

MECHATRONICS TUTORIAL GUIDE

48610 Introduction to Mechanical and Mechatronic
Engineering

"Learners need to get involved with new knowledge in order to consolidate their own understanding, and this cannot be done just through hearing information being presented clearly and logically by an expert. They will almost certainly need to try to use it themselves, under different circumstances, if they are to make the knowledge their own." – Neil Mercer, The Guided Construction of Knowledge - Talk Amongst Teachers and

Acknowledgements: Thank you to my students Jason Ho, Atlas Huang and Peter de Jersey who have made significant contributions to the content and development of this guide.

Spring 2018

Terry.Brown@uts.edu.au

Table of Contents

1.	Introduction	3
2.	Week 9	3
2.1	Background	3
2.1.1	Electronics Fundamentals (1 Hour 17 Minutes)	Error! Bookmark not defined.
2.1.2	Learning Arduino (~30mins)	3
2.1.3	Installing the Arduino IDE	4
2.1.4	Wiring	6
2.1.5	Resistors	7
2.1.6	LEDs	8
2.1.7	Schematics	9
2.2	Tutorial Exercises	11
2.2.1	Exercise 1: Simple LED Circuit	11
2.2.2	Exercise 2: LED Challenge	12
2.2.3	Exercise 3: Blink	12
2.2.4	Exercise 4: Blink the LED on breadboard	13
2.2.5	Exercise 5: Blink SOS	14
2.2.6	Exercise 6: Add a Pushbutton to turn LED on and off	16
2.2.7	Exercise 7: Add a Pushbutton to Blink SOS	17
2.2.8	Exercise 8: Potentiometer with LED	17
2.2.9	Exercise 9: Potentiometer challenge	17
3.	Week 10	18
3.1	Background	18
3.1.1	Electronics Fundamentals (~ 40 mins)	Error! Bookmark not defined.
3.1.2	Learning Arduino (~ 30 mins)	18
3.2	Tutorial Exercises	18
3.2.1	Exercise 1: Use a Potentiometer to Generate an Analog Signal	18
	Code for PotRead sketch	19
3.2.2	Exercise 2: Use Analog Signal	20
3.2.3	Exercise 3: LED and Photo Interrupter	22
3.2.4	Exercise 4: Line trace sensor	23
3.2.5	Exercise 5: Line trace sensor with serial plotter	24
3.2.6	Exercise 6: Line trace sensor with digital output to control a LED	26
3.2.7	Exercise 7: WPV desktop prototype	29
	Arduino coding cheat sheet	30

1. Introduction

Mechatronics is at the forefront of innovation and can be seen everywhere; from autonomous vehicles, to home automation, to robot arms, to satellites, and more.

By the end of this module you will have the foundations required to go out and make your own mechatronic systems.

2. Week 9

2.1 Background

2.1.1 Learning Arduino

You should have watched the following sections from the [Lynda.com course Learning Arduino \(aka Up and Running with Arduino\)](#)

- Introduction (4 mins)
- 1. Getting Started (11 mins)
- 2. Electronic Components (16 min)
- 3. Arduino Uno (21 min)

Use the link from the Week 9 to 11 Folder in Weekly Learning Materials on UTSONline. Alternatively, you can access the course here:

<https://www.lynda.com/Arduino-tutorials/Up-Running-Arduino/197594-2.html?org=uts.edu.au>

2.1.2 Electronics Fundamentals

If you need to brush up on your high-school science knowledge of electricity, you should have watched the following sections of the [Lynda.com course Electronics Foundations: Fundamentals](#)

- Introduction (4 mins)
- 1. Fundamentals of electricity (33 mins)
- 2. Multi-meter measurements (12 mins)

Use the link from the Week 9 to 11 Folder in Weekly Learning Materials on UTSONline. Alternatively, you can access the course here:

<https://www.lynda.com/Development-Tools-tutorials/Electronics-Foundations-Fundamentals/197537-2.html?org=uts.edu.au>

2.1.3 Installing the Arduino IDE

The [Lynda.com course Learning Arduino](#) went through how to download and install the Arduino IDE. Only brief instructions are provided here. The Arduino IDE can be downloaded from the link shown below.

You will be asked if you want to donate, if you do not want to, just click the “just download” button. Once downloaded and installed, run the Arduino IDE. It will look something like Figure 1. When you plug the board in via the USB cable, drivers should be downloaded and installed automatically. Some computers (especially macs) sometimes have a problem with this. If you have trouble, go to the [Arduino troubleshooting page](#). You can also refer to [this online guide](#).

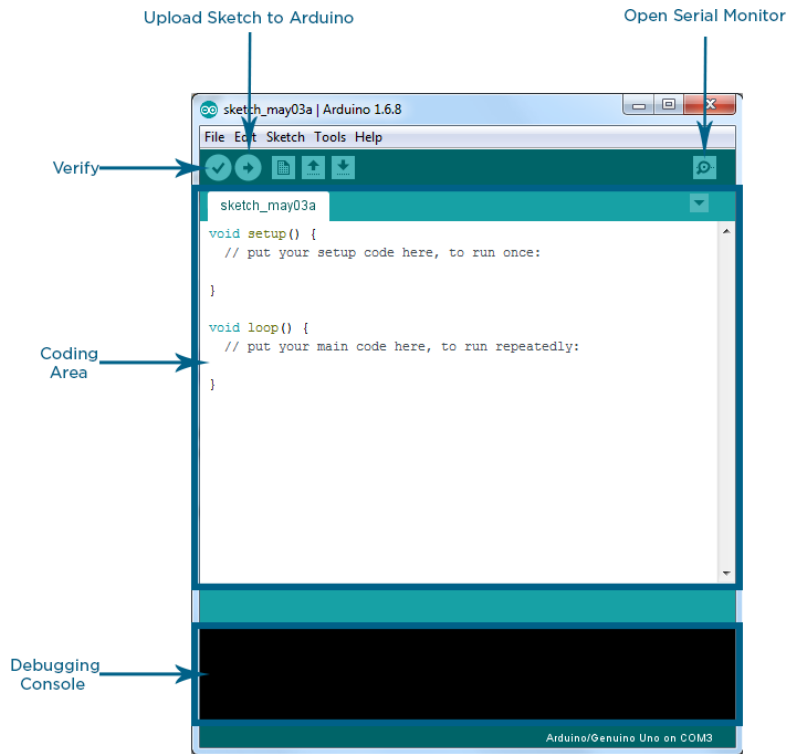


Figure 1 - Arduino IDE

<https://www.arduino.cc/en/Main/Software>

Before you can interact with, and upload code to, your Arduino, it's important that you select the correct board. After you plug in your board, select it from the Tools -> Board menu.

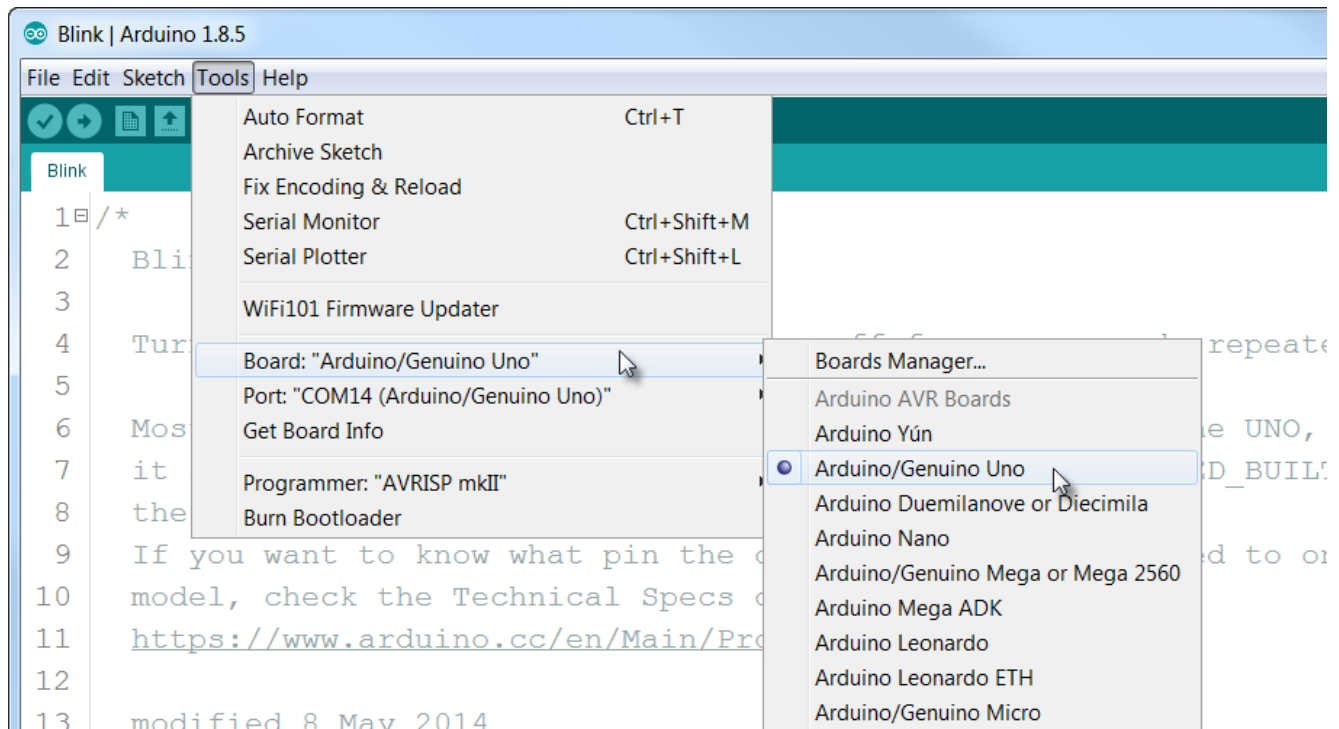
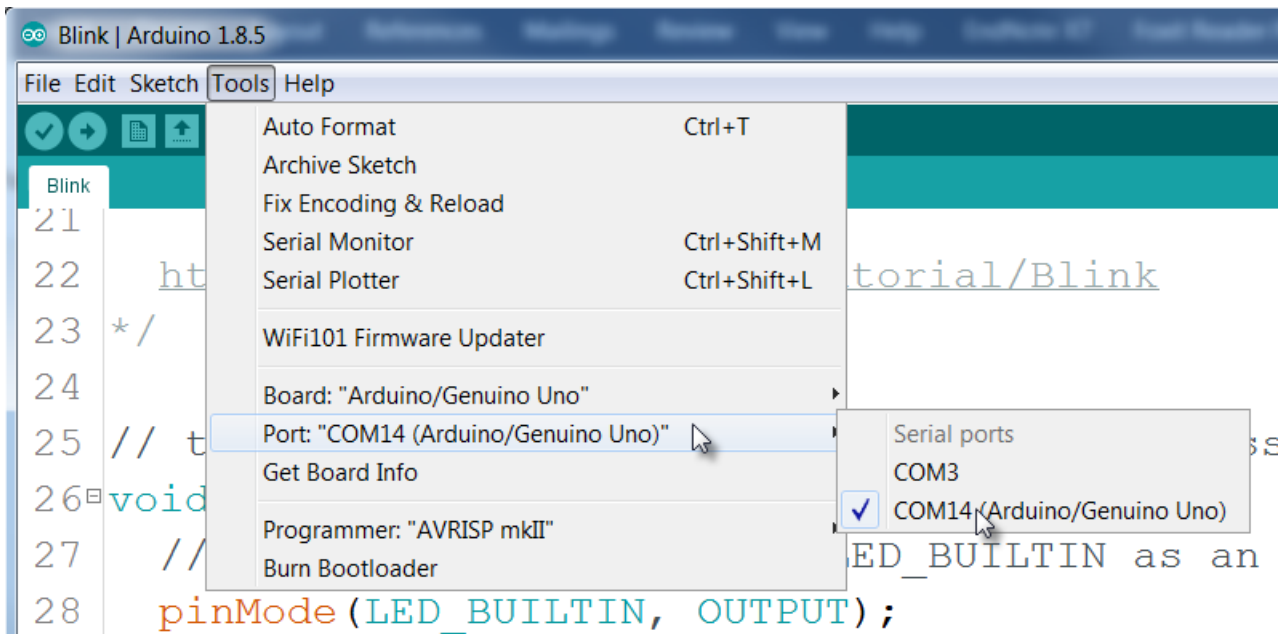


Figure 2 - Make sure you select your board after plugging in

Before you can upload code to your Arduino, it's important that you select the correct port your board is on. After you plug in your board, select it from the Tools -> Port menu (see below).



2.1.4 Wiring

Wiring is required to connect electrical components together according to the schematics and circuit diagrams. It is common practice to start from the power source and use electrical wires/cables to connect each component one after another.

The process is similar to connecting a piping system. In this subject, we use a breadboard and jumper cables for wiring (Refer to Figure 3 & 4). Breadboard and jumper cables are often used for prototyping because they can be easily assembled and disassembled.

Have a look at Figure 4, which shows an x-ray of the board. You can see that the pins in the middle of the board are connected horizontally, and the rails on the outside are connected vertically.

Breadboards should only be used for prototyping. Once circuit design has been finalised, printed circuit boards (PCB) should be produced and used.

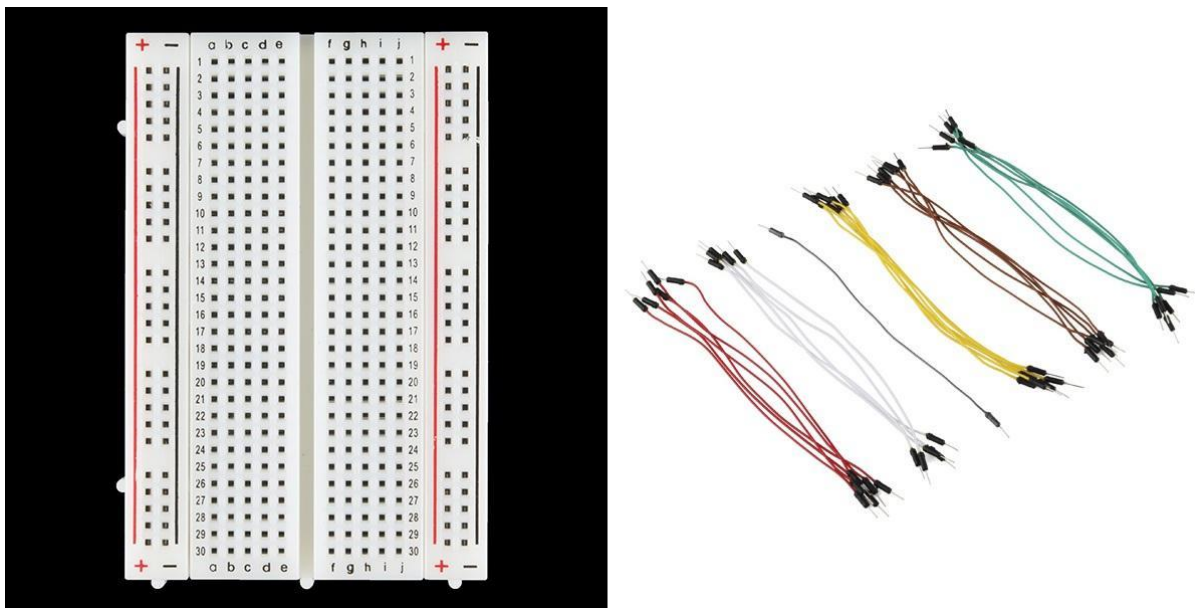


Figure 3 - Breadboard & Jumper Cables

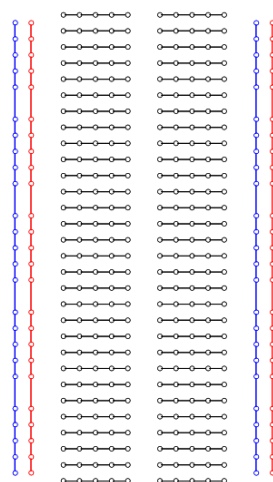


Figure 4 - Breadboard "X-Ray"

2.1.5 Resistors

If you look at circuit diagrams such as the one in Figure 9 carefully, you will notice that there is always a resistor in front of each LED. The resistor is used to reduce electrical current passing through the LED and to ultimately avoid damaging the LED.

Use Ohm's Law, V (voltage, volts) = I (current, amps) x R (resistance, ohms) to determine required resistor. There are [online calculators](#) that help you to calculate the most suitable resistor size for your project.

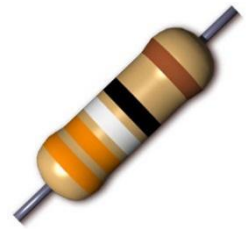


Figure 5 Resistor Example

The easier way to work out the resistance of a resistor is to use a multi-meter such as the one shown here. Basic multi-meters can be very inexpensive and can be found for less than \$10. For example, here's a link to a basic multi-meter from Jaycar:

<https://www.jaycar.com.au/low-cost-digital-multimeter-dmm/p/QM1500>



Figure 6 - Digital Multi-meter

The harder way to work out the resistance of a resistor is to use the colour markings on the resistor to determine the resistor value. This is useful when you do not have a multi-meter.

Here are the steps to read the colour code:

1. Identify the band that is slightly farther away from the others (e.g. the brown band in Figure 7)
2. Position the band identified in Step 1 to the right-hand side
3. Read the colour bands from left to the right and ignore the last one on the right
4. Different colour represents different value:

4-Band-Code

2%, 5%, 10% 560k Ω ± 5%

COLOR	1 ST BAND	2 ND BAND	3 RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

5-Band-Code

0.1%, 0.25%, 0.5%, 1% 237 Ω ± 1%

Figure 7 Resistor Colour Code

The resistor value in Figure 5 is 339Ω with +/-1% tolerance.

2.1.6 LEDs

LEDs (Light emitting diodes), like all diodes, do not work if they are plugged in the wrong way.

Electrical current goes 'into' the LED at the anode (positive terminal) and 'comes out' at the cathode (negative terminal). Figure 8 shows a few tips on how to identify the anode and cathode.

LEDs do not follow Ohm's Law. They have a voltage drop across them and have a safe, or limiting, current that may flow through them without damaging the LED. Resistors must be used to limit the amount of current flowing through a LED. Use Ohm's Law to work out the required resistance. First, subtract the voltage drop across the LED from the voltage in the circuit (or part of the circuit), then divide by the allowable current. There are also [online calculators](#) to help you work out the required resistor.

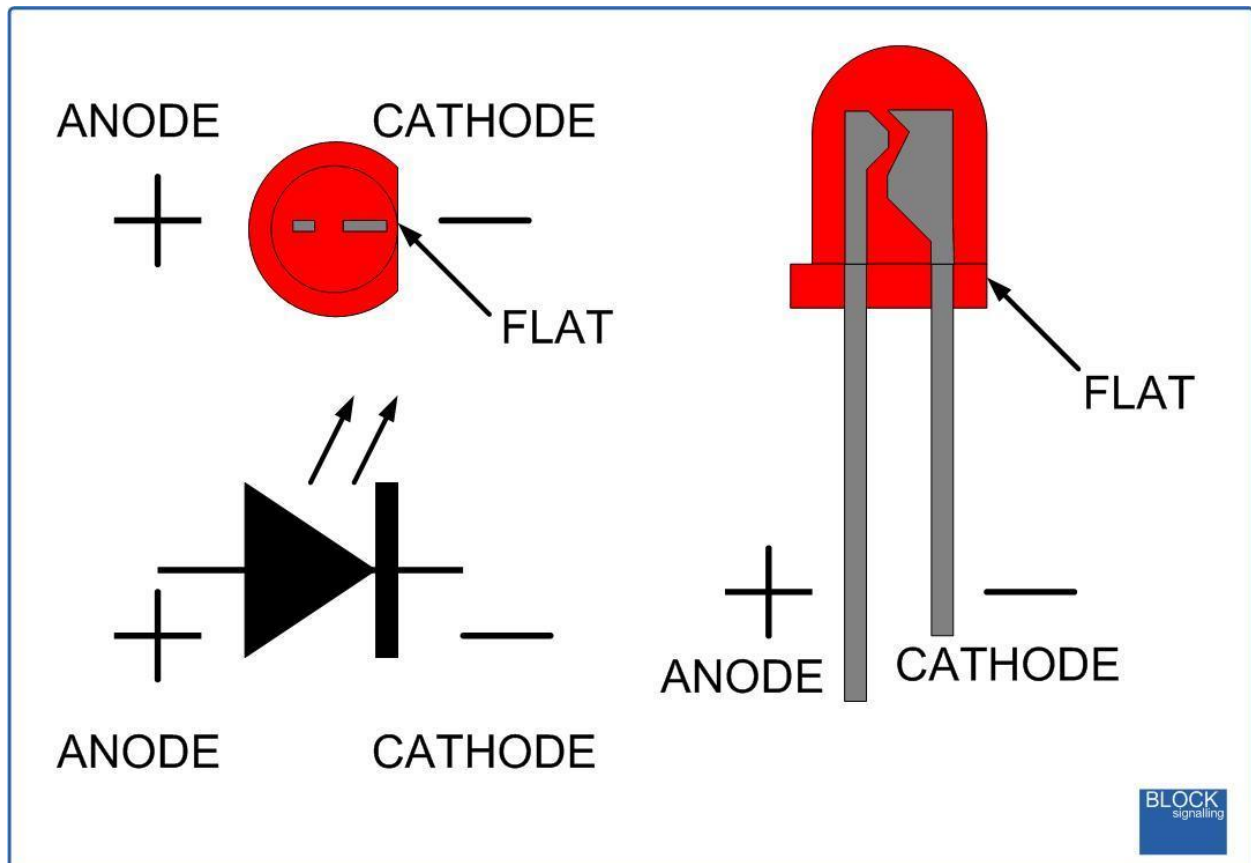


Figure 8 How to identify anode and cathode of an LED

2.1.7 Schematics

An important step to learning circuitry is to learn how to read schematics. Figure 9 is the schematic of an Arduino board. It is for demonstration purpose only, it is **NOT a requirement to understand schematics as complicated as the one below, this is an example of what you might be using in industry**. The tutorial exercises will help you to start reading, understanding and implementing simple schematics. Figure 10 lists a few electrical symbols that are frequently used in electronics projects. These electrical symbols help you to identify the components in the schematics. The highlighted ones in Figure 10 are used in this subject and it is a requirement to know how they are connected. You can test yourself by identifying all the LEDs in Figure 9. Do you notice that there is a component that is always next to a LED?

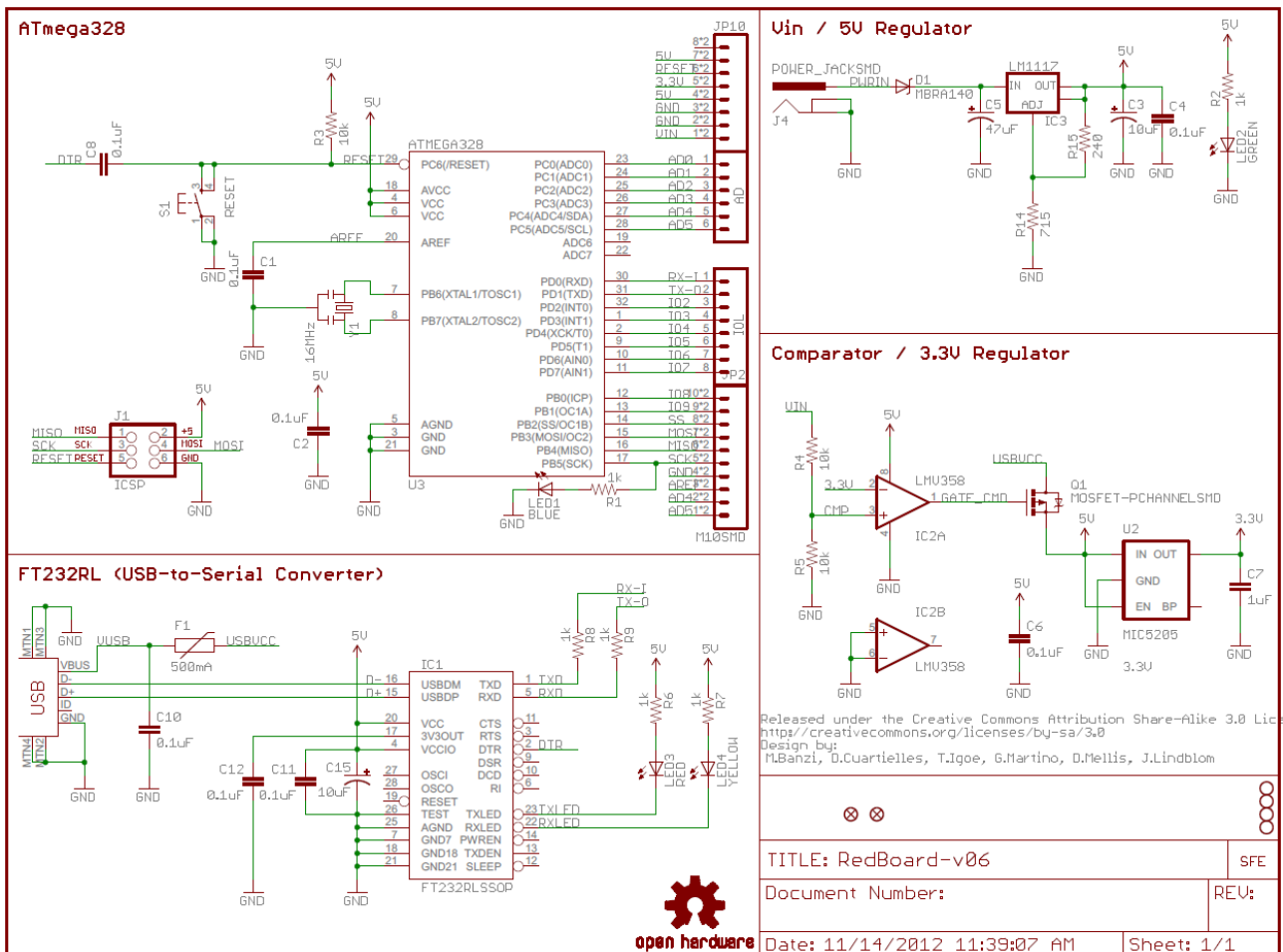


Figure 9 Schematics of Sparkfun RedBoard

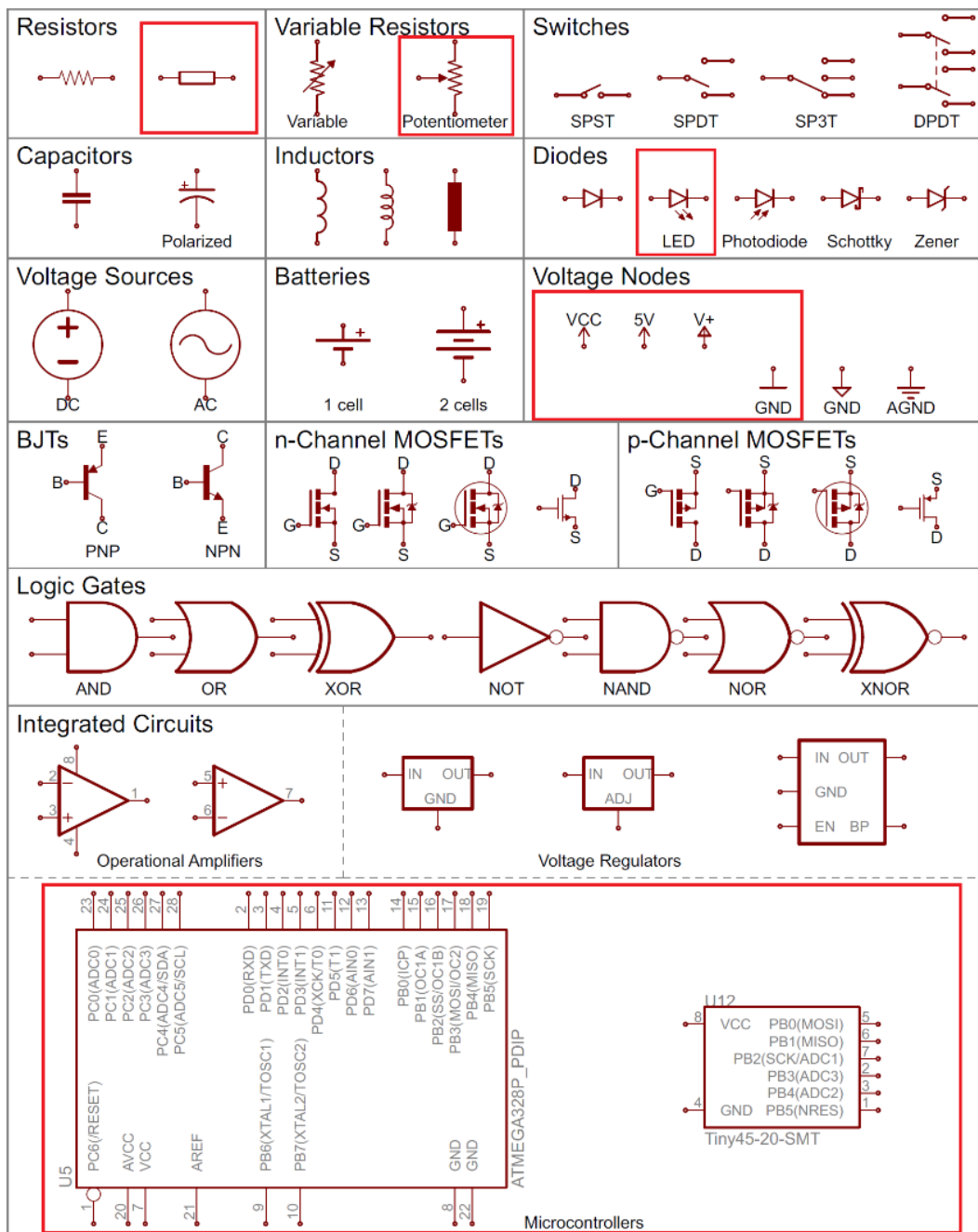


Figure 10 - Electrical Symbols

2.2 Tutorial Exercises

For these exercises we have provided you with a mechatronics kit that has all the components needed (you will need to share one kit between 2-3 students). However, you will need to have a computer with a USB port to connect and run the Arduino board. The power source provided in this tutorial is through the Arduino board. We will use the Arduino as a power source and use its microcontroller and code to control the circuits we create.

For these exercises we have provided both the schematic and the wiring diagrams. You should make sure you use and refer to both as you create the circuit. Usually in practice you would not have a wiring diagram, just the schematic. Resist the temptation to just copy the wiring diagrams. As you work your way through the exercises try to first implement the schematic without referring to the wiring diagrams.

The Arduino should ALWAYS be disconnected whenever you are wiring. Make sure the Arduino is unplugged from power (i.e. disconnect the USB cable).

2.2.1 Exercise 1: Simple LED Circuit

Attempt to wire up the circuit shown below as a schematic. Make sure the Arduino is unplugged from power (i.e. disconnect the USB cable). The Arduino should ALWAYS be disconnected whenever you are wiring.

Use the digital multi-meter (DMM) to measure the resistor values and check that you have the correct resistor. Use of the DMM is explained in the [Lynda.com course Electronics Foundations: Fundamentals](#). If you don't know how to use the DMM, ask other students at your table if they know and ask them to show you. If no students at the table know, ask the tutor.

If you don't have a red LED, use whatever colour you have.

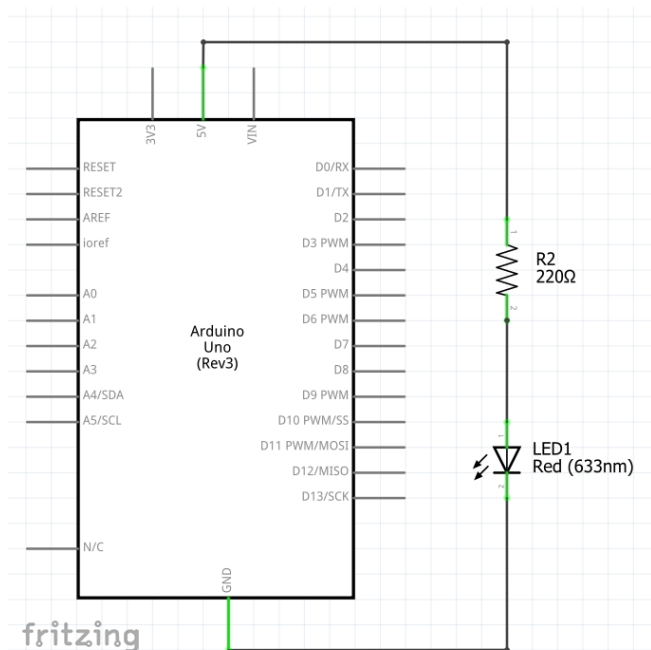


Figure 11 - Arduino LED Schematic

Two alternative wiring diagram solutions are shown below. Do you understand the difference between the two? Do you understand why they both work? Discuss with other students at your table.

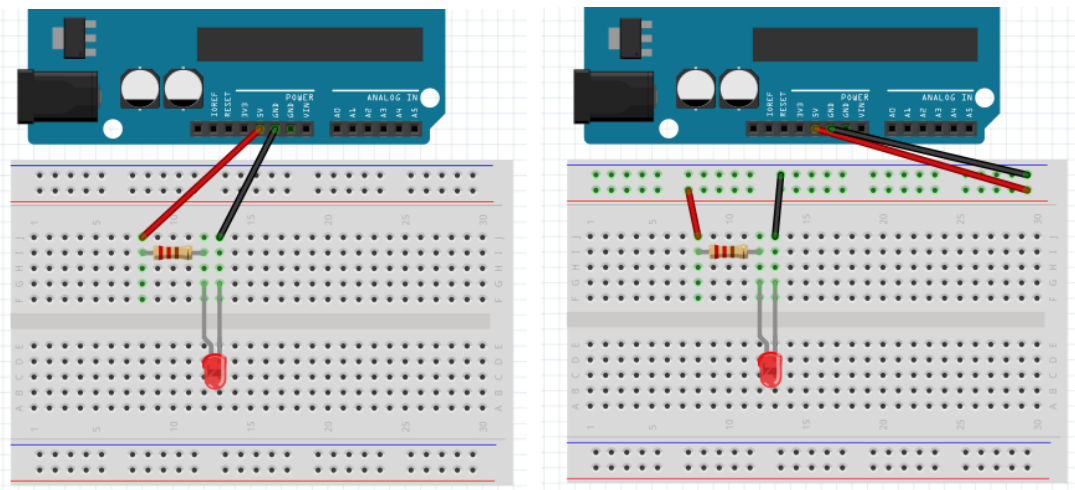


Figure 12 - Wiring diagrams for two alternate wiring methods

Connect the USB cable to the Arduino to provide power to the circuit. The LED should light up. If it doesn't, check that the power light indicator and the onboard LED on the Arduino board are lit up. If they are not lit up there may be a problem with the cable or the Arduino. If they are, check that the LED is connected the right way. If it still does not light up, try another LED.

2.2.2 Exercise 2: LED Challenge

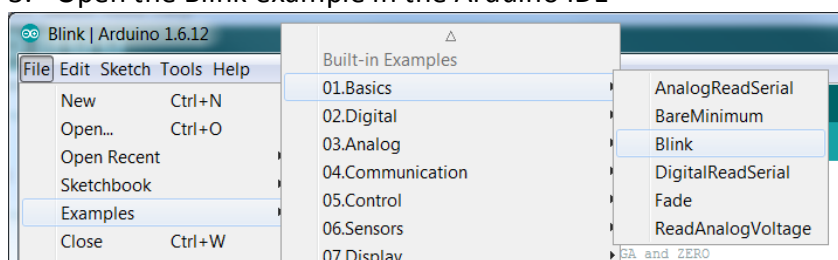
In this challenge, you are required to think about and modify the circuit in Exercise 1 and complete the following:

1. Calculate the current in the circuit
 - a. Assume LED resistance is negligible (note that LEDs do not follow Ohm's Law)
 - b. Attempt to use LED data: Typical working voltage of a LED: yellow/green/red/orange 1.8-2.2V, blue 2.6-3.0V.
 - c. Recall/look up Ohm's Law and/or use an [online calculator](#).
2. Connect another LED in series with the existing LED
3. Connect another LED in parallel with the existing LED (make sure there is a resistor limiting current to all LEDs)
4. Discuss with your group the current value you calculated and the differences in LED brightness for each circuit

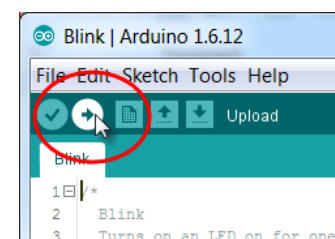
2.2.3 Exercise 3: Blink

In this exercise you will use the Arduino microcontroller and code to make the Arduino's onboard LED blink. You will need to:

1. Download, install and run the Arduino IDE software [as shown above](#).
2. Connect the Arduino board to your computer (via USB cable)
3. Open the Blink example in the Arduino IDE



4. Upload the code to the Arduino board



If all is well, you should see something like the following:

```

22 | http://www.arduino.cc/en/Tutorial/Blink
Done uploading.
Sketch uses 928 bytes (2%) of program storage space. Maximum is 32256 byte
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes fo
31 Arduino/Genuino Uno on COM14

```

If it takes a long time and you get an error something like the following, it is most likely because you have not selected the correct port. Refer to the instructions [shown above](#).

```

20 | This example code is in the public domain.
Problem uploading to board. See http://www.arduino.cc/en/Guide/Troubleshooting#upload for suggestions. Copy error messages
avrdude: stk500_recv(): programmer is not responding
avrdude: stk500_getsync() attempt 10 of 10: not in sync: resp=0x9e
Problem uploading to board. See http://www.arduino.cc/en/Guide/Troubleshooting
31 Arduino/Genuino Uno on COM3

```

- Once you have uploaded the code and observed the LED blinking, change the delay values, i.e. `delay(1000)` in the code, upload the code again and observe how the LED's blinking frequency changes.

2.2.4 Exercise 4: Blink the LED on breadboard

In this exercise we'll use the Arduino to control the LED on our breadboard.

- Unplug the USB
- Disconnect the lead from the 5V supply on the Arduino and plug it into digital pin 13.
- Plug the USB cable back into the Arduino

You should see the LED on the breadboard blinking in time with the Arduino onboard LED.

- Adjust the delay values in the code, re-upload the code to the Arduino and observe both LEDs blinking together at the adjusted rate

The onboard LED can be referred to as `LED_BUILTIN` in code and is connected to digital pin 13.

- Replace `LED_BUILTIN` with 13 in the `pinMode` function and the two `digitalWrite` functions.
- Re-upload the code and observe that nothing changes in the functioning of the LEDs
- Now change the three 13s to 12 in the code and upload to the Arduino. Now neither LED light up.
- Unplug the Arduino. Disconnect the lead from digital pin 13 and connect it to Pin 12. Plug in the Arduino. You should now see only the LED on the breadboard blinking.

Writing a reference number in multiple places like this is a very BAD way to write code. If we decided that we want the LED to be controlled from PIN 11 we would have to find all the places where we have used 12 in the code. Instead, we should declare a variable, i.e. give the number a name that can be referenced anywhere in the code. We will declare the variable as an integer. E.g.

- add the following to the code

```

int myLed = 12;
    just above

```

// the setup function runs once when you press reset or power the board

```

void setup() {

```

then replace everywhere you had 12 with `myLed`. Upload the code and observe that nothing changes.

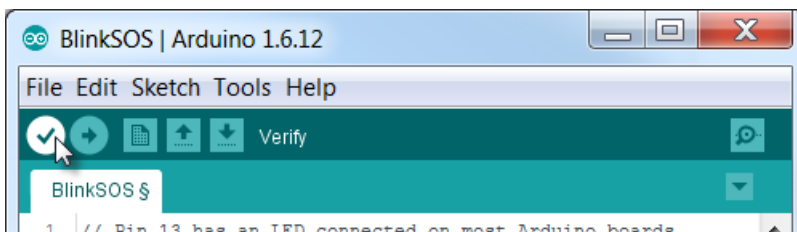
- Now change `myLed = 12` to `myLed = 11` and upload the code. The LED should stop blinking.

- Now unplug the Arduino. Disconnect the lead from digital pin 12 and connect it to Pin 11. Plug in the Arduino. You should now see only the LED on the breadboard blinking again.

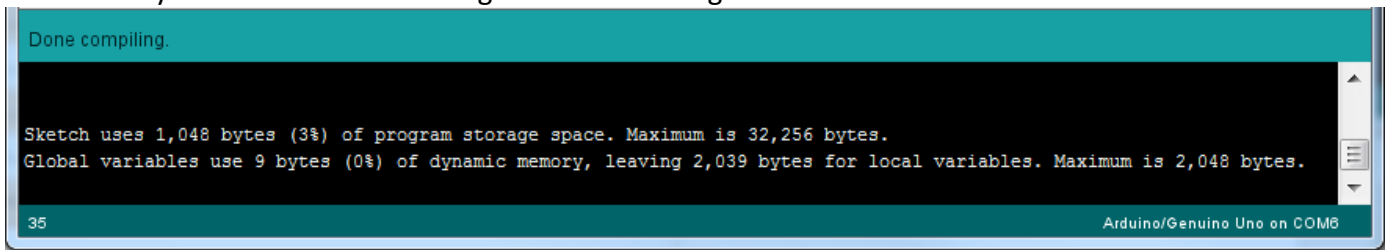
2.2.5 Exercise 5: Blink SOS

In this exercise we'll modify the code from the previous exercise. By the end of the exercise we'll have the Arduino blinking SOS in Morse code.

- Save the Blink sketch as BlinkSOS
- Modify the code so the on-board LED blinks SOS (see Morse code table and starter code below)
- Verify that the code is correctly written. This is called "compiling" the code. It only tells you if you have used the language correctly. It doesn't tell you if your code will do what you think it should/will do.



If all is well you should see something like the following:



If not, you will get an error message with an indication of where the compiler thinks the error is.

Note: SOS in Morse code is 3 times short, 3 times long and 3 times short (refer to the chart below)

Letter	Morse	Letter	Morse	Digit	Morse	Letter	Morse
A	.-	N	-.	0	-----	Ä	.-.-
B	...-	O	---	1	.-----	Á	...-
C	-.-.	P	.-.-	2	..----	Å	...-
D	-. .	Q	---.	3	...--	Ch	----
E	.	R	.-.	4-	É	..-..
F	..-.	S	...	5	Ñ	--..
G	--.	T	-	6	-....	Ö	---.
H	U	..-	7	--...	Û	..--
I	..	V	...-	8	---..		
J	.-.-	W	.-.-	9	----.		
K	-. -	X	-.-.				
L	.-..	Y	-.--				
M	--	Z	---.				

*Taken from <https://morsecode.scphillips.com/morse2.html>

*In the table, "." means short and "-" means long

You can use the code below to get started. You can copy, paste and then modify parts of this code to create your own section of code.

```

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // Flash the LED in the 'S' Sequence.
  // This is three 'dots' - or 3 short flashes
  // Flash once
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
level)
  delay(100); // wait for a short period of time
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage
LOW
  delay(100); // wait for a short period of time
  // Flash a second time
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
level)
  delay(100); // wait for a short period of time
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage
LOW
  delay(100);
  // Flash a third time
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
level)
  delay(100); // wait for a short period of time
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage
LOW
  delay(100);

  //Flash the LED in the 'O' Sequence.
  // This is three 'dashes' - or 3 long flashes

  // ----- INSERT YOUR CODE HERE! ----- //

  //Flash the LED in the 'S' Sequence.
  //This is three 'dashes' - or 3 long flashes
  //Wait a little while so that our SOS messages don't blend together

  // ----- INSERT YOUR CODE HERE! ----- //

  //Wait a few seconds so that there's a pause between our SOS messages
  delay(3000);
}

```

If you are thinking “there must be a better way than writing similar code repetitively in order to do this”, that’s good. Because there is. But we need to use control structures and subroutines or subfunctions. We will do this next week.

2.2.6 Exercise 6: Add a Pushbutton to turn LED on and off

In this exercise, you will need to:

- Find the Button example under Examples/ 02.Digital/
- Connect a 10K resistor and the push button to pin 2 (refer to the schematic and wiring diagrams below) and also <https://www.arduino.cc/en/tutorial/button>. Have we wired a pull-up resistor or a pull-down resistor?

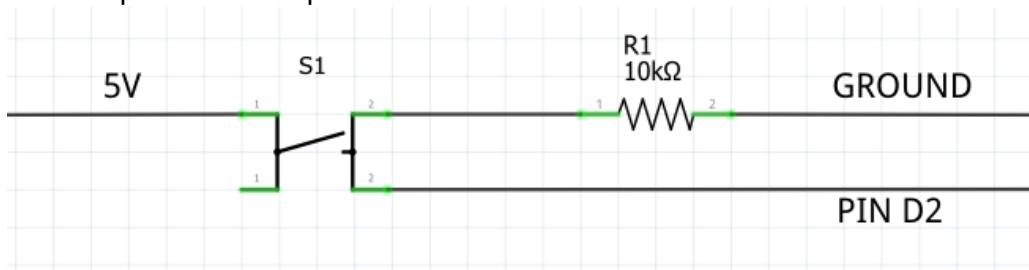


Figure 12 - Pushbutton Schematic

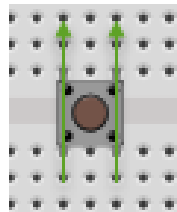
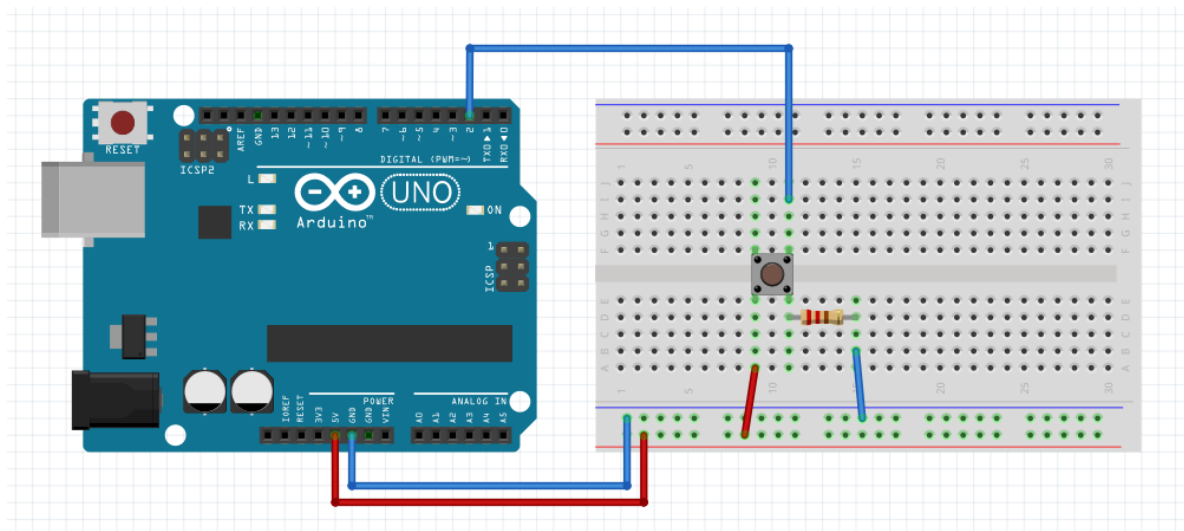


Figure 13 - Connect the pushbutton across the middle of the breadboard like this



- Upload the code

Test whether the code is working.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW.

We can also wire the circuit the opposite way, with a pullup resistor keeping the input HIGH, and going LOW when the button is pressed. Doing this, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.



2.2.7 Exercise 7: Add a Pushbutton to Blink SOS

In this exercise, you will need to:

- Understand what you have done in the previous exercises
- Adjust the code and your circuit so that when the push button is pressed, the LED on the breadboard blinks SOS

2.2.8 Exercise 8: Potentiometer with LED

A potentiometer (or 'pot' for short) is a variable resistor. Create the circuit shown below.

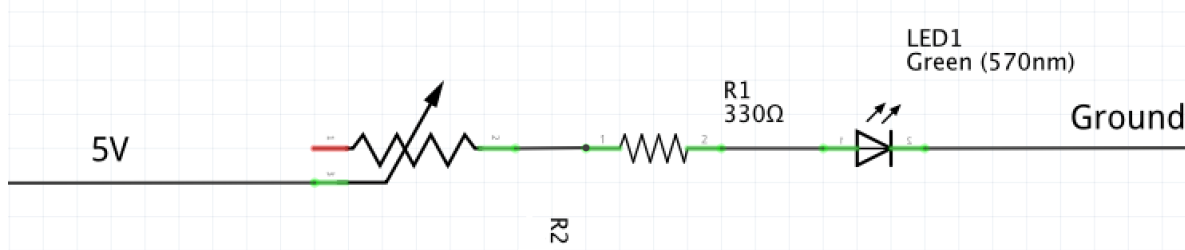


Figure 14 - Arduino Potentiometer Schematic

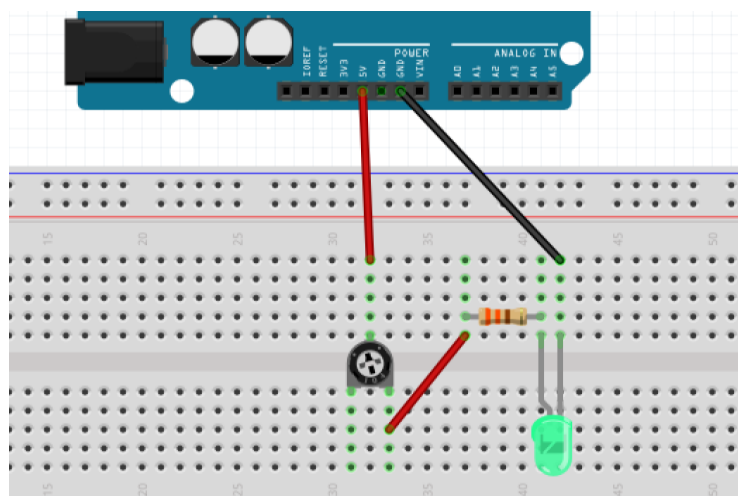


Figure 15 - Potentiometer Wiring

If you don't have a 330Ω resistor a 220Ω will do. If you don't have a green LED, any other colour will do.

Try turning the potentiometer and observe how the brightness of the LED changes.

2.2.9 Exercise 9: Potentiometer challenge

A potentiometer (or 'pot' for short) is a variable resistor. Create the circuit shown below.

- Think about how you can add a different colour LED to the circuit so that when one LED gets brighter the other LED gets dimmer and vice versa.
- Discuss with your group and draw a schematic.
- Don't forget to include a resistor in series with the second LED
- Show your schematic to your tutor before you wire it up to avoid damaging any component

Solution shown at bottom of page but have a go at it yourself first.

Exercise 9 solution: connect the second LED to the spare pin of the potentiometer

3. Week 10

3.1 Background

3.1.1 Learning Arduino

You should have watched the sections indicated in the [previous week's background](#) and the following sections from the Lynda.com course **Learning Arduino (aka Up and Running with Arduino)**

- 8. Advanced Projects (at least *Programming the Uno to output Morse code* 6m 53s)

Use the link from the Week 09 and 10 folder in Weekly Learning Materials on UTSONline.

Alternately, you can access the course here:

<https://www.lynda.com/Arduino-tutorials/Up-Running-Arduino/197594-2.html?org=uts.edu.au>

3.1.2 Electronics Fundamentals (~ 40 mins)

You should have watched the sections indicated in the [previous week's background](#) and the following sections from the Lynda.com course **Electronics Foundations: Fundamentals**

- 3. Power (27 mins)

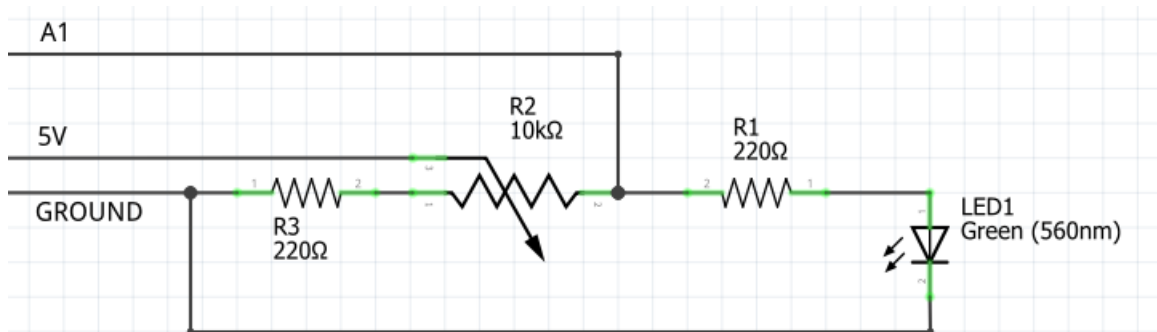
Use the link from the Week 09 and 10 Folder in Weekly Learning Materials on UTSONline.

3.2 Tutorial Exercises

The power source provided in this tutorial is through the Arduino board. We will use the Arduino as a power source and use its microcontroller and code to control the circuits we create.

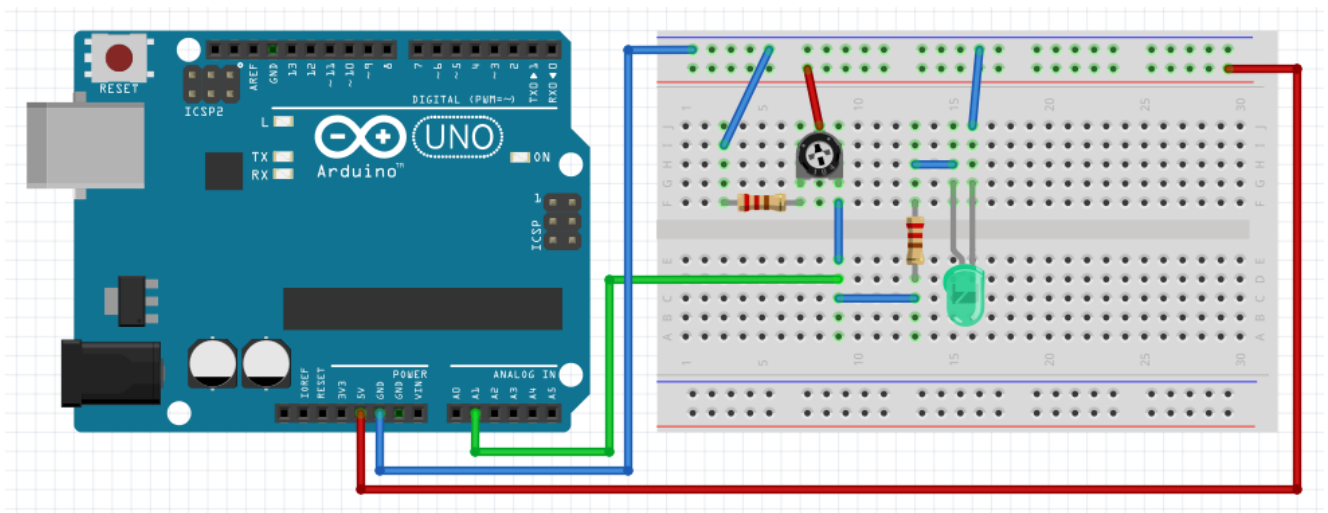
3.2.1 Exercise 1: Use a Potentiometer to Generate an Analog Signal

In this exercise you will use the Arduino IDE's Serial Monitor to view the voltage change in the circuit as you adjust the potentiometer. Create the circuit shown below. Make sure the Arduino is unplugged from power (i.e. disconnect the USB cable). The Arduino should ALWAYS be disconnected whenever you are wiring.



If you don't have a 220Ω resistor a 330Ω will do. If you don't have a green LED, any other colour will do.

An indicative wiring diagram is provided below. Try to complete the wiring BEFORE looking at the wiring diagram below. Make sure you understand how to read the schematic and how it gets implemented in a physical circuit. Do not just copy the wiring diagrams. Your wiring doesn't have to be exactly the same as the examples shown.



Before connecting the USB cable to power the board:

- Check your wiring. Get a person in the group who has not done (much/any of) the wiring to check the wiring. If in doubt, ask the tutor to check your wiring.
- Once checked, connect the USB cable to the Arduino.
- Adjust the potentiometer to observe the LED brightness change.

Now we will use the Arduino to read the voltage in the circuit.

- Create a new Sketch called PotRead. Copy and paste the [code provided below](#) (replace all of the default code) and upload the sketch to the Arduino board.

Code for PotRead sketch

```

/*
Potentiometer Analog Sensor
UTS 48610 - Introduction to Mechanical and Mechatronics Engineering - Autumn 2017
Written By Jason Ho, modified by Terry brown
Any Questions? Try finding the answer for yourself before just asking your tutor.
Feeling confident? Try modifying or adding to this code to add special features for your WPV i.e Flashing lights or
even a data transmitter.
Additional Notes:
Camelback notation: You will see words like "statusLightsAreGood" with a lowercase first letter.
*/

```

```

#define Serial_Update_Interval 500
#define Analog_Pin A1

```

```

unsigned long oldMillis; // this stores the last value of millis when the Serial monitor printed the value of A1

```

```

void setup()
{
  Serial.begin(57600); // this connects the serial port
  pinMode(Analog_Pin, INPUT); // this sets the pin's mode to be an input
  Serial.println("-----");
  Serial.println("UTS IMME Analog Pot Reader");
  Serial.println("-----");
  Serial.println("");
}

```

```

void loop()
{
  if (millis() - oldMillis >= Serial_Update_Interval) { // this checks if the interval time has passed (to avoid
  spamming the monitor)
    Serial.print("Analog ");

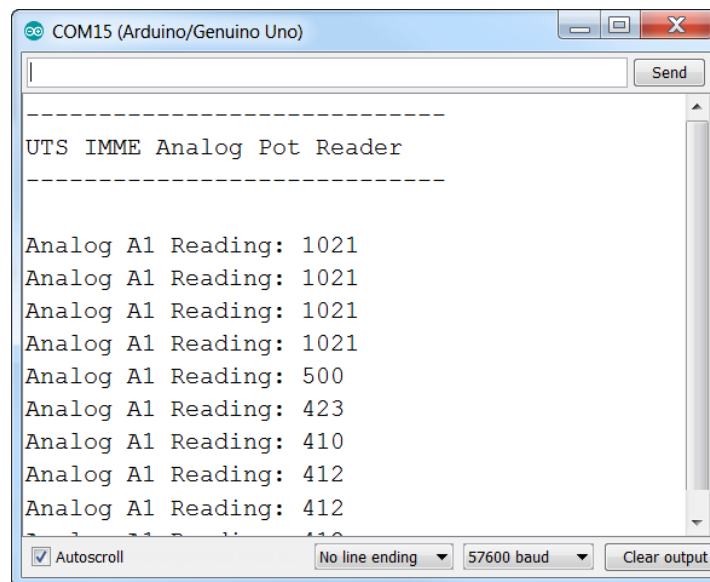
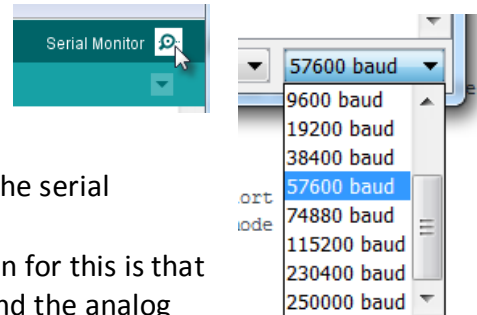
```

```

Serial.print("A1");
Serial.print(" Reading: ");
Serial.println(analogRead(Analog_Pin));
oldMillis = millis();// this stores the current time which will be used to check if the interval time has passed for
the next iteration
}
}

```

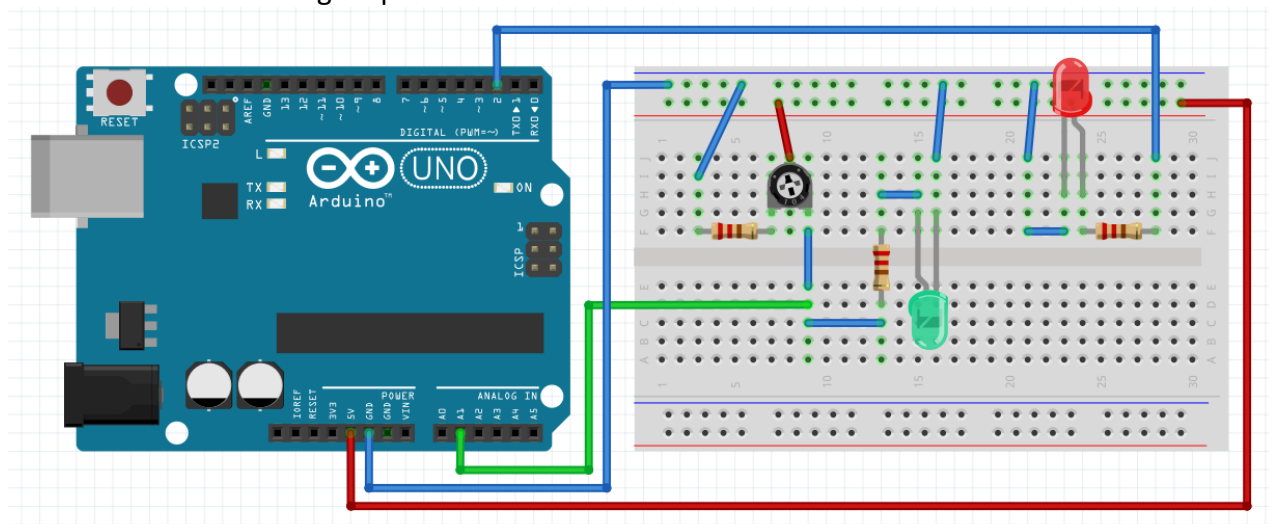
- Use the serial monitor to see the output from the potentiometer
- You will need to change the baud rate to match the rate set in the code
- Adjust the potentiometer and observe the output in the serial monitor and the LED brightness change.
- the numbers will vary between 0 and 1023. The reason for this is that the Arduino is doing an analog to digital conversion and the analog pin is converting a voltage between 0 and 5V to a discrete number.



3.2.2 Exercise 2: Use Analog Signal

In this exercise you will use the value recorded by the analog pin to adjust the brightness of another LED.

- Add a red (or any other colour) LED to the circuit (don't forget the resistor) and connect the LED to digital pin 2 as shown below:



- We will add the code `#define myLED 2` which uses `#define` to declare a constant (`#define` is a way of defining a constant in C. It can be useful but requires care in its use. Read more about it [here.](#))
- we can now use the name `myLED` to refer to digital pin 2 elsewhere in the code
- we now need to set pin 2 as an output. Have go at doing that on your own before looking to see if you can find it in the modified code provided below.
- We will also use the *if...else* control structure to control the new LED based on the value of A1. Read more about it [here](#). Can you find where it is used in the code and understand how it works?
- Create a new sketch. Name it whatever you like. Cut and paste and upload the following code:

```

/*
Potentiometer Analog Sensor to control LED
UTS 48610 - Introduction to Mechanical and Mechatronic Engineering - Autumn 2017
Written By Jason Ho, modified by Terry Brown
*/

#define Serial_Update_Interval 500
#define Analog_Pin A1
#define myLED 2

unsigned long oldMillis; // this stores the last value of millis when the Serial monitor printed the value of A1

void setup()
{
  pinMode(myLED, OUTPUT); // this sets the pin's mode to be an output
  Serial.begin(57600); // this connects the serial port
  pinMode(Analog_Pin, INPUT); // this sets the pin's mode to be an input
  Serial.println("-----");
  Serial.println("UTS IMME Analog Pot Reader");
  Serial.println("-----");
  Serial.println("");
}

void loop()
{
  if (millis() - oldMillis >= Serial_Update_Interval) { // this checks if the interval time has passed (to avoid
  spamming the monitor)
    Serial.print("Analog ");
    Serial.print("A1");
    Serial.print(" Reading: ");
    Serial.println(analogRead(Analog_Pin));
    oldMillis = millis(); // this stores the current time which will be used to check if the interval time has passed for
  the next iteration
  }
  if (analogRead(Analog_Pin) >= 700) { // this checks if Analog_Pin is greater than or equal to 700
    digitalWrite(myLED, HIGH); // turn the LED on (HIGH is the voltage level)
  }
  digitalWrite(myLED, LOW); // turn the LED off by making the voltage LOW
}

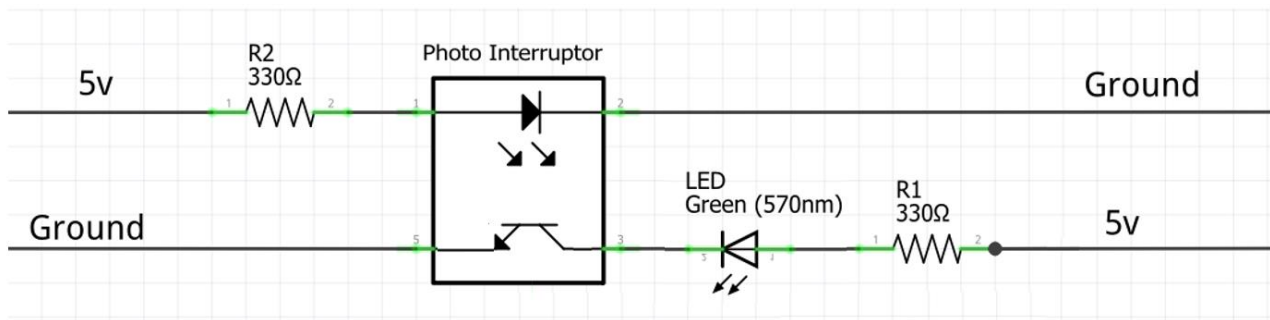
```

- Observe the serial monitor values and watch the new LED turn on and off as you adjust the potentiometer.
- Adjust the code to experiment with how it works to further your understanding.

3.2.3 Exercise 3: LED and Photo Interrupter

In this exercise you will use a photo interrupter to turn a LED on and off

- Attempt to create the circuit shown in the schematic below. Be careful. Have your circuit checked before connecting power.
- Refer to the photo interrupter drawings and schematic when connecting the photo interrupter (be careful, the terminal lettering isn't obvious. Look carefully at the top of the photo interrupter and you will see the terminal letters. Also note the chamfered edge on the top left edge (front view) to help you correctly orient and connect the photo interrupter).
- Attempt to interpret the circuit schematic and the photo interrupter drawing and schematic and wire up your circuit before looking at the wiring diagram provided further below.



If you don't have a 330Ω resistor a 220Ω will do. If you don't have a green LED, any other colour will do.

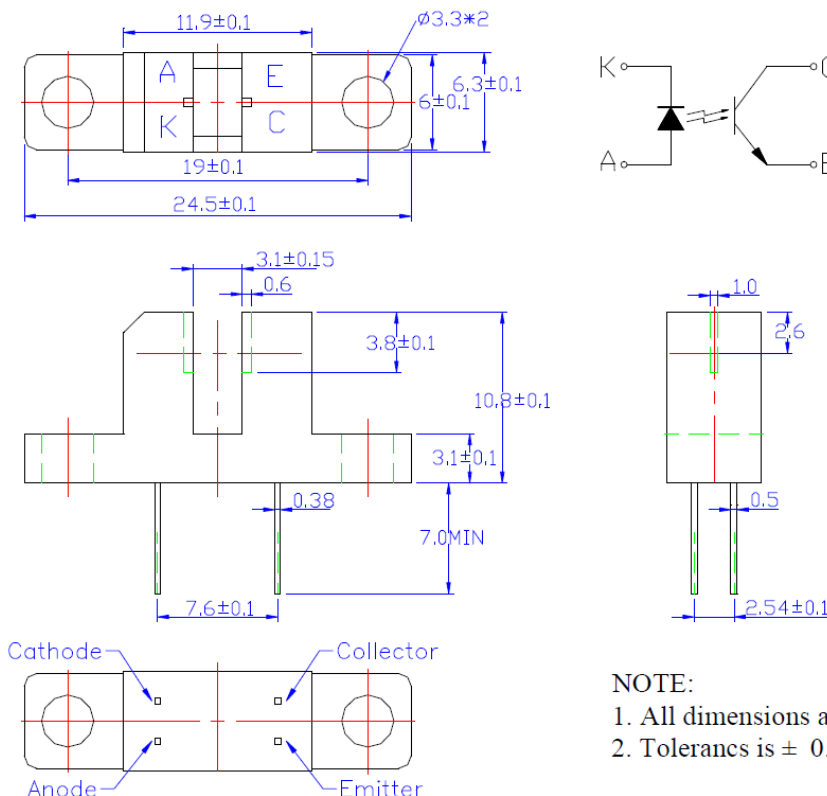


Photo Interrupter drawing and schematic (note that the schematic is the view from the bottom)

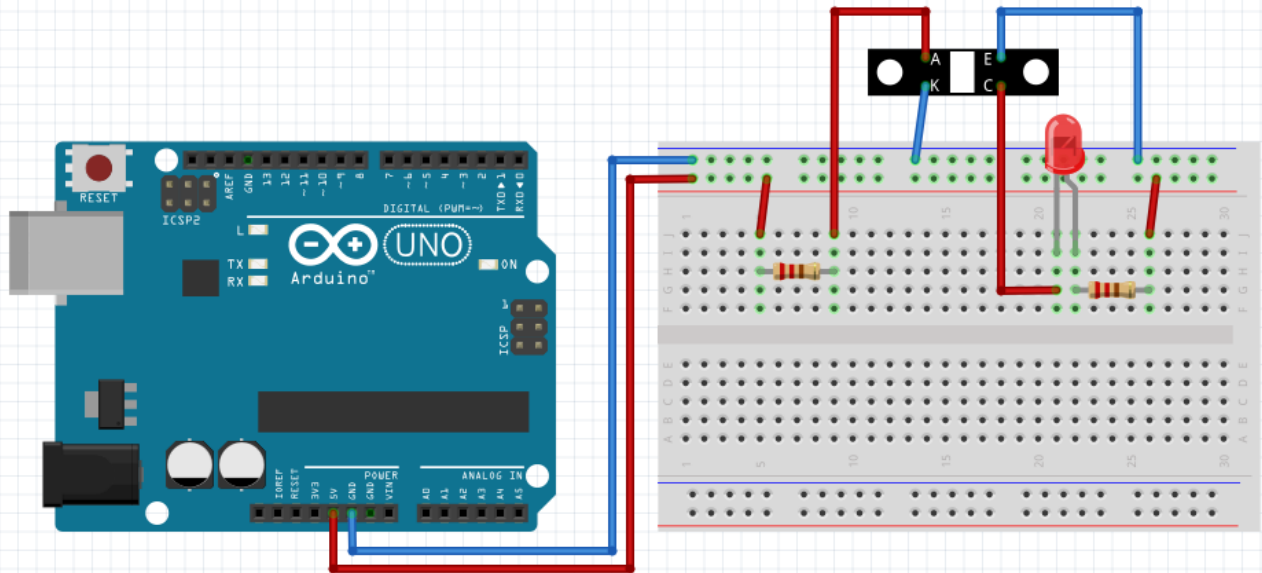


Photo Interrupter wiring diagram

- Place your student card or something similar in the photo interrupter slot.
- What happens to the LED?

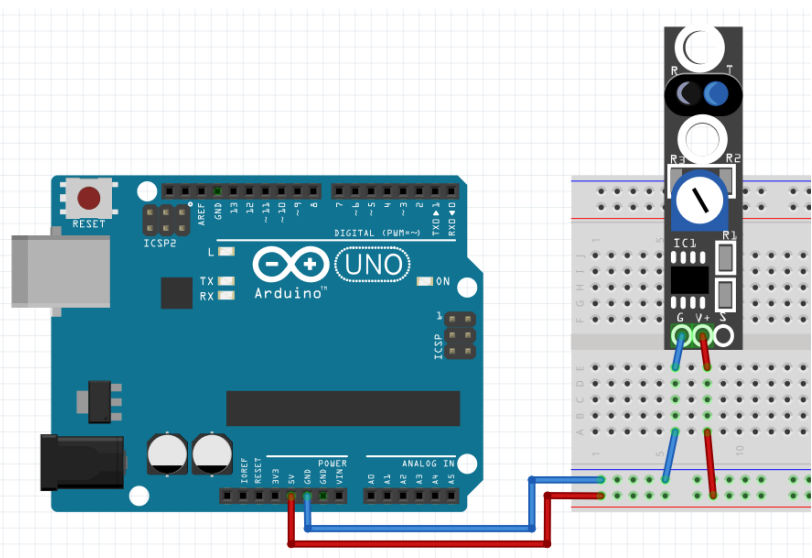
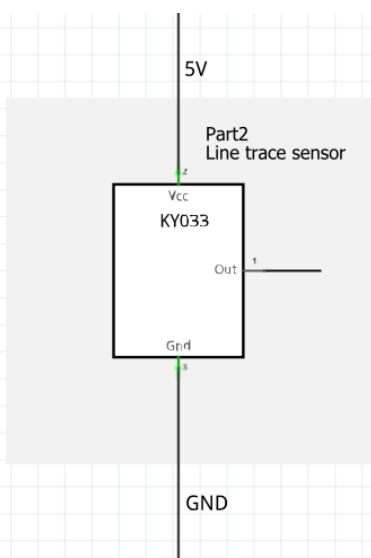
3.2.4 Exercise 4: Line trace sensor

In this exercise you will use a line trace sensor to detect an object or an edge. This is a very useful sensor that can be used to monitor start and finish lines. You'll be using one in your wind powered vehicle project.

- Find the line trace sensor module in your mechatronics kit



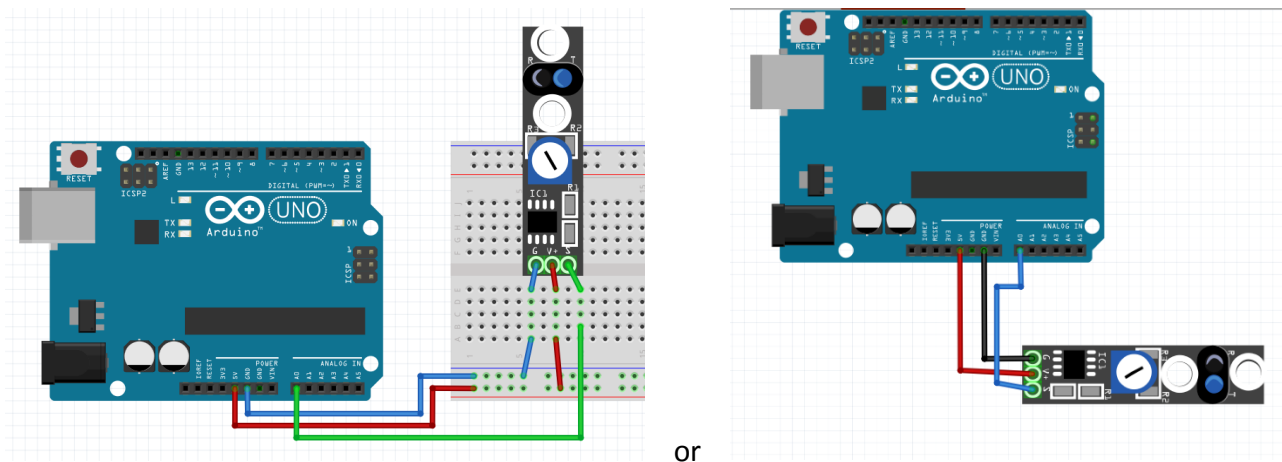
- Create the circuit shown below to power the line trace sensor.
- Place your student card (or something similar) in front of the sensor to observe the sensor's onboard red LED light up
- Turn the potentiometer to adjust the sensor sensitivity. Investigate and observe how this affects how close to the sensor the object is detected. Test different surface finish objects.



3.2.5 Exercise 5: Line trace sensor with serial plotter

In this section we'll write some code to interface with the line trace sensor module and use the Arduino's serial plotter.

- Add a connection from the sensor's signal terminal to the analog pin A0 as shown below.



- Create a new Sketch called LineTraceSerialPlotter and then cut and paste and upload the following code.

```
/*
Line Trace Serial Plotter
UTS 48610 - Introduction to Mechanical and Mechatronic Engineering
Written By Peter de Jersey
Any Questions? Google it before asking your tutor.

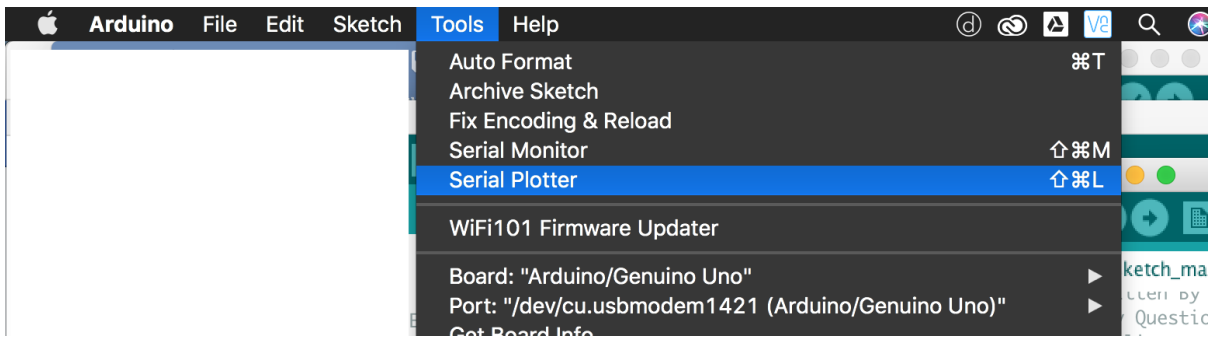
*/
#define Analog_Pin A0

unsigned long oldMillis; // this stores the last value of millis when the Serial
monitor printed the value of A0

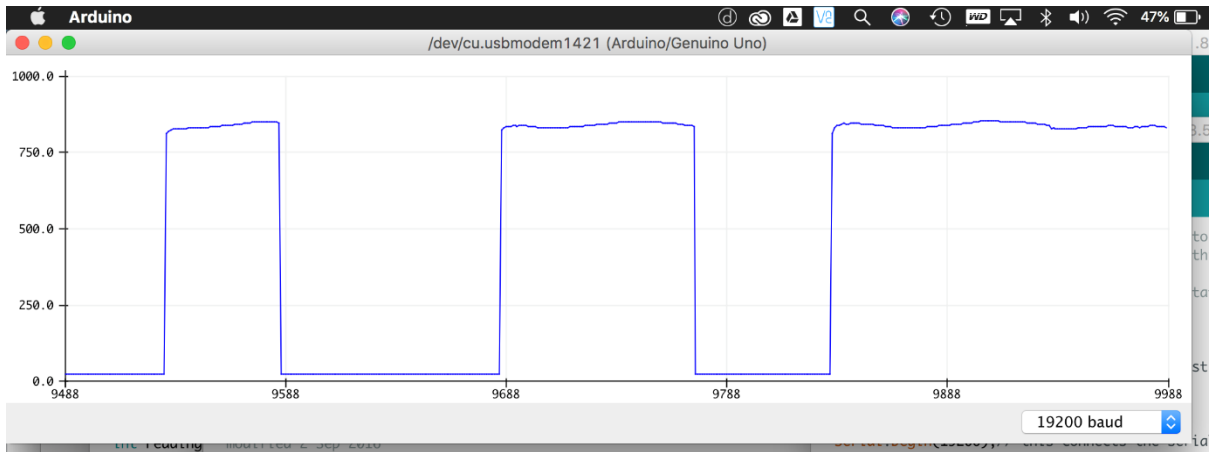
void setup()
{
  Serial.begin(19200); // this connects the serial port
  pinMode(Analog_Pin, INPUT); // this sets the pins mode to an input
}

void loop() {
  int reading = analogRead(Analog_Pin);
  Serial.println(reading);
  delay(20);
}
```

After you've uploaded the code, open the serial plotter. This is an extremely useful tool that allows you to plot information from the serial port. See below:



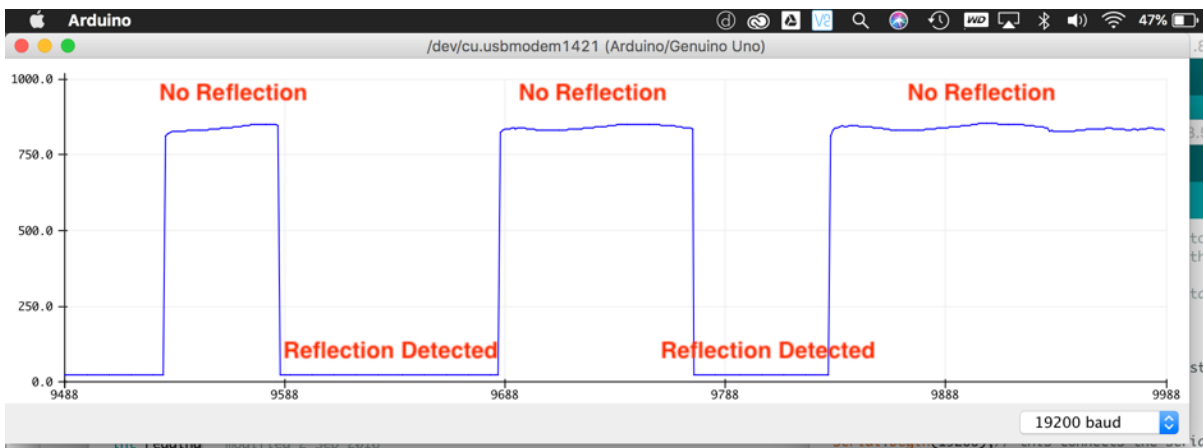
Once you've opened the serial monitor you will see a graph that updates continually. It will look something like this:

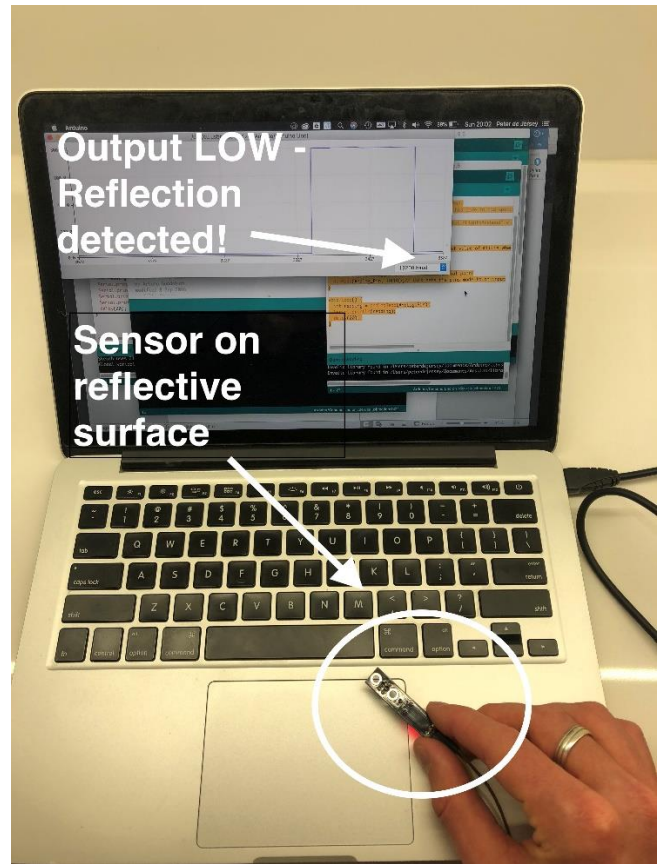
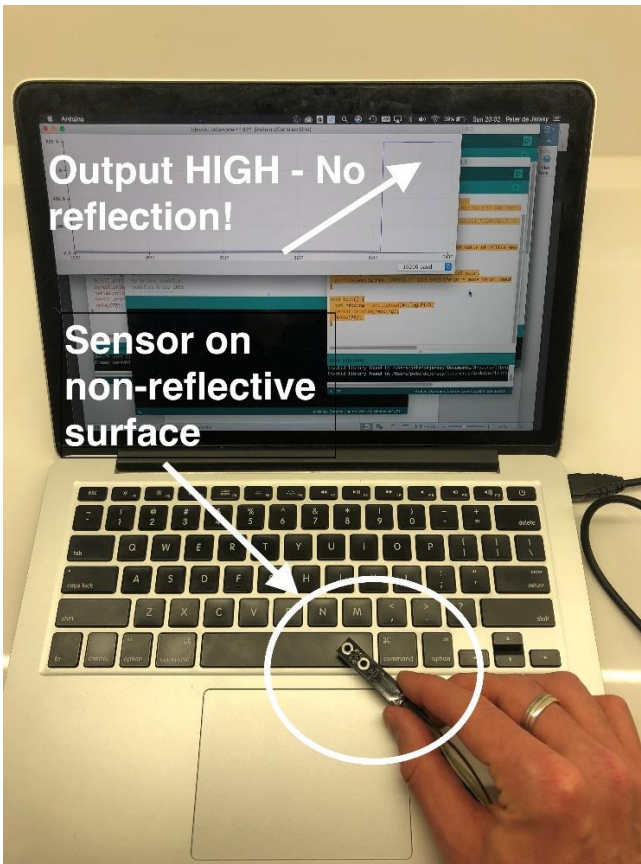


Note that you may need to change the baud rate to match the setting in the code. You may also see a bit more 'noise' on the high signal depending on your computer and other electronic noise sources.

Here you can see the value of the line trace module. When the sensor **does not** detect a reflection the value is **high**. When it **does**, it is **low**. The high value will depend on your sensor's calibration. This can be changed by adjusting the potentiometer on the board with a screw driver.

1. Find a white/light reflective surface (e.g. your student card) and bring the sensor close to it
2. The sensor will detect a reflection. The red LED on the circuit board will also light up.
3. Find a black surface and bring the sensor close to it. The light should turn off. Your computer keyboard can work well if your computer is white and your keys are black. Otherwise think creatively of what else you could use.



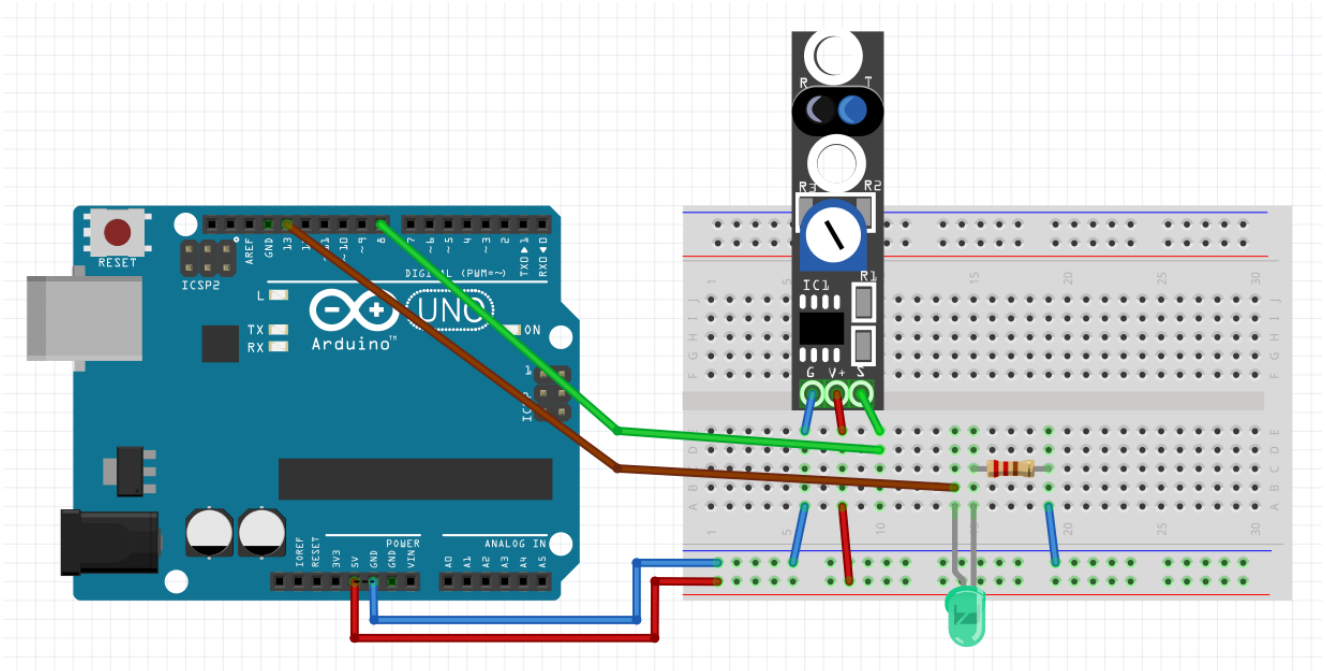


Think about how this might be used to detect the start and stop lines on your WPV track.

3.2.6 Exercise 6: Line trace sensor with digital output to control a LED

In this exercise you will use a line trace sensor to detect and count the number of times an object or an edge is detected and to control a LED.

- We will connect a digital pin (8) to read the line trace sensor's output and use that output to control a LED connected to another digital pin (13).
- Create the circuit shown below.



- Check that the line trace sensor is working by observing the onboard red LED light up when you bring your student card close to the sensor.
- Create a new sketch and copy, paste and upload the following code.

```
/*
Line Trace Detection
UTS 48610 - Introduction to Mechanical and Mechatronic Engineering
Written By Terry Brown
```

```
Modified from StateChangeDetection
created 27 Sep 2005
modified 30 Aug 2011
by Tom Igoe
```

State change detection (edge detection)

Often, you don't need to know the state of a digital input all the time, but you just need to know when the input changes from one state to another. For example, you want to know when a button/sensor goes from OFF to ON. This is called state change detection, or edge detection.

This example shows how to detect when a button or sensor changes from off to on and on to off.

The circuit:
- see 48610 IMME Mx tutorial
*/

```
// these constants won't change:
const int lineTraceInput = 8; // the pin that the line trace sensor is attached to
const int ledPin = 13; // the pin that the LED is attached to

// Variables will change:
int lineTraceCounter = 0; // counter for the number of line trace sensor changes
int lineTraceState = 0; // current state of the line trace sensor
int lastlineTraceState = 0; // previous state of the line trace sensor

void setup() {
// initialize the line trace sensor pin as an input:
pinMode(lineTraceInput, INPUT);
// initialize the LED as an output:
pinMode(ledPin, OUTPUT);
// initialize serial communication and set the baud rate:
Serial.begin(9600);
}

void loop() {
// read the line trace input pin:
lineTraceState = digitalRead(lineTraceInput);

// compare the lineTraceState to its previous state
if (lineTraceState != lastlineTraceState) {
// if the state has changed, increment the counter
if (lineTraceState == HIGH) {
// if the current state is HIGH then the line sensor went from off to on:
lineTraceCounter++;
Serial.println("on");
Serial.print("number of line trace sensor hits: ");
```

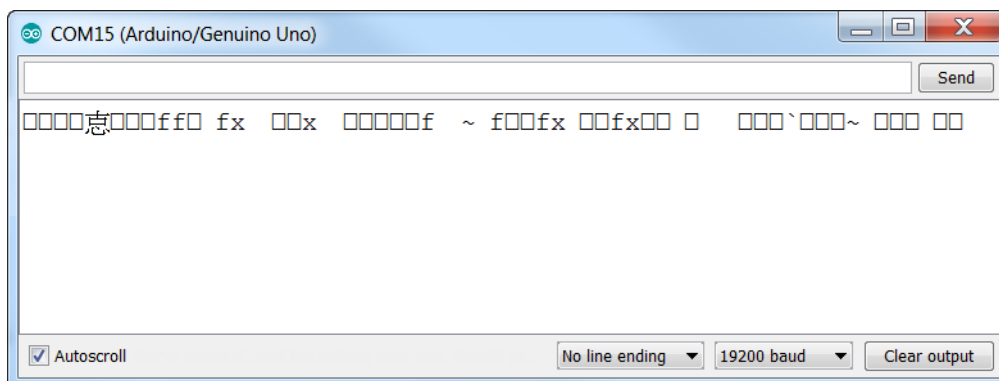
```

Serial.println(lineTraceCounter);
} else {
  // if the current state is LOW then the line sensor went from on to off:
  Serial.println("off");
}
// Delay a little bit to avoid bouncing
delay(50);
}
// save the current state as the last state, for next time through the loop
lastlineTraceState = lineTraceState;

// turns on the LED every four line sensor 'hits' by checking the modulo of the
// line sensor hit counter. the modulo function gives you the remainder of the
// division of two numbers:
if (lineTraceCounter % 4 == 0) {
  digitalWrite(ledPin, HIGH);
} else {
  digitalWrite(ledPin, LOW);
}
}
}

```

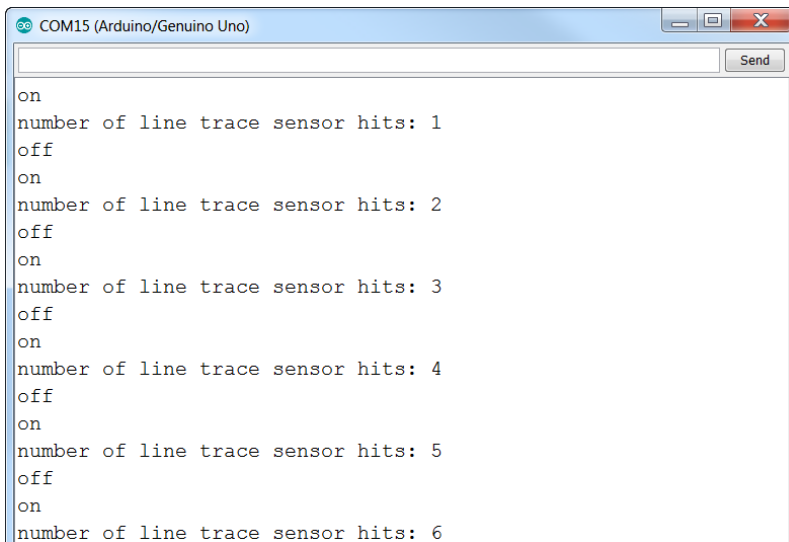
- Check that the line trace sensor is working by observing the onboard red LED light up when you bring your student card close to the sensor.
- Upload the following code.
- Start the serial monitor. If you see something like the following you will need to change the baud rate to that specified in the code. Find where the baud rate is set in the code.



- Once you set the correct baud rate you should see the following (if the line trace sensor is not detecting anything)



- Now use your card to 'trigger' the sensor several times.
- You should see something like the following
- You will also see the LED connected to pin 13 turn on and off intermittently. See if you can interpret the code to understand what is happening.



```
on
number of line trace sensor hits: 1
off
on
number of line trace sensor hits: 2
off
on
number of line trace sensor hits: 3
off
on
number of line trace sensor hits: 4
off
on
number of line trace sensor hits: 5
off
on
number of line trace sensor hits: 6
```

- When writing this code I modified open source code based on using a push button, hence “on” and “off” as output. See if you can adjust the code so that it outputs “detected” and “not detected” instead.

3.2.7 Exercise 7: WPV desktop prototype

You have now used all of the components that you will use for your data acquisition module for your wind powered vehicle. It is good practice to create desktop prototypes of electronic control modules before installing them in a device. In practice we would also create printed circuit board (PCB) from our schematic rather than using a breadboard in an actual mechatronic device.

- Download from UTSONline the WPV Mx module schematics and wiring diagram.
- Attempt to create a desktop prototype of the WPV Mx module
- Test your prototype to see if it works correctly

Arduino coding cheat sheet

(there is a better resolution version of this on UTSONline)

ARDUINO CHEAT SHEET

For more information visit: <http://arduino.cc/en/Reference/>



```

Structure
/* Each Arduino sketch must contain the
following two functions. */
void setup()
{
  /* this code runs once at the beginning of
the code execution. */
}

void loop()
{
  /* this code runs repeatedly over and over
as long as the board is powered. */
}
    
```

```

Comments
// this is a single line
/* this is
a multiline */

Setup
pinMode(pin, [INPUT\OUTPUT\INPUT_PUL-
LUP]);
/* Sets the mode of the digital I/O pin.
It can be set as an input, output, or an
input with an internal pull-up resistor.
*/
    
```

```

Control Structures
if(condition)
{
  /* if condition is TRUE, do something here
}
else
{
  /* otherwise, do this
}

for(initialization; condition; increment)
{
  /* do this
}
/* The 'for' statement is used to repeat
a block of statements enclosed in curly
braces. An increment counter is usually
used to increment and terminate the loop.
*/
    
```

```

    parenthesis
    declares variable (optional)
    initialize test increment or
    decrement
    for (int x = 0; x < 100; x++) {
      println(x); // prints 0 to 99
    }
    
```

```

Digital I/O
digitalWrite(pin, val);
/* val = HIGH or LOW write a HIGH or a LOW
value to a digital pin. */
int var = digitalRead(pin);
/* Reads the value from a specified digital
pin, either HIGH or LOW. */

Analog I/O
analogWrite(pin, val);
/* Writes an analog value to a pin.
val = integer value from 0 to 255 */
int var = analogRead(pin);
/* Reads the value from the specified
analog pin. */
    
```

```

Advanced I/O
tone(pin, freq);
/* Generates a square wave of the specified
frequency to a pin. Pin must be one of the
PWM (-) pins. */
tone(pin, freq, duration);
/* Generates a square wave of the specified
frequency to a pin for a duration in
milliseconds. Pin must be one of the PWM (-)
pins. */
noTone(pin);
// Turns off the tone on the pin.
    
```

```

Time
delay(time_ms);
/* Pauses the program for the amount of time
(in milliseconds). */
delayMicroseconds(time_us);
/* Pauses the program for the amount of time
(in microseconds). */
millis();
/* Returns the number of milliseconds since
the board began running the current program.
max: 4,294,967,295 */
micros();
/* Returns the number of microseconds since
the board began running the current program.
max: 4,294,967,295 */
    
```

```

Data Types
void // nothing is returned
boolean // 0, 1, false, true
char // 8 bits: ASCII character
byte // 8 bits: 0 to 255, unsigned
int // 16 bits: 32,768 to 32,767, signed
long // 32 bits: 2,147,483,648
to 2,147,483,647, signed */
float // 32 bits, signed decimal

Constants
HIGH\LOW
INPUT\OUTPUT
true\false
    
```

```

Mathematical Operators
= // assignment
+ // addition
- // subtraction
* // multiplication
/ // division
% // modulus
    
```

```

Logical Operators
== // boolean equal to
!= // not equal to
< // less than
> // greater than
<= // less than or equal to
>= // greater than or equal to
&& // Boolean AND
|| // Boolean OR
! // Boolean NOT
    
```

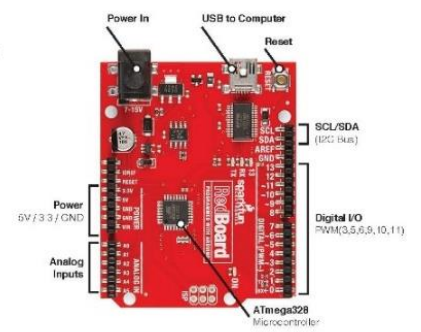
```

Bitwise Operators
& // bitwise AND
| // bitwise OR
^ // bitwise XOR
~ // bitwise INVERT
var << n // bitwise shift left by n bits
var >> n // bitwise shift right by n bits
    
```

```

Libraries
#include <libraryname.h>
/* this provides access to special
additional functions for things such as
servo motors, SD card, wifi, or bluetooth.
*/
    
```

Red Board:



LilyPad ProtoSnap Simple:

