch is open the digital input is connected to ground through the pull-down resistor. Since the current at the digital input is ~0, there is no voltage drop across the resistor, so the voltage at the digital input = 0 V and is read as LOW.

Experiment

Hardware: Connect the switch and pull-down resistor to d2 as shown in **Schematic B**.

Software: The outline of a sketch to implement the stoplight controller with digital sensor is given below. You must complete the sketch and demonstrate to the Instructor or TA that the stoplight controller with sensor functions properly.

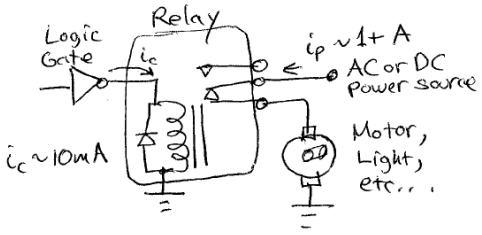
```
#define N_S_red 5              // assigning digital pins to each light
#define N_S_yellow 6
#define N_S_green 7

#define E_W_red 11
#define E_W_yellow 12
#define E_W_green 13

#define sensor 2               // NEW: sensor (switch) read by d2

#define yellow 1000            // yellow light (both N-S and E-W) = 1s (1000ms)
#define EW_g_time 2000         // E-W green for 2s
```

```
#define E_W_delay 5000          // NEW: this is the delay between when the E-W sensor is activated
                                // and when the lights begins to cycle

boolean PinSwitch = FALSE;      // NEW: here we declare Boolean variable PinSwitch that will store the
                                // value read from d2

void setup () {
  pinmode (N_S_red, OUTPUT);
  pinmode (N_S_yellow, OUTPUT);
  pinmode (N_S_green, OUTPUT);
  pinmode (E_W_red, OUTPUT);
  pinmode (E_W_yellow, OUTPUT);
  pinmode (E_W_green, OUTPUT);
                                // One line of code is missing here!
}
void loop () {


  digitalWrite (N_S_red, LOW);          // State 1: N_S green, E_W red
  digitalWrite (N_S_yellow, LOW);
  digitalWrite (N_S_green, HIGH);

  digitalWrite (E_W_red, HIGH);
  digitalWrite (E_W_yellow, LOW);
  digitalWrite (E_W_green, LOW);

do
{
                                // One line of code is missing here!
} while (PinSwitch == FALSE);           // The do loop will keep executing until the switch is closed

delay (E_W_delay);

// The rest of the code goes here to cycle through States 2, 3, and 4
```

Exercise 7-3: Controlling motor speed with a potentiometer

Theory: using a power transistor to control motor power. As described in class, the speed of a motor can be controlled using a digital PWM signal. The Arduino microcontroller has built-in hardware and software support to put out PWM signals on pins d3, d5, d6, d9, d10, and d11. However, microcontrollers are designed to receive, process and transmit information, not to supply significant power. The Arduino digital pins can supply up to 40 mA when configured as digital outputs, therefore the maximum power they can deliver is (5V)(40mA) = 0.2 W. This is plenty for a low-power LED, but not enough to run a larger motor, much less an oven or air-conditioner! Therefore an intermediate switching device that can pass larger amounts of current is needed that can turn on and off in response to a digital signal from a microcontroller.

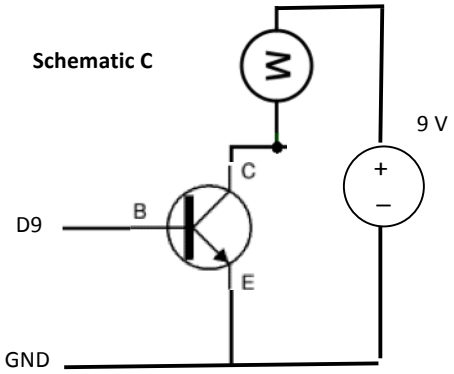
Relays use a magnetic coil energized by the digital signal to flip a mechanical switch and can deliver many amperes of current to a load. However, relays are slow to activate and can draw substantial current from the digital device. Relays are commonly used to deliver AC power to a load.



Power transistors can switch very rapidly and supply tens of Watts of power, depending on the transistor. In this lab you will use a TIP120 NPN-type transistor. NPN is a type of bipolar junction transistor (BJT) with a symbol given at right. When a BJT transistor is used as an amplifier a small current into the Base (**B**) results in a much larger current flowing from the Collector (**C**) to the Emitter (**E**). The current amplification is ~1000 for the TIP 120. Therefore we connect the digital output voltage from the Arduino between the Base and the Emitter. We then connect the load we want to power (e.g., motor) in series with the Collector or Emitter. In this way a current of ~10 mA from the Arduino can be amplified to amperes in the load. Conservation of energy applies, however, so a voltage (power) source is connected between the Collector and Emitter to provide the larger current. We will use a 9V battery for this as depicted in **Schematic C**.

Our use of this transistor is more like a two-state (digital) switch, or relay, rather than an amplifier. The transistor is considered **"off"** if V_{BE} (voltage between Base and Emitter) is small, and, ideally, no current goes into the base ($I_B = 0$), nor does current flow from the Collector to the Emitter ($I_C = 0$). Therefore, when the digital output of the Arduino (d9) is low (0V), current to the motor is switched off, and $V_{CE} = 9$V.

Schematic C

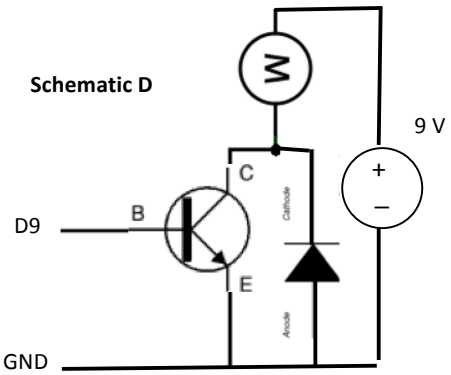


In contrast, if V_{BE} is sufficiently large, then the transistor is driven into saturation ("**on**" state) and V_{CE} drops to a small value (~ 2V) resulting in a large I_C / motor current. Therefore if d9 is high (~5V), the voltage dropped across the motor is 9 – 2 = ~7 V and the motor runs.

If a PWM signal is applied to d9, then the transistor is rapidly turned on and off and the motor spins at a speed determined by the average voltage that is applied.

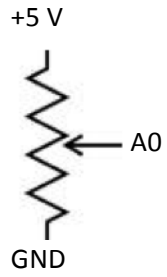
Flyback diode. A final component needed to control motor power with a transistor is a "flyback diode". A motor acts as an inductor in that it stores energy in a magnetic field that is created when current is passed through a coil of wire. Just as capacitors resist sudden changes in voltage, inductors resist sudden changes in current. Therefore when the current is switched off the motor will transiently develop a large "flyback" voltage

Schematic D



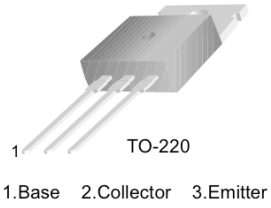
that can damage the transistor. If a diode is placed across the transistor, as shown in **Schematic D**, it will pass current to suppress the flyback voltage and protect the transistor. Note that the diode needs to be placed in the proper direction with the anode connected to ground and the cathode connected to the Collector of the transistor.

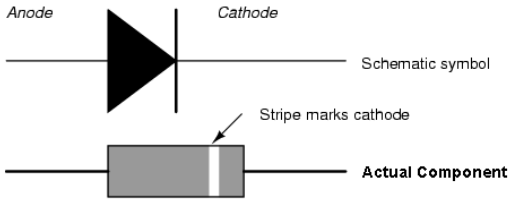
A potentiometer as an analog input device. Many of the knobs you see on instruments are potentiometers. A potentiometer is a device that has a fixed resistance between two of its terminals. However a third terminal "swipe" makes physical contact with this resistor and essentially divides the resistor into two parts. The position of the swipe along the resistor is determined by the angular position of the potentiometer shaft. If there is no current drawn from the swipe contact, then the voltage at the swipe contact is intermediate between the voltages at the two terminals in proportion to the shaft rotation. In the example shown on the right, the voltage A0 will vary between 0 and 5V as the shaft is rotated clockwise. In this lab you will connect the swipe output to A0, one of the analog inputs of the Arduino. Since this is a 10-bit Analog-to-Digital converter, it can distinguish $2^{10} = 1024$ different values. Therefore a voltage between 0 and 5 V will be converted to a number between 0 and 1023.



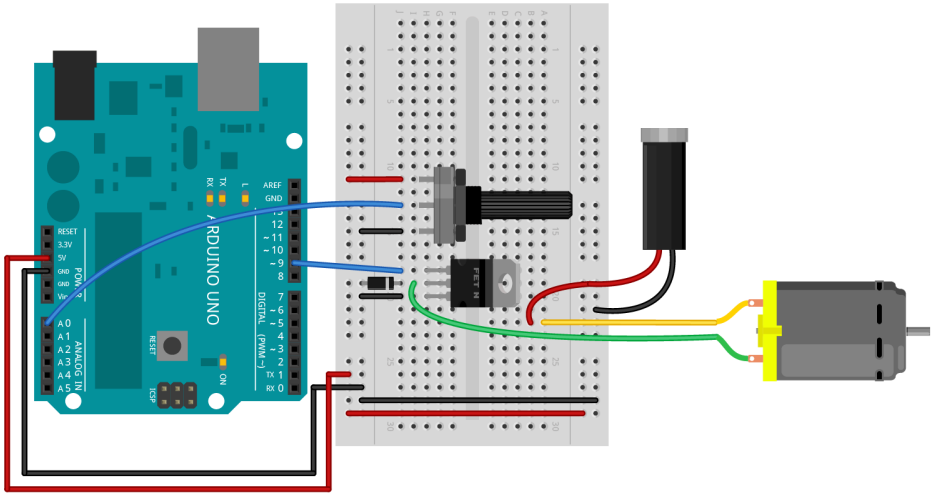
Experiment

Hardware: Connect the transistor, diode, motor (fan), and 9V battery as indicated in **Schematic D**. The d9 pin of the Arduino should be connected to the Base of the transistor and the GND pin of the Arduino to the Emitter of the transistor. Refer to the package layout given on the right as to which pin in the Base, Collector and Emitter. Also note that the diode must be inserted in the proper direction where the stripe indicates the cathode.





Connect the potentiometer with one end connected to +5 V (Vcc), the other end connected to GND, and the middle pin (swipe) connected to analog input A0 of the Arduino. A sample wiring scheme is given at right.



Software: First write a simple sketch to test that your motor control circuit works properly to by turning the motor on and off.

```
#define transistorPin 9              //d9 is connected to base of transistor

 void setup() {
   // set  the transistor pin as output:
   pinMode(transistorPin, OUTPUT);
 }

 void loop() {
   digitalWrite(transistorPin, HIGH);
   delay(5000);
   digitalWrite(transistorPin, LOW);
   delay(5000);
 }
```

Now try changing the speed of the motor using the potentiometer. To do that, read the voltage of the potentiometer using analogRead(). This returns a value from 0 to 1023. The command analogWrite() sets the PWM output to control the motor speed. However, the values for analogWrite vary from 0 to 255. Therefore to map the potentiometer reading to the PWM power output you divide by 4.

```
#define transistorPin  9    // d9 is connected to base of transistor

int sensorValue             // this integer variable stores the value read from the potentiometer

int outputValue             // this integer contains the value to send to the PWM output

 void setup() {
     pinMode(transistorPin, OUTPUT);   // set  the transistor pin as output:
  }

  void loop() {
                                        // One line of code is missing here! (read the potentiometer)
    outputValue = sensorValue / 4;      // map the sensor value to a range from 0 - 255:

                                        //One line of code is missing here! (write outputValue)
  }
```

1. Demonstrate to the Instructor or TA that you can control the speed of the motor / fan using the potentiometer.
2. Connect the PWM output (d9) and GND to the Oscilloscope and observe how the PWM signal changes as you turn the potentiometer. Show this to the Instructor or TA.

Acknowledgements: Portions of this lab are adapted from**:**
http://itp.nyu.edu/physcomp/Tutorials/HighCurrentLoads

Figures are also taken from https://www.adafruit.com/ and http://www.arduino.cc/