School of Engineering, Discipline of Mechanics and Mechatronics, pp. 1–9

Introduction to Arduino and Digital I/O

ENGG1500
Semester 1 2017

4th Draft, February 1, 2017

---

1 Introduction

In this lab you will be introduced to some of the components you will use in this project. An Arduino development board will be the core of your project and you will become familiar with the hardware and software. An Arduino is essentially a small self contained processor which you can use to process and output data. How the Arduino reads and processes information is determined by the code that gets programmed onto the Arduino. The Arduino programming language is based on the C programming language that you will learn in ENGG1003. The Arduino software comes with many previously written modules of code called *functions*. These pre-written functions increase the usability of the software so much that by the end of this tutorial you will have written and deployed your first program.

In addition to the Arduino software having many included functions there are many hardware components that are designed to work easily with the Arduino. You will wire up your first sensor in this lab to the Arduino and read data from it.

By the end of this lab you will learn the basics of C programming as it relates to the Arduino and use digital input/output. There will be many programming, and some hardware, concepts which will be introduced very rapidly here. You may not fully understand all of them when you first use them. These topics will be developed in later weeks and your understanding will develop with them.

2 Your Kit

Each group has been supplied with a kit. Inside this kit are the parts you will need to make a robot for your project.

LIST OF PARTS IN KIT

chassis

more stuff

and things

LIST OF PARTS NEEDED FROM OUT THE FRONT

The parts will be progressively introduced in tutorials as required. This tutorial will only require the Arduino board, an IR line sensor and some wires.

3 Plugging in

Figure 1 shows the Arduino board that will be used in this project, the *Arduino Uno R3* development board. Although the Arduino is programmed through the USB port of the computer using the blue cable included.When programming an Arduino:

1) If you have not already logged on to a computer, do so using your student number and password i.e. myhub/blackboard details.

2) Plug the Arduino into the computer using the blue USB cable.

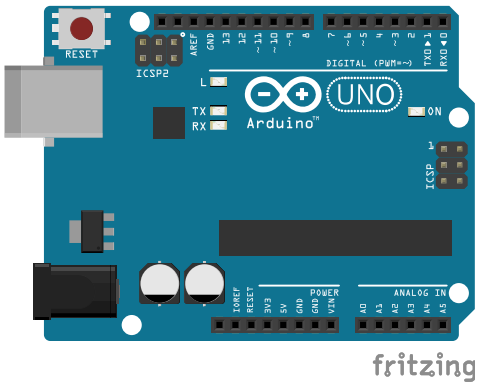

Figure 1: Arduino Uno R3

3) Wait until the driver has installed then unplug the Arduino. (This should take about a minute and is nececary because of the method used to run the Arduino software on the university computers.)

4) Open **Arduino** by selecting **Start** → **All Apps** → **Arduino**, If you wish to install the software on your ocmputer at home download the Arduino IDE software here: https://www.Arduino.cc/en/main/software. Note that Arduino has a newly released **Arduino Web Editor** which is a cloud based version of its software. Make sure the drivers install with the software.

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. Do not try to save anything to the C: drive on a university computer.

4 Digital output, LED

In this section we will write some code which will control an output from the Arduino. On the long sides of the board you will see a number of pins labelled 1-13 (figure 1) which can be connected to something you would like to control (or read).

The simplest way of controlling something is using whats called digital logic, which effectively works like a switch. Digital logic can be HIGH (on) or LOW (off). Arduino logic level is 5 volts, that means that if a pin is set to HIGH (on) it will produce 5 volts, and if it is set to LOW (off) it will produce

0 volts. The Arduino has a built in light emitting diode (LED) wired to digital pin 13, which may be flashing if the Arduino is brand new. In this part of the lab you will use digital output to switch this LED on and off.

The following example code is used to make an LED blink on pin 13. Read through the grey comments which explain what each line does. The following section (section 5) will introduce some basic elements of programming so you may better understand the following code.

Listing 1: LED Blinking

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN (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 the voltage level)
  delay(1000);                  // wait for a second
  digitalWrite(13, LOW);        // turn the LED (pin 13) off by making the voltage LOW
  delay(1000);                  // wait for a second
}
```

1) Select **File** → **Examples** → **01.Basics** → **Blink**. Which is functionally identical to the code above, however the code above is more heavily commented.

2) Change **delay(1000);** to **delay(100**;).

3) To send this program to the microcontroller click **Upload** (or **Ctrl + U**). If the program will not upload after about a minute an incorrect port may be selected. Repeat steps 2-6 in the **Plugging in** section to determine which is the correct port.

4) After your program has uploaded on the board you should see, *Done uploading* your program has been successfully uploaded to the Arduino board. Other information shows how much program memory has been taken up etc.

5) Now that your program has successfully loaded on the board you should see the LED flash on and off rapidly and repeatedly. Change the delays so that the LED flashes faster or slower and upload you code again.

Sending the code to the ardunio is also called flashing the program. This process takes the script (program) written on the screen and uploads/*flashes* Arduino's *flash memory* (i.e. like a flash drive). The code will run on the boards whenever it is turned on. As the code is now on the board it can be unplugged from the computer. The instructions to turn the LED on and off are in a loop so the program will simply keep re-running the instruction over and over until we replace the program.

5 Programming

In order to help you understand the code above a brief introduction to programming will be presented in this section. This is not intended to fully develop the concepts, just to help you understand the code you are using. Learning a programming language for the first time can be 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 to help us get a required output. For example, if I needed to know how many robots kits I needed. I would first need to know how many people are in the room so I would make a variable:

  ```
  int peopleInRoom = 32;
  ```

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

  ```
  int peoplePerKit = 4;
  ```

  Here, `peoplePerKit` is a variable of integer type. The variable is assigned value: 4. Now we can do some maths to figure out how many kits we require, the whole sequence looks like:

  ```
  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 `peopleInRoom / peoplePerKit` (4).

- The most common variable type you will use in this project is an integer (int). An int stores a positive or negative whole number between 32768 (or (215)) and +32767 (or 215 1). It is the most appropriate type to use for most applications in this project. For example storing values read from sensors. See the Arduino reference page https://www.Arduino.cc/en/Reference/Int.

- **Functions:** A function is a segment of code that performs a predefined task. Using our example of determining the number of kits we require, we need to count the number of people in the room. We could first make a generic piece of code which counts, then use that to count the people in the room. I can then call the function in my code by simply writing the name of the function. In the previous example program, delay(ms) and digitalWrite(pin) were functions that just we called.

- **Arguments**: You will notice that both the functions **delay** and **digitalWite** 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 take the information from to write.
  The Arduino library had 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 tutorials. 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.

  - Every line of code ends 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.
  - 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.
  - 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` ≠ `peopleperkit`, be very careful.

6 Digital input Sensing the world

Task: In this section you will use a reflective infra-red (IR) sensor 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 is an IR sensor in your kit.

1) Connect the reflective sensor (figure 3) to the Arduino.
   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.
   ii) Connect the digital output (**DO**) pin of the sensor to pin **8** of the Arduino (arbitrary, could be any digital pin except for 1,2 which is reserved for communication, if you wish to use another pin you have to change the pin used in the code too).
2) Create a new project beginning with the following code, be sure to copy all of it. Save it to your U/USB drive with an appropriate name.

Figure 2: IR reflective sensor

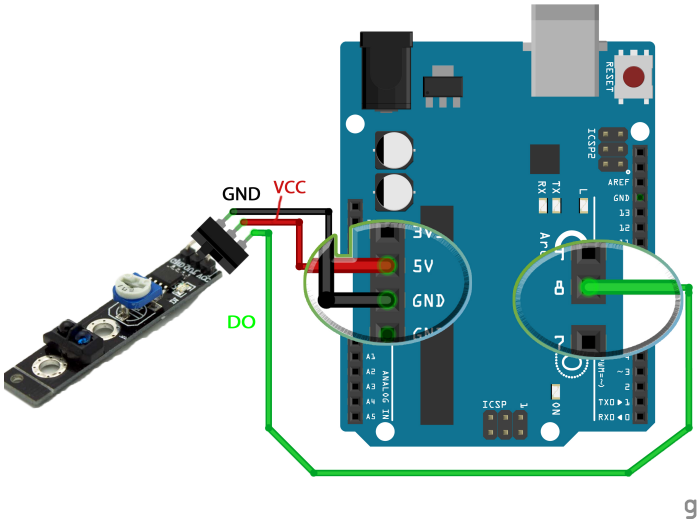

Figure 3: IR reflective sensor circuit

Listing 2: Digital Input - Line Sensor

```
//START
void setup() {
  // put your setup code here, to run once:
  pinMode(8,INPUT);  //Initialising pin 8 as an input
  Serial.begin(9600);//Setting up serial communication to view results,
  //9600 is a common and default baudrate,
  //Ctrl+Shift+M to open the serial monitor.
}

//Declaring a variable "sensor" 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 sensor = 0;

void loop() {
  // put your main code here, to run repeatedly:

  //TODO: Insert 100ms delay here.
  //TODO: Read the sensor into "sensor" here.
```

```
  //Print the value read to the serial monitor.
  Serial.println(sensor);//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.
}
//END
```

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.

4) Use the Arduino function **digitalRead()** to read the value of pin 8 (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).

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

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

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[1]. This can be done with a screwdriver. This will allow you to distinguish between a black surface and a white surface at the various distances.

Figure 4: Conveniently placed black surface

9) Write the value read from the Arduino to the LED using **digitalWrite()**.

   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);
```

[1] A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.

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

10) Once finished there is no need to *safely eject* the microcontroller or anything like that, just pull it out (as long as you are not in the middle of uploading a program to it).