School of Engineering, Discipline of Mechanics and Mechatronics, pp. 1–14

Lab 1: Introduction to Arduino and Digital I/O

ENGG1500
Semester 1 2019

Nicholas O'Dell, Brenton Schulz, Dylan Cuskelly.

---

1 Introduction

In this lab you will be introduced to some of the components you will use in this project. An Arduino Uno development board will be the core of your project and you will become familiar with development on this microcontroller platform.

The Arduino is a small self contained computer which can be programmed to process and output data. The operations the Arduino performs are determined by code that you write and program onto it. The Arduino programming language is based on C, which is formally covered in ENGG1003. The Arduino software comes with many pre-written modules of code called *functions*. These functions can perform complicated tasks while being relatively simple to use.

By the end of this lab you will learn the basics of digital I/O programming on the Arduino platform. There will be many programming, and some hardware, concepts which will be introduced very rapidly. You may not fully understand all of them at first however these topics will be developed further in later weeks.

You will be provided with an Arduino Uno for practice (and play!) outside of lab hours. This is yours to keep, don't be afraid to experiment! It is also intended for you to use to build any additional electro-mechanical obstacles that are part of the track (the gate).

In this lab there are approximately 2 PCs per team. We suggest that you use this time and your own Arduino (that you will be provided with) as well as the one included in the kit so that as many team members as possible can gain some programming experience in this first lab.

There are hundreds of hardware components designed to work with Arduino development boards available. Browsing online stores such as eBay or AliExpress with an "Arduino" search keyword will provide you with an idea of what is available.

At the end of each lab you will find sections entitled **Required tasks for next week** and **Recommendations**. Within the first you will find tasks that must be completed before each successive lab or you will be considered behind schedule. Within the second you will find highly recommended tasks for you to compete outside the lab. Completing the listed tasks before each successive lab will allow you to more usefully use the time in the lab.

2 Your Kit

Warning

Do not assemble your robot until you have finished the lab!

You should have approximately 2 computers per team in EE103A. Use both of these computers to complete the lab in pairs.

Each group has been supplied with a kit. Inside this kit are the parts you will need to make a robot for your project. You are free to purchase other parts if you wish to experiment but this is not required to achieve a high mark.

Parts included in your kit box:

Hardware

- ☐ 1 × small storage box
- ☐ 1 × Acrylic chassis
- ☐ 4 × Acrylic T-pieces
- ☐ 1 × MDF sonar adapter
- ☐ 1 × MDF Board mount
- ☐ 2 × Wheels
- ☐ 2 × Slotted wheels (encoder disc)
- ☐ 1 × Rotor castor wheel
- ☐ 1 × Rotating servo mount
- ☐ 3D printed sonar adapter
- ☐ 2 × No.1 phillips screwdriver
- ☐ 2 × Velcro dots
- ☐ 1 × Rubber Duck

Fasteners

- ☐ 4 × 12 mm standoffs
- ☐ 4 × 30 mm stand offs
- ☐ 5 × 25 mm stand offs
- ☐ 12 × 10 mm standoffs
- ☐ 16 × M3 x 6 mm Screws
- ☐ 20 × M3 x 10 mm Screws
- ☐ 10 × M3 Nuts
- ☐ Cable ties

Electronics

- ☐ 1 × Arduino Uno R3 with cable
- ☐ 1 × Sensor shield
- ☐ 1 × Band of jumper wires (200 mm)
- ☐ Assorted jumper wires (100 mm)
- ☐ 1 × Long USB cable (for Arduino)
- ☐ 1 × 3 pin JST connector
- ☐ 1 × 2 pin JST connector
- ☐ 1 × 0.96" OLED screen

Sensors

- ☐ 2 × HC-SR04 Ultrasonic distance sensor
- ☐ 5 × IR line sensors
- ☐ 2 × IR obstacle sensors
- ☐ 1 × Encoder PCB
- ☐ 1 × I2C RGB colour and gesture sensor

Power Electronics

- ☐ 1 × L298N motor driver
- ☐ 1 × Power distribution board
- ☐ 1 × Battery and charger

Actuators

- ☐ 2 × DC motors with gearbox
- ☐ 2 × SG90 micro-servo

The parts will be progressively introduced in tutorials as required. This tutorial will only require the Arduino board, Sensor shield, an IR line sensor and some jumper wires. You are of-course welcome and encouraged to experiment with the included sensors at home, however, please refer to the project reference manual (available on Blackboard) before applying power to a sensor to ensure it is connected correctly[1].

[1] There may not be any spares.

3 Plugging in

Figure 1 shows the Arduino board that will be used in this project: the *Arduino Uno R3* development board.

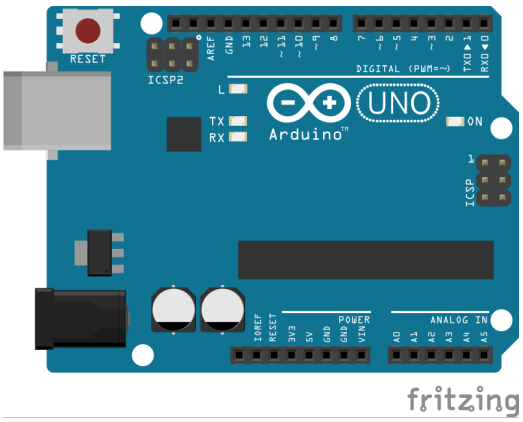

Figure 1: Arduino Uno R3

The Arduino is programmed through the USB port of the computer using the included USB cable. To program the Arduino:

1) Log into the lab PC using your student number and password. This is the same login as Uonline.
2) Plug the Arduino into the computer using the blue USB cable.
3) Wait until the driver has installed, then unplug the Arduino. (This should take about a minute and is necessary because of the university's IT policy).
4) Open **Arduino** by selecting **Start** → **All Apps** → **Arduino**. If you wish to install the software on your computer at home, download the Arduino IDE software here: `https://www.Arduino.cc/en/main/software`. There is also a newly released **Arduino Web Editor** which allows Arduino development without installing the IDE.
5) Select **Tools** → **Port** and note the number of each port that appears.
6) Plug in the Arduino, go back to **Tools** → **Port** and connect to the new port which should appear when the Arduino is plugged in the second time. It may take up to 20 seconds or so to appear after plugging in the Arduino.
7) Select **Tools** → **Port** the default is probably COM1, select the COM port that just appeared.
8) It is important to select the correct board for the compiler. This provides information such as available program memory, pin configuration etc. To do this select **Tools** → **Board** → **Arduino/Genuino Uno**.

To make a new project at any time: Open **Arduino (1.6.12)**, create a new project by selecting **File** → **New** (or press **Ctrl** + **N**), save this project into a folder on your U: drive. The U: drive is a personal space on the university server which you can access from any university computer [2]. Do

[2] A little known feature of the university's IT system is that you can access your U: drive files using *ssh* (secure shell) to the address jumpgate.newcastle.edu.au . Under Linux the *scp* command can be used, a Windows port called pscp

Listing 1: Blink.ino

```
/*
  Blink.ino - Blinks the onboard LED at 0.5Hz (once every 2 seconds).
*/

// The setup function runs once when you press reset or power on the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop(){
  digitalWrite(13, HIGH);       // turn the LED (pin 13) on, HIGH is +5V
  delay(1000);                  // wait for a second (1000 ms)
  digitalWrite(13, LOW);        // turn the LED (pin 13) off by making the voltage 0V
  delay(1000);                  // wait for a second
}
```

not try to save anything to the C: drive on a university computer as it won't be available on other machines and is periodically deleted.

If you purchase an authentic Arduino and use it on your personal computer many of the above steps will happen automatically however if you ever have trouble these steps should always work. Setting up new hardware or software often requires multiple attempts, do not let it discourage you!

4 Digital output, LED

In this section we will write some code which will control a digital output on the Arduino. On the long sides of the board you will see a number of pins labeled 1, 2, ... , 13 (see Figure 1) which are the digital input-output (I/O) pins. When used as outputs you can think of these as on/off switches as these pins will produce either +5V (logic `HIGH`) or 0V (logic `LOW`).

The Arduino has a built in light emitting diode (LED), connected to digital pin 13, which will be switched on and off by code you write. Note that the Arduino boards are programmed at the factory with code which causes this red/orangeLED to flash on your board right now.

The following code makes the LED on pin 13 blink. The grey text is commenting and is ignored by the Arduino, it is only there to help you understand the code. Commenting is further explained in Section 5.

1) Type the code shown in Listing 1 into your Arduino sketch. Be careful with capital letters and symbols! (what we call syntax, explained in Section 5)

2) To send this program to the microcontroller, click **Upload** in the top left corner or press **Ctrl + U**. If the program will not upload after about a minute, an incorrect port may be selected.

is available from http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html. Login using your normal UOnline credentials

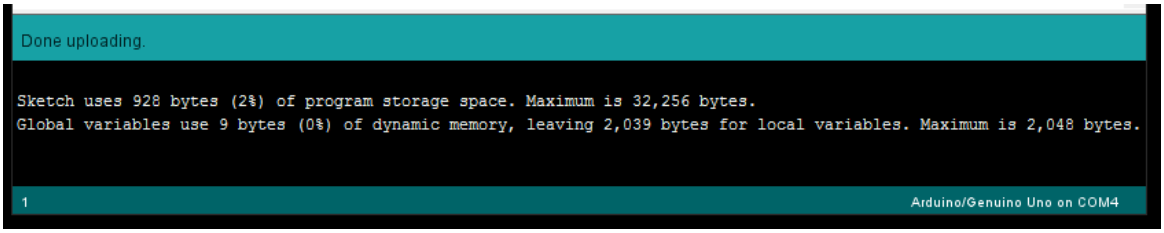

Figure 2: Arduino console showing a successful program upload

Repeat steps 2-6 in the **Plugging in** section to determine which is the correct port. The output console at the bottom of the Arduino IDE will provide technical information. In the event of an error try Googling the error if your lab demonstrator is not available.

3) After your program has uploaded on the board you should see output similar to Figure 2 in the console.

4) Congratulations you just wrote your first bit of code[3]! Now let's change something!

5) Change `delay(1000);` to `delay(100);` and hit **Upload** again.

6) Now that your program has successfully loaded on the board you should see the LED flash on and off rapidly. Experiment with the delays and observe the change in LED behaviour.

Sending the code to the Arduino is called *flashing* the program. This process takes the script (program) written on the screen, turns it into machine code the Arduino understands and *flashes* it into the Arduino's *flash memory*[4]. Code in the Arduino's flash memory will be executed as soon as it is turned on and continue doing so until power is lost. The internal flash memory is known as non-volatile which means that it retains its contents after the Arduino is switched off.

5 Programming

To further your understanding of programming, a brief introduction is presented in this section. This is not intended to fully develop the concepts, just to help you understand the basic principles. Learning a programming language for the first time can be as intimidating as learning a new spoken language, but with repeated practice you will pick it up surprisingly quickly.

- **Variables**: Variables are pieces of information. The number of people in this room, for example, could be a variable. Variables are always initialized before they are needed, often at the start of the program. Once we have a variable, it can be used in an operation (arithmetic or program flow control) to help us achieve a required output. For example, say code is required to calculate the number of semi-autonomous vehicle kits required for a lab group. We would first need to know how many people are in the room so we would *declare* (create) a variable:

```
int peopleInRoom = 32;
```

Here, `peopleInRoom` is a variable of integer type. The variable is assigned the value 32. We would require a variable for the number of people sharing each kit:

[3] https://pics.me.me/just-finished-my-first-arduino-project-a-bl-inking-led-21796485.png
[4] The flash memory inside the Arduino is very similar to that in a USB flash drive or SATA SSD

```
int peoplePerKit = 4;
```

Here, `peoplePerKit` is another variable of integer type and is assigned the value 4. Now we can do some arithmetic to calculate how many kits are required. The whole sequence is written in code as follows:

```
int peopleInRoom = 32;
int peoplePerKit = 4;
int kitsNeeded = peopleInRoom/peoplePerKit;
```

We just declared `kitsNeeded`, a variable of integer type and it holds the solution of the calculation `peopleInRoom / peoplePerKit` (4).

- **Functions:** A function is a segment of code that performs a predefined task. Typically functions perform a task which is required multiple times so that it only needs to be written once and *called* where required. In the previous example program, `delay(ms)` and `digitalWrite(pin, state)` were functions that were each called twice.
- **Arguments**: You will notice that both the functions `delay` and `digitalWrite` had a value in brackets (). Almost all functions need an input, called arguments. In the case of `delay` it is the time of, the delay in milliseconds, and in `digitalWrite` it is the pin we wish to change the output state of along with what that state will be.
  The Arduino library has many prewritten functions to help you write programs quickly and efficiently. A comprehensive guide of the Arduino functions and syntax, the Arduino Language Reference page outlining the use of the Arduino specific functions, can be found here: https://www.Arduino.cc/en/Reference/HomePage. Have a brief read. At this stage the reference page may not seem particularly useful, however once you have some more experience, you will want to use it to help you program your Arduino to do tasks not specifically covered by these laboratories. You will be regularly referred to the Arduino page to make you familiar with their format.
- **Syntax**: Just as a written language has grammatical rules, a coding language has specific rules known as syntax. Some immediately useful syntax is listed below.

  - Most lines of code end with a semicolon (;). This marks the end of a line. Forgetting this will result in a compilation error. It is similar to a full stop ending a sentence. The lines which do not end with a semicolon are the start of functions and loops along with `if()` and `switch()`[5] control blocks. It will be confusing at first but should become obvious with practice.
  - Curly brackets { } enclose a function.
  - Normal brackets () enclose the arguments passed to a function.
  - Anything that appears in grey below is a comment, and is not part of the program. Comments are essential both for you to understand how your code works and for others to understand your code. Anything in a line after and including "//" is not part of the program. E.g., //The list you are currently reading helps explain syntax.. As you attempt

[5] `switch()` statements are not explicitly taught in ENGG1500 but you are welcome to learn about and use them.

to learn a new programming language you should note the extreme importance of well commented code.

- ORANGE words are function names.
- BLUE words are predefined variables. E.g. In the last section LOW and HIGH are defined as 0 and 1 respectively. If HIGH was replaced with 1 and LOW were replaced with 0, nothing would change. Another predefined value in Arduino is PI.
- C code is case sensitive. E.g., `peoplePerKit` is not the same as `peopleperkit`. Be very careful.

6 Digital input
Sensing the world

Your Arduino board can be connected to a wide range of sensors to measure the world around it and objects to manipulate the world (like motors and LEDs). In general to interface with a component you need to power it and connect it to an output pin to tell it what to do. While large devices like motors need an external power source smaller devices like LEDs and many sensors can be powered by the Arduino's power source through the Arduino itself.

In this lab we will begin to use the 'sensor shield'. It is a circuit board which plugs on top of your Arduino and is shown in Figure 3. The Arduino Uno only contains one 5 V pin and three ground pins. This is a limiting factor when wishing to power many devices, as each plugged in module requires both +5 V and ground. The sensor shield contains a +5 V and ground pin for every digital and analogue I/O pin available on the Arduino, allowing for many sensor modules to connect without having to share pins. Carefully connect it to the Arduino as shown in Figure 4.

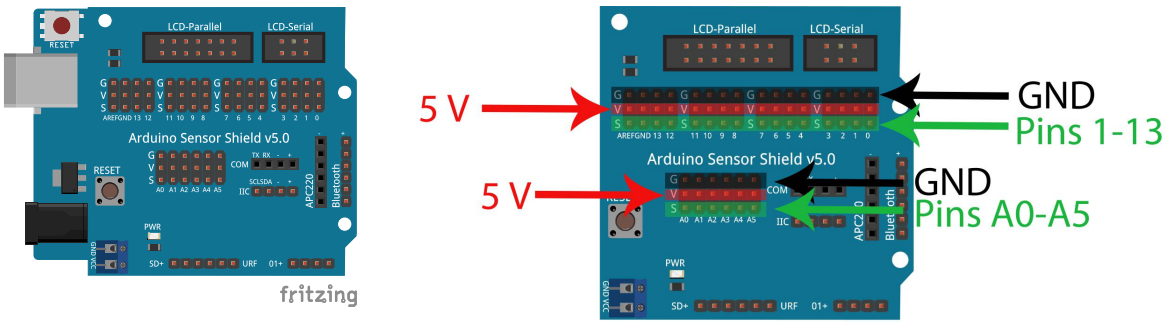

Figure 3: Arduino sensor shield mounted on the Arduino UNO R3

Task: In this section you will use a reflective infra-red (IR) sensor to allow the Arduino to tell the difference between a black or white surface. Some of the code is written for you but you will need to fill in some blanks to make the code work. A reflective IR sensor contains an emitter and a receiver. The receiver varies its voltage depending on the amount of detected IR. That is, the more IR it receives from a reflected surface, the higher voltage it produces. Once the voltage has increased over a threshold voltage, it registers as `HIGH` — otherwise it reads `LOW`. So when the sensor sees a reflective (e.g. white surface) it will be on and when it is looking at a dark surface (or nothing) it will be off. The sensor contains a potentiometer (variable resistor) to tune the sensitivity. There are two types of

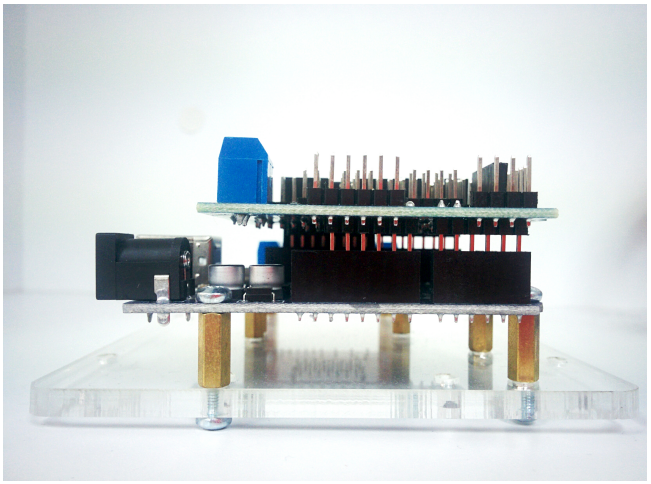

Figure 4: How to connect the Sensor Shield or Breadboard Shield to the Arduino.

IR sensors supplied in your kit: one points down, to detect lines; and the other points horizontally to detect obstacles.

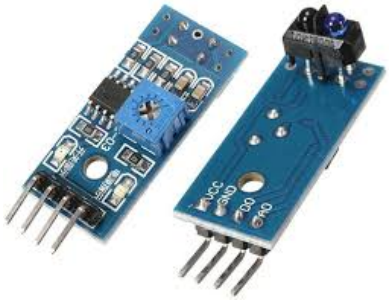

Figure 5: IR reflective sensor

1) Connect the reflective sensor (Figure 5) to the Arduino as shown in Figure 6 and Table 1. Connections are shown for both the SensorShield, included in your kit, and for the *ProtoShield* provided with your individual Arduinos.

   i) Connect **VCC** pin of the line sensor to the **5 V** pin of the Arduino and connect the **GND** pin of the sensor to the **GND** pin of the Arduino. This provides power to the sensor.

> **Warning**
>
> If connected incorrectly your sensor will get very hot and then cease to function.

   ii) Connect the digital output (**DO**) pin of the sensor to pin **A0** of the Arduino.

2) Create a new project beginning with the following code, which is contained in the `lab1_files.zip` on Blackboard. Save it to your `U:` drive or a personal USB drive with an appropriate name.

Table 1: IR sensor pin

Sensor	Arduino
VCC	5V / V
GND	GND / G
DO	A0

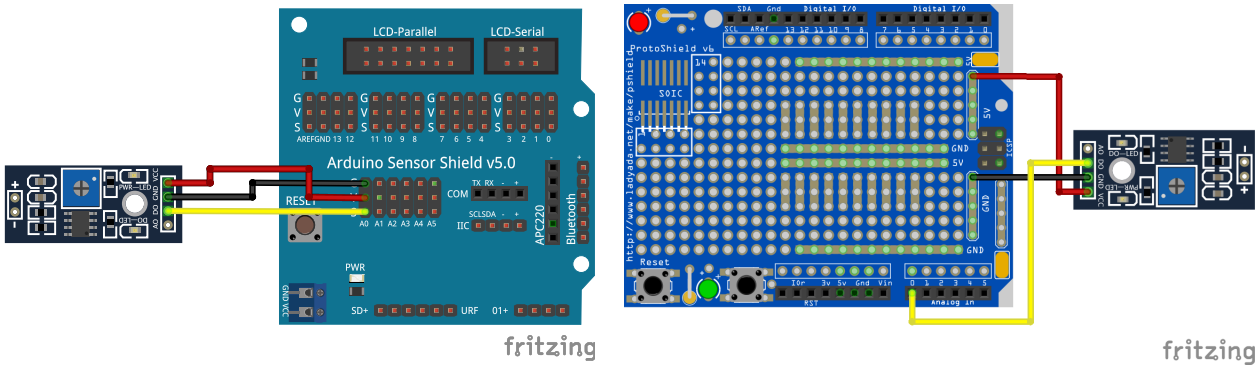

Figure 6: IR reflective sensor circuit for SensorSheild (left) and ProtoShield (right). Connect **VCC** to **V**, **GND** to **G**, and **D0** to **A0**. The order of the pins will vary from sensor to sensor, check the Project Reference Manual on Blackboard and connect each sensor according to its instructions.

Listing 2: DigitalInputIR.ino

```
//Digital Input IR
void setup() {
  // put your setup code here, to run once:
  pinMode(A0,INPUT); //Initialising pin A0 as an input
  Serial.begin(9600);//Setting up serial communication to view results,
  //9600 is a common and default baud rate,
  //Ctrl+Shift+M to open the serial monitor.
}

//Declaring a variable "sensorReading" that we can use later,
//At the same time we declare it we set it equal to 0,
//It is declared here so that it can be used inside functions such as void loop()


int sensorReading = 0;
void loop() {
  // put your main code here, to run repeatedly:

  //TODO: Insert 100ms delay here.

  //TODO: Read the sensor into "sensorReading" here.

  //Print the value read to the serial monitor.
  Serial.println(sensorReading);//Note that an l(L),1,I all look similar in code
       //...font
```

```
  //Extended: Set the LED on the board to the value read from the sensor
  //using digitalWrite.
}
```

3) Use the function `delay()`, the same function as the last task to insert a 100 ms delay. Remember to put a semicolon (;) at the end of every line. This tells the compiler that that is the end of the line. i.e. `delay(100);`

4) Use the Arduino function `digitalRead()` to read the value of pin `A0` (or whichever pin the sensor is attached to) and store it in the previously declared variable `sensor`.

> **Hint**
>
> Read the reference documentation for the function `digitalRead` (https://www.Arduino.cc/en/Reference/DigitalRead) which contains an example of how to correctly use the function.

The reference page contains the following example:

```
int val = digitalRead(inPin);   // read the input pin
```

The variable declared here is of integer type and stores the value read from the pin (1 or 0). To read the value of pin A0 into the variable `sensor` we would call:

```
sensor = digitalRead(A0);
```

`digitalRead()` is the first function you've used which creates a *return value*. Functions which return values can be called anywhere on the right hand side of an `=` symbol, even if it is part of arithmetic, for example:

```
value = 2 * digitalRead(4) + digitalRead(3);
```

is completely fine. After execution of this line, `value` will be hold the number 0, 1, 2 or 3 as `digitalRead()` returns either a 0 or 1.

When programming don't think of "=" as an equals sign. It is the *assignment operator*. The variable to the left of the = is *assigned* the value resulting from evaluating the code to its right.

5) Verify your program by clicking **Verify** or pressing **Ctrl+R**.

6) Open the serial monitor **Ctrl+Shift+M** and check that the baud rate is set to 9600. Baud rate is the speed at which the Arduino communicates with the computer.

> **Hint**
>
> Use the *serial plotter* to visualise the data instead of the *serial monitor*. Open the serial plotter by selecting **Tools→Serial Plotter**, or pressing **Ctrl+Shift+L**.

7) Upload your program to the Arduino.

8) Every 100 ms the value being read from the pin should be printing to the serial monitor. You can adjust the sensitivity by turning the potentiometer[6] with the supplied screwdriver. This will allow you to distinguish between a black surface and a white surface at various distances.
Now test this on one of the track segments available in the lab.

9) Modify your code so that the value read from `digitalRead()` is written to the LED connected to digital pin 13.

   i) Initialise the LED as an output in the setup section:

```
pinMode(13,OUTPUT);
```

   ii) Use the value we just stored in **sensor** as an input to turn the LED off or on after reading the sensor in the loop.

```
digitalWrite(13,sensor);
```

   This will take the value stored in sensor, 0 or 1, and write it to the LED.

10) Modify your code to remove the need for the `sensor` variable. You can do this by inserting the call to `digitalRead()` into the argument of `digitalWrite()`.

11) Once finished there is no need to *safely eject* the microcontroller, just disconnect the USB cable.

Warning
Do not unplug the Arduino if it is in the process of having code flashed.

7 Required tasks for next week

- Enclosed footwear is required in the EE labs! It is mandatory that you complete the EE safety induction titled Lab Induction for EE & ES Buildings within the **SEEC LAB INDUCTIONS & ACCESS QUIZZES (CALLAGHAN 2019)** 'course site' on Blackboard[7] as soon as possible, if you do not complete this before next week's scheduled lab you will not be allowed to enter the lab.
- Assemble your robot! You will need the motors, motor driver, and wheels mounted to your chassis (as shown in the assembly video) for the next lab.

[6] A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.
[7] listed under **my other course sites**

Warning

Refrain from picking your robot up by the front[a]. This is the most fragile part of your robot and caused 100% of the chassis failures in 2018.

[a] https://www.youtube.com/watch?v=3m5qxZm_JqM

- Communication with team group is key! Start a Facebook chat or Google Hangout.

8 Recommendations

Here are some suggestions on what you should work on next:

- The various sections of the vehicle track are available with the project documentation on Blackboard. In order for your group to work on the project outside of labs you will need a copy of this document. The track is available as a Course Reader from the UoN print shop on 150 gsm[8] paper. Each group will receive *two copies* of the track for $26.00, provided in a document wallet. Due to the accumulated wear and tear while testing on the track it is highly recommended to reserve one copy as a 'clean' track.
- Tools needed to construct the robot are available in the labs and in your kit, however as you are here in order to become an Electrical, Mechatronics or Computer Engineer we suggest you obtain (at lease some of) the following tools for use throughout this project, your degree, and your career.

Tip

You may be hesitant to spend extravagant amounts of money on good quality tools, but also hesitant to buy the cheapest available tool for fear that its quality is so bad you shouldn't have bought it at all. Consider the following philosophy when beginning to buy your own tools:
If you buy the cheapest tool available and it eventually breaks, this means that you have used it enough to justify getting a better quality one. However if it never breaks because it is seldom used, you have just saved money to put towards the tool that you use frequently.

  - Side cutters
    http://www.kmart.com.au/product/side-cutting-pliers—20cm/747853
    http://www.supercheapauto.com.au/Product/ToolPro-Diagonal-Cutters ...
  - Needle nose pliers
    http://www.supercheapauto.com.au/Product/ToolPro-Needle-Nose-Pliers-Mini ...
    https://www.bunnings.com.au/trojan-120mm-mini-long-nose-plier ...
  - Screwdriver set
    https://www.ebay.com.au/itm/Generic-53in1-Precision-Screwdriver-Set ...

[8] suitable for multiple uses

https://www.jaycar.com.au/32-piece-precision-driver-set/p/TD2106

http://www.kmart.com.au/product/23-piece-precision-screwdriver ...

Alternatively Jaycar sells a tool-kit that is of *reasonable* quality for $30
https://www.jaycar.com.au/30-piece-tool-kit-with-case/p/TD2166
This kit is also available for purchase on campus from the post office (inside the Shortland Building near Gloria Jeans).

The electrical engineering tech staff have kindly offered use of their engraving tool to students who wish to engrave their own tools with their name. If you would like to use it go and knock on EEG08 and they will instruct you how to correctly and safely use it.