CPSC 1000 - Laboratory manual

Robert Benkoczi

January 1, 2019
version 3.0

Contents

License

Introduction

This manual accompanies the lab activities for course CPSC 1000, Introduction to Computer Science, offered at the University of Lethbridge. In this offering of the course, the students learn to make simple projects using the Arduino micro-controller.

The first part of this manual concentrates on four basic functions of a micro-controller: reading digital and analog signals, and generating digital and analog signals. Then, in Part 2, the students start working on a larger project that involves controlling the movement of a simple three wheel robotic gadget and making it autonomous to some degree.

Students who complete the activities in this manual will have a basic understanding of how to program an Arduino micro-controller and will posses some elemental knowledge concerning simple electronic components such as resistors, light emitting diodes, various sensors, and direct current motors. They will be ready to learn about other, more advanced electronic components and how to build more complex projects.

Suggested lesson plan: This course was offered at the University of Lethbridge during a twelve week semester. Every week, students participated in a two hour lab session and 150 minutes of lectures. For Part I, a format for lectures and lab sessions that seems to have worked well was the following:

- Lectures held during a week introduced the background necessary for the lab activity taking place the following week. The labs started in week two, so there were only 11 weeks of lab activities.
- Lab sessions, the first 45 - 60 minutes of the two hour long session: the lab instructor demonstrated the activity described in this manual and each student, individually, followed the instructions in order to complete the activity.
- Lab sessions, the remaining time: after the activities were completed, the students were assigned one or more exercises from the lab manual and were given marks for completing them.

For Part II of the course, the students were paired in teams and were asked to complete the three challenges described in Part II of this manual. The lectures were moved from the standard classroom into the lab and were converted to lab sessions. Instructors provided guidance and help as needed, but the students were expected to work on their challenges at their own pace.

The twelve weeks of the semester were allocated to the activities as follows.

Week 1: Activity 0 (Chapter 1) and general concepts for lectures, input/output. No labs.
Week 2: Activity 1 (Chapter 2) for lectures, functions and variables. Activity 0 for labs.
Week 3: Activity 2 (Chapter 3) for lectures, expressions. Activity 1 for labs.
Week 4: Activity 3 (Chapter 4) , pull-up/pull-down resistors for lectures. Activity 2 for labs.
Week 5: Activity 3 (c'ed), selection statements for lectures. Activity 3 for labs.
Week 6: Activity 4 (Chapter 5) for lectures, PWM, loops. Activity 3 (c'ed) for labs.
Week 7: Activity 5 (Chapter 6) for lectures. Activity 4 for labs.

Week 8: Software engineering concepts in lectures. Activity 5 for labs.
Weeks 9-12: Lab activities only, students complete the project challenges.

Editing this manual: This manual is written in LaTeX. Figure 8.1 was created using the XFig vector drawing program (see `http://mcj.sourceforge.net/installation.html`), which is available on most linux distributions. The circuit diagrams were created with Autodesk Eagle (see `https://www.autodesk.com/products/eagle/free-download`), for which a free version is available for download.

Code: The source code listings in this manual are extracted from the Arduino source files located in subdirectory *src* in the source directory of this manual. These programs can be compiled and uploaded onto the Arduino board.

Acknowledgements: I have many people to thank for their contribution to making this manual better: my CPSC 1000 class of Spring 2018, who used it in the first offering; my excellent team of graduate student lab instructors: Aryal Chudamani, Tanvir Fuad, Adam Lefaivre, Nicholas Parsons, and Mainul Polash; my graduate student Nicholas Parsons who drew the circuit diagrams, wrote the initial version for most of the Arduino code, and implemented almost all of the activities in this manual to make sure the information provided here is accurate; and last but not least, my colleague Nicole Wilson, for valuable advice and feedback on this material.

Part I

Fundamental concepts

Chapter 1

Activity 0: Arduino and its programming language

1.1 Lab activity objectives

An Arduino micro-controller is essentially a special kind of computer. It is a computer with a very limited "user interface", because a micro-controller is not supposed to be handled directly by humans. It is designed to interact with other electronic components.

In this activity, students will interact with an Arduino micro-controller using the Arduino IDE computer program as an intermediary. Students will upload code onto (or program) an Arduino micro-controller and will learn some basic programming commands that will allow the Arduino to perform operations like any other computer.

1.2 Component list

– 1 x Arduino micro-controller
– 1 x USB cable

1.3 Circuit set-up

For this activity, no electronic circuit is really necessary. Simply connect the USB cable to the Arduino board and then connect the other end of the cable into the USB port of your workstation, thus powering up the Arduino.

The USB cable will be used to provide power to the device and to transmit information back and forth between the micro-controller and the workstation.

1.4 Arduino program

An Arduino program contains two mandatory functions that must be defined by the programmer: (1) Line 1: *setup*, which is executed a single time when the micro-controller is powered up or when the code is uploaded to the Arduino, and (2) Line 6: *loop* which is executed repeatedly, for as long as the Arduino is powered up and no other program is uploaded. The instructions to be executed by *setup* and *loop* must be inserted in the body of these functions, namely between { and }.

Listing 1.1: Arduino code for Activity 0: Output "Hello world"

```
1 void setup() {
2   Serial.begin(9600);
3   Serial.println("Hello world");
4 }
5
6 void loop() {
7
8 }
```

- Line 3: Since the Arduino has a very limited user interface, we will use the workstation and the *Serial Monitor* application to interact with the micro-controller via the USB cable. In line 3, the command initializes the speed of the serial communication on the cable. The same speed must be selected in the Serial Monitor application.
- Line 4: the command `Serial.println` outputs messages that can be received and displayed by the Serial Monitor.

1.5 Instructions

- Start the Arduino IDE application on the workstation. Open the Serial Monitor Window and check that the speed of the serial port is set to 9600.
- Make sure the Arduino is connected to the workstation via the USB cable. Upload the program on the Arduino.
- You should see the "Hello world" message in the serial monitor window.
- Add a `Serial.println` command in the body of function `loop` (on line 7, between { and }) to output message "Hello again". Please observe the specific syntax of the command:
  Serial . println (message) ;
  The parenthesis and semicolon are mandatory. Upload the modified program. What do you notice?
- The command delay(value); instructs the micro-controller to take a pause of a specified number of milliseconds. That is, the following instruction will be executed after the given number of milliseconds (value) have passed. This number must be provided in the command, between the parenthesis. For example, delay(200); creates a pause of 200 milliseconds.
  What value should be given to the command to create a pause of 1 second? If you are required to output the message "Hello again" only once every second, where would you insert the command in the Arduino code?
  Implement the change and run the program.
- A computer is sometimes used to perform calculations. For example, the Arduino can be used as a basic calculator. For example, the following piece of code outputs the product of 7 and 8:

```
1     void setup() {
2       Serial.begin(9600);
3       Serial.println(7*8);
4     }
5
```

  The basic arithmetic operations are +, -, *, /, and % (sum, difference, product, division, and modulus).
  Create a new Arduino project with no commands for the loop function but with the appropriate command to set-up the speed of the serial communication module (see the sample code above).

Write code that outputs the perimeter of a rectangular room with sides 12 and 17 feet. Use the arithmetic operations.

– Some mathematical operations are available as functions. For example, computing a^b for some numerical values a and b is achieved using function `pow`:
pow(a, b);
Information about the Arduino functions is available at `https://www.arduino.cc/reference/en/#functions`. Using this reference, modify your code to display $\sqrt{3.14}$. Then, write another println command that computes $3.14^{\frac{1}{2}}$.

– A computer program has the ability to represent and store data, using variables. For example, we can store numerical values in variables and we can perform numerical computations on those variables.
Variables are locations in memory that have a name. To create such a location, we need to *define* or *declare* the variable.
The code **int** a; defines a variable with the name "a". Variable "a" can hold integer values (**int**). Other types of values are byte (integers between 0 and 255), **long** (integers from -2^{31} to $2^{31}-1$), **float** (decimals between approximately -3.4×10^{38} to 3.4×10^{38}), and String (text).
The following program declares a *global* integer variable with the name "x". The code, otherwise does not use the variable. Notice that the variable is declared outside of the functions loop and setup!

```
1   int x;
2
3   void setup() {
4     Serial.begin(9600);
5   }
6
7   void loop() {
8   }
9
```

To do: modify your existing code (or start a new project if you prefer). Then, write the code given above. Compile the project to see if it compiles correctly. Declare a second variable that can hold decimals with the name "f".

– To be used, variables need to be initialized or given a value. When declaring a variable, we can also initialize it. For example, we can declare and initialize variable "x" to zero:
int x=0;
Modify your code to declare and initialize "x" to zero and "f" to 1.5.

– Once initialized, variables can be used in calculations or they can be displayed on the serial monitor in the same way we displayed expressions or calculations. Examples:
Serial.println(x);
Serial.println((x+7)/2);
These examples must be written in the body of functions. Write in your code the instruction to display the value of "x" in the body of function *loop*. Add also the appropriate delay command so that the value is displayed only once every second.

– The advantage of a variable lies in the ability to change the value stored by the variable. We use the operator "=" to assign new values to the variable. This operation is called *assignment*. Examples:
x = 1;
x = x + 1;

The last command increases the value of the variable by one, and it is a very common operation for programmers.

The syntax of the assignment operation is as follows:

variable to be modified = expression for the new value ;

Modify your code so that you increase the value of variable "x" by one after you display the value of "x" in the loop function. Compile and upload the code.

Congratulations. You are now familiar with the the basic instructions for an Arduino program. More information is available from the Arduino Notebook by Brian Evans, available from the course web page.

1.6 Exercises

1. Write Arduino code that displays on the Serial monitor a count down from 1000, every second. Example: the counter displays first 1000, then 999 after 1 second, then 998, etc.

2. Reduce the time between successive counts for the count down counter to half a second.

3. (*) Using only arithmetic expressions, create an Arduino program that displays two counters one after the other. The first counter increases by one every second, the second increases by one every 10 seconds. For example, the program should display 0 0, then after a second 1 0, then 2 0, ..., 9 0, 10 1, 11 1, ... 19 1, 20 2, etc.

Chapter 2

Activity 1: blinking led (digital output)

2.1 Lab activity objectives

Digital data is binary data or data represented by 1-s and 0-s. The hardware of a computer encodes 1-s and 0-s for example, using the presence or absence of voltage. In this session, students will learn how an Arduino micro-controller outputs digital information on its digital pins, how bits[1] are represented, and how we can use the voltage that represents digital information to turn a LED on. Digital output has many uses in practical systems. For example, the digital output from an Arduino micro-controller can control sensors, such as the cheap ultrasonic distance sensor HC-04. The circuit with LED described here illustrates the steps needed to work with digital output in a simple and direct way.

In this session, students will:

- use the Arduino IDE[2] application to upload a simple program to an Arduino micro-controller,
- recognize the polarity of a LED[3],
- connect a LED, resistor, and Arduino using wires and a breadboard[4] according to a given circuit diagram,
- extend the circuit by adding a second resistor and LED, and extend the Arduino program to perform a new function,
- draw the diagram for the new circuit.

2.2 Component list

Including the components necessary for the exercises, the following parts are needed:

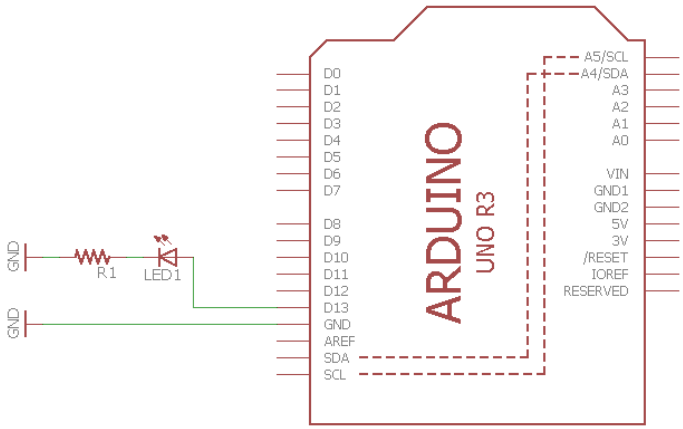
- 1 x Arduino micro-controller
- 1 x USB cable
- 1 x breadboard
- 5 or 6 M-M jumper wires of assorted length
- 2 x 220 Ohm resistor
- 2 x LED

[1] **bi**nary digi**ts**

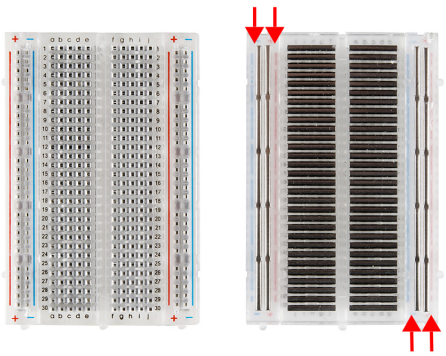
[2] Integrated Development Environment

[3] Light Emitting Diode

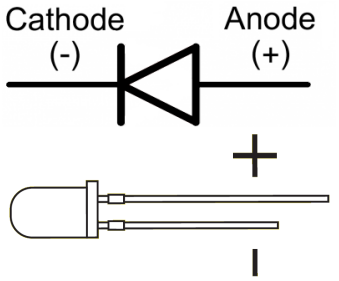
[4] A breadboard is used to create circuits that can be easily reconfigured using wires (see Fig. 2.1b).



(a) Circuit diagram for Activity 1



(b) Breadboard internal connections, source `learn.sparkfun.com`



(c) Polarity of a LED, source `learn.sparkfun.com`

Figure 2.1: Circuit set-up

2.3 Circuit set-up

- Select the breadboard from the component box and examine the circuit diagram from Fig. 2.1a. Using some of the jumper wires and the breadboard, you will set-up the circuit in the diagram, as follows.
- Select one LED from the component box and identify the polarity of its pins (see Fig. 2.1c). The LED works only if inserted in the correct polarity according to the diagram. Insert the LED into the breadboard, making sure its pins are inserted in different rows on the breadboard (why?).
- Select one resistor from the component box. Insert one pin of the resistor on the row containing the LED cathode. The resistor can be inserted in any direction. Insert the other pin of the resistor to the negative breadboard power bar (the power bars are marked by red arrows in Fig. 2.1b).
- Use a jumper wire to connect the row containing the LED anode to digital pin 13 (D13) on the Arduino. Use another jumper wire to connect the Arduino GND pin to the negative breadboard power bar containing the pin of the resistor (the two negative/positive power bars on the breadboard are not interconnected).
- The circuit from Fig. 2.1a is now complete.

2.4 Arduino program

– Line 3: an Arduino has digital and analog pins (which we will examine in future activities), and the digital pins can be configured to either read digital information (input), or output digital information. On line 3, we tell Arduino to use its pin number 13 for output.
– Line 8: in output mode, the digital pin can be either in HIGH mode, representing bit 1, or low mode, representing bit 0. In this case, we set the pin to output bit 1. This information is encoded by a certain voltage present on the pin. Exercise 1 asks you to determine the voltage encoding this information.
– Line 10: we set pin 13 to output bit 0.
– Lines 9 and 11: we ask the Arduino board to pause, which means that the command following function *delay* will only be executed after 1000 milliseconds (1 sec) have passed (see Chapter 1).

Listing 2.1: Arduino code for Activity 1

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);    // Turn on the LED
9    delay(1000);               // Wait for one second
10   digitalWrite(13, LOW);     // Turn off the LED
11   delay(1000);               // Wait for one second
12 }
```

2.5 Instructions

– Assemble the circuit according to the instructions from Section 2.3.
– Connect the Arduino to your computer using the USB cable from the component box. The Arduino micro-controller is powered by the computer, via the USB cable. This power is sufficient to light a few LEDs, but for more energy hungry components such as motors, we will need a different set-up.
– Start the Arduino IDE on the computer. Navigate to the *Tools/Ports* menu and select the port with a description containing the name "Arduino". Ensure that *Tools/Board* is set to "Arduino/Genuino Uno".
– Navigate to the *File* menu and load the lab code provided by the instructor. Convince yourself that the code displayed matches Listing 2.1.
– Compile and upload the code to the Arduino by clicking the arrow icon on the toolbar. The check-mark icon to the left will compile the sketch without uploading it to the Arduino. This is useful for checking that your program is syntactically correct. It is a good idea to save frequently, compile early, and compile often.
– Once the IDE has completed the upload, the circuit LED should now be flashing.

2.6 Exercises

1. Use a voltmeter to determine the voltage representing bit 1 and the voltage representing bit 0 in an Arduino circuit.

2. Change the Arduino program so that the LED stays on for 2 seconds and off for 1 second.

3. Change the Arduino program so that the LED blinks 4 times in a second.

4. Extend the circuit to add the second LED and resistor in such a way that the two LEDs can be independently turned on or off.
   (a) Make a drawing of your new circuit.
   (b) Extend the Arduino code so that your circuit blinks the two LEDs, alternatively, using a pattern similar to that of the lights at a railway crossing. Set the delays so that each LED is on for one second and off for one second.

5. Modify the code from Exercise 4 to implement different blinking patterns, for example one LED blinks twice as fast as the other, or both LEDs blink at the same time and with the same frequency, or both LEDS blink with the same frequency but one LED starts 250 milliseconds after the other, etc.

Chapter 3

Activity 2: potentiometer (analog input)

3.1 Lab activity objectives

Many sensors provide information via an analog signal, more exactly via a voltage that can take any value between 0V and 5V. In this activity, students will program the Arduino board to read the variable voltage obtained over the pins of a variable resistor or potentiometer. The value will be sent, via the USB cable, to the host computer to be displayed. More precisely, the students will:

– identify the pins of a variable resistor,
– assemble a circuit using the variable resistor,
– program the Arduino board to measure the voltage of an analog signal using the Arduino's analog pins,
– program the Arduino to send the value read from the analog pin, over the serial USB connection, to the workstation,
– open the serial monitor window in the Arduino IDE on the workstation to view the value read by the analog pin.

3.2 Component list

Including the components necessary for the lab assignment, the following parts are needed:

– 1 x Arduino micro-controller
– 1 x USB cable
– 1 x breadboard
– 8 or 10 M-M jumper wires of assorted lengths
– 1 x 220 Ohm resistor
– 1 x LED
– 1 x 10k Ohm Variable Resistor (Potentiometer)

3.3 Circuit set-up

The circuit set-up is very simple for this activity. We need to identify the pins on the potentiometer first. The resistance between the middle pin and one of the outer pins is variable. The connection in Fig. 3.1 can be interpreted as two series connected resistors for which the sum of their electrical resistance is constant and equals 10 k Ohm, but each resistor has a variable resistance. This way,

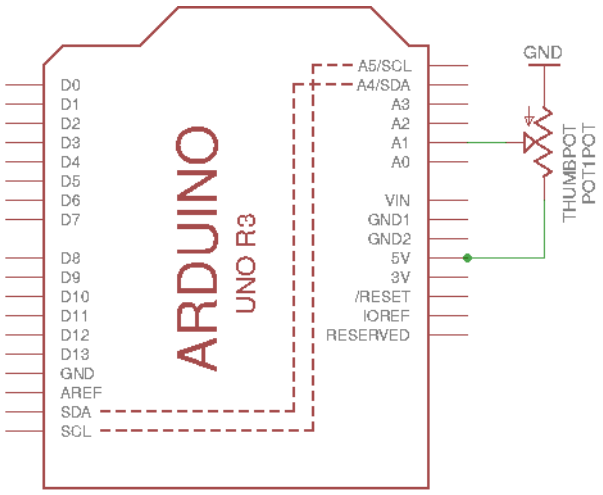


Figure 3.1: Circuit set-up for Activity 2

the current from the +5V pin to the GND pin on the Arduino is limited, but the voltage over the resistors is variable. We are measuring on Analog pin 1 the voltage over one of the resistors and we output this value by communicating it to the host computer which will display it.

3.4 Arduino program

The short program demonstrates the use of serial communication (see Chapter 1, and commands for reading the voltage applied to an analog pin on the Arduino.

Serial communication: the syntax appears slightly particular, as function calls are prefixed by the identifier `Serial`, which identifies the so called serial port, followed by period and the function name. The port `Serial` is used for the connection via the USB cable, and depending on the type of the Arduino board, other physical connections may be available, such as `Serial1`, etc. In this document, we will only use `Serial`.

To initialize the serial port, we must first call function `begin` which needs one integer argument that specifies the speed of communication, namely how many symbols per second to transmit. This speed is known as the baud rate. In Listing 3.1, line 5, we use the typical rate of 9600 baud. At the other end of communication, we must configure the serial monitor of the Arduino IDE running on the workstation to use the same speed value, 9600 baud (see Chapter 1).

On line 12 we output the numeric value provided as argument to the serial monitor window of the Arduino IDE running on the workstation.

Analog input: The six analog pins numbered from 0 to 5 are only capable of input. Function `analogRead` returns an integer in the range 0 to 1023, where integer 0 corresponds to a voltage of 0 V on the analog pin, and value 1023 corresponds to the maximum voltage of 5 V that can be measured on the pin. The function argument is an integer from 0 to 5 that identifies the pin used for input.

The last command in function `loop` is a call to the delay function. We do not need to sample the voltage on the analog pin too frequently.

Listing 3.1: Arduino code for Activity 2 (potentiometer)

```
// input pin:
const int potentiometer_pin = 1 ;

void setup() {
  Serial.begin(9600); // initialize the serial port
}

void loop() {
  // variable to store the voltage
  int value;
  value = analogRead(potentiometer_pin);
  Serial.println(value);  // send the value to the serial monitor window
  delay(500);  // avoid reading too frequently...
}
```

3.5 Instructions

- Assemble the circuit in Fig. 3.1.
- Start the Arduino IDE program and select the appropriate port for the Arduino board as in the previous activities.
- Select menu Tools, then Serial Monitor. In the window that opens, make sure that 9600 baud is selected from the pull down widget.
- Download the starting code provided by the instructors and upload it. The serial monitor should display the value read from Analog pin 1, every 500 ms.
- Turn the knob on the potentiometer and observe how the value shown by the serial monitor changes.

3.6 Exercises

1. The value of the resistance between the outer pins on the potentiometer is 10 k Ohm. Calculate the value of the current flowing from pin +5 to GND on the Arduino. As one turns the potentiometer knob changing the voltage applied to the analog pin, do you expect that the value of the current will change? Explain.

2. Suppose we change the circuit from Fig. 3.1 by moving the connector from pin +5V to pin +3V on the Arduino. What is the range of values that are likely to be returned on the Serial monitor in this case? You may assume that integer values are assigned uniformly in the range 0 V to 5 V.

3. Add an LED and a 220 Ohm resistor to the circuit from Fig. 3.1 and connect the LED to one of the digital pins in the range 2 to 13 (pins 0 and 1 are used during the transmission of data over the serial link). Copy the *blink* function from the sample code provided on Moodle for lectures into the code from Listing 3.1. Adjust the code so that the duration the LED is on or off equals to the value read by the analog pin in milliseconds (the LED should be on for a time equal to the value read in milliseconds, and off for an equal amount of time). Your program should continue to send the value read by the analog pin to the serial monitor.

4. Add two LED's to the circuit from Fig. 3.1, and arrange them so that one LED is on the left, the other on the right. Program the Arduino so that when the voltage measured by the analog pin is in the range 0V to 2V, the left LEDs is on and the right one¡ is off. When the voltage is between 3V and 5V, the right LED is on and the left is off. When the voltage is between 2V and 3V, both LEDs should be on.

Chapter 4

Activity 3: push button (digital input)

4.1 Lab activity objectives

Any interesting Arduino project should allow for some user input. The simplest way for a user to interact with an Arduino project is through buttons. In this lab activity, we investigate a circuit set-up and the corresponding Arduino program that allows the micro-controller to read the state of a momentary button[1] which can be either in pressed or depressed state. The circuit we use is a standard one using a so called *pull-up resistor*. A pull-up resistor connects the input pin on the Arduino to +5V when the button is not pressed (bit 1) and to 0V when the button is pressed (bit 0). By reading the input pin, the micro-controller determines the state of the button.

In this session, students will:

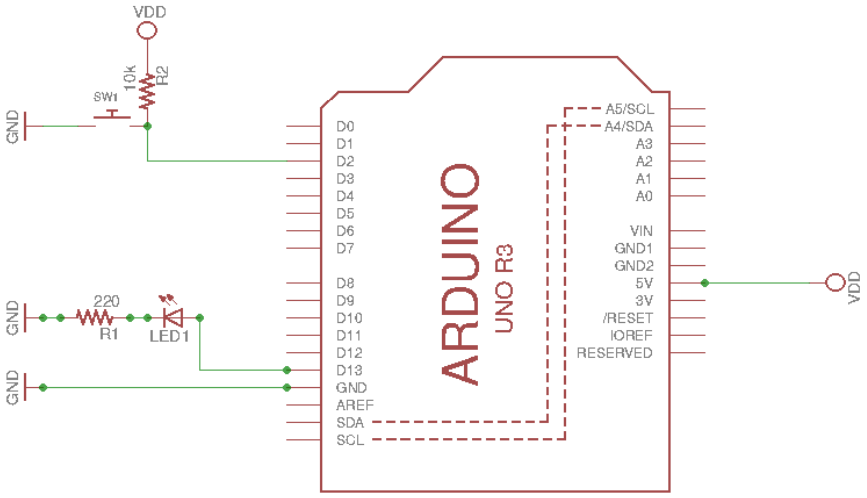
– use a breadboard, an LED, and resistors to examine the wiring of the push button and to determine which pins on the button become connected when the button is pressed,
– use a breadboard to create a pull-up circuit for the Arduino using a resistor and the push button,
– combine this circuit with the circuit from Chapter 2 so that pressing the button will change the state of the LED as follows.
  - Initially, the LED is off.
  - When the button is pressed, the state of the LED is toggled: if the LED is off, it will be turned on, and if it is on, it will be turned off.
– use appropriate delay values in the Arduino program to read the state of the button reliably.

4.2 Component list

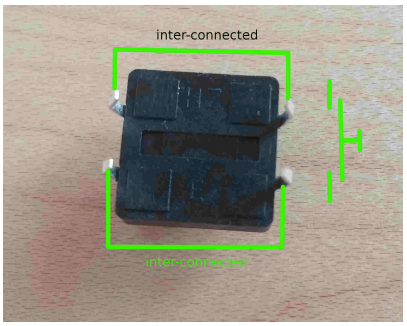
Including the components necessary for the lab assignment, the following parts are needed:

– 1 x Arduino micro-controller
– 1 x USB cable
– 1 x breadboard
– 8 or 10 M-M jumper wires of assorted lengths
– 1 x 220 Ohm resistor
– 1 x LED
– 1 x 10k Ohm Resistor (Pull-up)
– 1 x push button

[1] A momentary button closes the circuit, or connects the two terminals together, for as long as the button is pressed.



(a) Circuit diagram for Activity 3



(b) Back side of the push button and its internal connections

Figure 4.1: Circuit set-up for Activity 3

4.3 Circuit set-up

Use the breadboard and the supplied parts to assemble the circuit from Fig. 4.1a. Please take note of the identity of the Arduino pins used. The sample program is written on the assumption that pin D2 is connected to the push button and pin D13 is connected to the LED.

The supplied push button has four pins that can be inserted in the breadboard. Two of the four pins are inter-connected internally. The other two are inter-connected as well. See Fig. 4.1b to identify the pins that are inter-connected. When the button is pressed, the two pairs of inter-connected pins become connected to each other (all four pins are inter-connected). When the button is depressed, the connections return to their initial configuration. To be able to use the button, make sure you insert it in the breadboard in such a way that the pair of pins that are farther apart share the same row on the breadboard. To have room for additional connections, the button should straddle the vertical axis of symmetry of the breadboard. For example, the button pins should be inserted in columns *d* and *g*.

4.4 Arduino program

In this program, we use the global variable LED_state to remember whether the LED is on or off. We define two functions, isButtonPressed which returns true if the button is pressed, and setLED which accepts one argument that specifies what state we wish the LED to be in: false for LED off and true for LED on. Line 16 is responsible for the logic of this program: the state of the LED changes from continuously on to continuously off each time the button is pressed.

Notice the delay value from function isButtonPressed, which ensures that we do not read the state of the input pin too frequently, but also not too infrequently so that we miss most of the button presses. In function loop, the larger delay value ensures we do not immediately read the state of the button to change the state of the LED again.

Consider what happens if we keep the button pressed. How about if we change the delay values in the code significantly? How about if we do not have delay values at all? Try to modify the code, if you have time during the lab, to investigate these questions.

Listing 4.1: Arduino code for Activity 3 (push button)

```
1  const int button_pin = 2;
2  const int LED_pin = 13;
3  int LED_state;  // state of LED: false (0) = off, true (non-zero) = on
4
5  void setup() {
6    pinMode(LED_pin, OUTPUT);
7    pinMode(button_pin, INPUT);
8    LED_state = 0;  // LED is initially off
9    setLED(LED_state);  // turn the LED off initially
10 }
11
12 void loop() {
13   //Read button state//
14   if (isButtonPressed()) {
15     // toggle LED state only if button is pressed
16     LED_state = !LED_state;  // toggle the state variable
17     setLED(LED_state);   // light the LED according to state
18     delay(500);  // once state is changed, do not read the
19                  // button too quickly to change LED back again
20   }
21 }
22
23 // RETURN: 0 (false) if button is not pressed
24 //         non-zero (true) if button is pressed
25 int isButtonPressed() {
26   int button_state = digitalRead(button_pin);
27   // we do not wish to read the input pin too frequently...
28   delay(125);
29   // if button pressed, we read LOW
30   return (button_state == LOW);
31 }
32
33 // state = true: turn the LED on, otherwise turn LED off
34 void setLED(int state) {
35   if (state) // on
36     digitalWrite(LED_pin, HIGH);
37   else
38     digitalWrite(LED_pin, LOW);
39 }
```

4.5 Instructions

- Assemble the circuit according to the instructions from Section 4.3. Verify that you are using the correct resistors in the connections and that you are inserting the button in the breadboard correctly.
- Connect the Arduino to your workstation using the USB cable, open the Arduino IDE, and load the sample code provided by the instructor.
- Upload the code and test the behaviour of the circuit.
- Complete the lab assignment given by the instructor.

4.6 Exercises

1. The Arduino board has built-in pull-up resistors for its digital pins. Read the online documentation about how to enable the built-in pull-up resistor. Modify the circuit to eliminate the 10k Ohm pull-up resistor and change the program if needed by your new circuit but without changing the behaviour of the circuit.

2. Use the circuit from Fig. 4.1a. Modify the program so that the LED has three possible states: continuously on, continuously off, and blinking with a frequency of 5 blinks per second. Pressing the button should now change the state of the LED among these three states.

3. Assemble the circuit from Fig. 4.1a and load the program from Listing 4.1. Add a second button. Change the program so that the button is blinking continuously with a frequency of 1 Hz (1 blink per second). Pressing one of the button should increase the blinking frequency by 1 Hz and pressing the other should decrease the frequency by 1 Hz. For simplicity, consider that the minimum frequency is 1 Hz.

4. Start with the circuit from Fig. 4.1a. Add one light sensor brick or one sound sensor brick to your design (connect the + or V pin on the sensor to 5V pin on the Arduino, the - or G pin on the sensor to the GND pin on the Arduino, and the S pin on the sensor to one of the analog pins on the Arduino, like demonstrated in the lecture). Then extend the program from Exercise 2 so that the state of the LED changes between the three states when the button is pressed AND also when the sensor detects a change in the environment. Ideas for environment conditions that can trigger LED state changes: light intensity increases from low (below a certain threshold) to high value (above the threshold), the intensity of sound is above a certain threshold.

Chapter 5

Activity 4: variable LED intensity (analog output / PWM)

5.1 Lab activity objectives

In the previous activities, we learned how the Arduino board can process input in the form of a digital and an analog signal, as well as how to generate a digital signal for output. The goal of this activity is to output an analog signal. Such an operation is useful when controlling the rotation speed of a direct current (DC) motor. We saw from the lectures that the Arduino cannot generate a truly analog signal (it cannot supply electricity with a voltage that can be programmed between 0 V and 5 V). However, the speed of a DC motor is not determined by voltage but by the energy transferred from the source to the motor in the unit of time (also known as power).

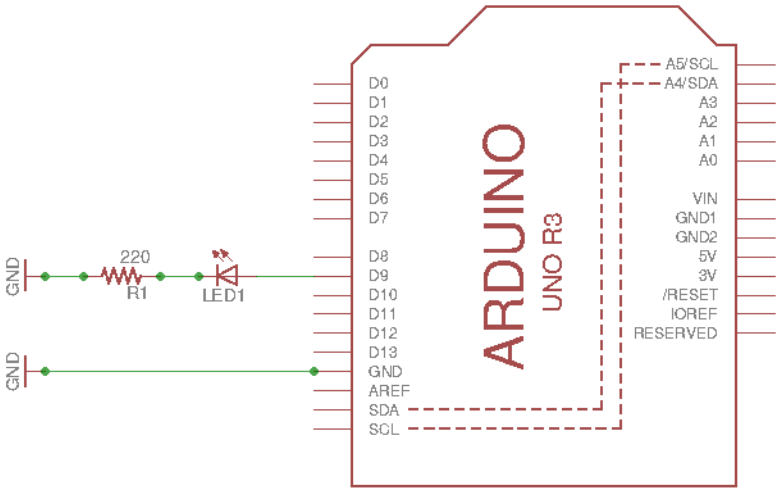
Of course, the power transferred to a resistor for example, is proportional to the square of the voltage across the resistor, as we might know from high school or first year physics courses[1]. Pulse width modulation (PWM) is another way to control the power transferred to a circuit, as explained in the lecture. In this activity, students will use the PWM capable digital pins on the Arduino to control the amount of power transferred to an LED. They will effectively control the light intensity of the LED. The same code and principles could apply to controlling the speed of rotation of a DC motor, except that the power needed by the motor is larger than the power the Arduino can transfer. We will see in Chapter 6 that we can use an additional board, a motor shield, which can supply the additional power needed from a pack of four AA batteries.

More exactly, in this activity, the students will:
– identify the PWM enabled digital pins on the Arduino board,
– connect an LED to PWM capable pin 9, using a circuit identical to that from Chapter 2,
– program the Arduino to vary the pulse width of the signal it generates on pin 9,

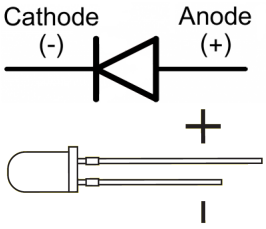
5.2 Component list

The list of components for this activity is the same as for the activity described in Chapter 3. For convenience, the components needed for the activity and the lab assignment are enumerated below:
– 1 x Arduino micro-controller

[1]Check's Ohm's law and the definition of electric power to verify that $P = \frac{U^2}{R}$, where P represents power, U voltage, and R resistance.



(a) Circuit diagram for Activity 4



(b) Polarity of a LED, source `learn.sparkfun.com`

Figure 5.1: Activity 4 circuit

- 1 x USB cable
- 1 x breadboard
- 8 or 10 M-M jumper wires of assorted lengths
- 1 x 220 Ohm resistor
- 1 x LED
- 1 x 10k Ohm Variable Resistor (Potentiometer)

5.3 Circuit set-up

We first identify the digital pins capable of PWM. On the Arduino Uno, these are pins 3,5,6,9-11. On the board, these pins are marked by symbol ~ next to the pin number. The Arduino program uses pin 9, so we will connect the LED to pin 9, taking care of the LED polarity as in Fig. 5.1b (the Arduino pin connects to the positive pin on the LED).

5.4 Arduino program

The source code in Listing 5.1 demonstrates the use of one new function.

analogWrite: takes two arguments. The first, identifies the digital pin on which the pulse width modulated signal is output. The second corresponds to the amount of power transferred via the pin. It is an integer in the range 0 to 255 inclusive, where 0 corresponds to no power transferred and 255 corresponds to maximum power transferred.

We note that, to use PWM, the pin does not need to be configured in any special way in the `setup` function.

Listing 5.1: Arduino code for Activity 4 (variable LED intensity)

```
1 const int LED_pin = 9; // PWM pin
2
3 void setup() {
```

```
4   randomSeed(analogRead(1));
5 }
6
7 void loop() {
8   int LED_intensity = random(0,256);
9
10  // change the intensity
11  analogWrite(LED_pin, LED_intensity);
12  delay(1000);
13 }
```

5.5 Instructions

– Assemble the circuit in Fig. 5.1. Connect the Arduino to the workstation via the USB cable.
– Open the Arduino IDE, download the source code for the activity and open the source code in the Arduino IDE.
– Compile and upload the program on the Arduino.
– You should notice that the LED intensity varies in a random fashion, maintaining its intensity for 1 second.

5.6 Exercises

1. Modify the program in Listing 5.1 so that the intensity of the LED is assigned randomly from the range 127 to 255.

2. Using while statements, write a program that increases the intensity of the LED gradually from 0 to 255 in an interval of approximately 1 sec, then it decreases the intensity from 255 to 0 in the same interval of time.

3. Modify the program from Exercise 2 so that the LED stays off for 1 sec at the end of fade off, and on for 1 sec at the end of fade in.

4. Modify the program from Exercise 2 so that the speed of fade in and out is twice as slow (the LED changes from off to maximum intensity and vice versa within 2 sec).

5. Add a second LED to the circuit in Fig. 5.1a and change the program from Exercise 2 so that the two LEDs fade in and out alternatively (one LED fades in and the other fades out and vice versa).

6. Add a potentiometer to the circuit in Fig. 5.1a using the circuit from Chapter 3 as reference. Modify the code in Listing 5.1 to read the value of the potentiometer and to adjust the intensity of the LED as a function of the position of the knob on the potentiometer. More precisely, the LED should be off when the analog signal read is 0 and should have maximum intensity when the analog signal read is 1023.

7. Extend Exercise 6 with a second LED in the circuit. Position the LEDs in such a way that one of the LEDs is to the left of the other. Program the Arduino so that when the knob of the potentiometer is completely turned to the left, the LED on the left has maximum intensity while the LED on the right is off. Similarly, when the knob is completely turned to the right,

the LED on the right is on and the one on the left is off. Turning the potentiometer on any intermediate position will modify the intensity of the two LEDs proportionately.

8. Modify the program in Listing 5.1 so that the value output by the analogWrite function is between 0 and 511. What can you say about the behaviour of analogWrite outside the documented range of values?

Chapter 6

Activity 5: variable speed DC motor (PWM via motor shield)

6.1 Lab activity objectives

In this Chapter, we will learn how we can use the Arduino Uno board to control the speed of rotation of a DC motor.[1] In Chapter 5 we investigated how we can use the Arduino Uno board to simulate an analog signal with pulse width modulation. This signal was used to control the intensity of an LED. The same principles apply to controlling the speed of DC motors, except that the motors require more energy than what the Arduino board can supply via its PWM signal. For this reason, our circuit will use a so called *Motor shield* which is a circuit board very much similar to the Arduino Uno in size, but that has pins which can insert on top of the Arduino board. The programming of this *motor shield* is also a little different, requiring the use of a few special functions available from the motor shield library. Instructions how to enable this library in the Arduino IDE are available in Section 6.5.

6.2 Component list

Including the components necessary for the lab assignment, the following parts are needed:
– 1 x Arduino Uno micro-controller
– 1 x USB cable
– 1 x Adafruit motor shield version 2.3
– 1 x battery holder with 4 AA batteries
– 2 x DC motors

6.3 Circuit set-up

The circuit for this activity is very simple. It does not need a breadboard. First, the motor shield needs to be inserted on top of the Arduino Uno. There is only one orientation of the shield for which all of the male pins on the shield are inserted in the Arduino female slots. The circuit in Fig. 6.1 shows only the connections on the motor shield. Essentially, the DC motor needs to be

[1] A DC motor (direct current motor) is the simplest motor that is powered by a source of continuous electrical current.

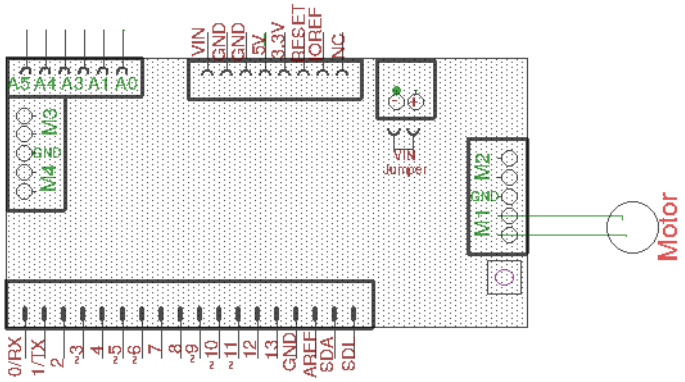


Figure 6.1: Circuit diagram for Activity 5

connected to the shield. In addition, the Arduino board needs to be powered from the battery holder via the power jack. The USB cable should also be connected to allow our programs to be uploaded.

Important: the Adafruit motor shield uses a jumper to configure how the motor shield receives power. The jumper should short-circuit the *Vin jumper* pins as shown in Fig. 6.1. This way, the shield receives power from the Arduino board and the four AA batteries that supply it.

6.4 Arduino program

The first requirement for a program that uses the Adafruit motor shield is to load the library functions. This is achieved using the three include directives at the beginning of the program. Line 6 declares and initializes a special variable, called an object. This object is a representation, in code, for the Adafruit shield. An object does not only store values like any regular variable, but also can call functions. On line 9, the shield object calls function getMotor to initialize another object, which, this time, is a representation of a motor connected to the shield. Through this object, the programmer can turn the motor off (RELEASE), move it in one direction (FORWARD) or the other (BACKWARD). The object can also control the speed of rotation of the motor using function setSpeed().

Listing 6.1: Arduino code for Activity 5 (DC motor control with shield)

```
1  #include <Wire.h>
2  #include <Adafruit_MotorShield.h>
3  #include "utility/Adafruit_MS_PWMServoDriver.h"
4
5  // Create the motor shield object with the default I2C address
6  Adafruit_MotorShield AFMS = Adafruit_MotorShield(); //
7
8  // Select which 'port' M1, M2, M3 or M4. In this case, M1
9  Adafruit_DCMotor *myMotor = AFMS.getMotor(1); //
10 // OBS: can declare another motor variable on a different port
11
12 void setup() {
13   AFMS.begin();  // create with the default frequency 1.6KHz
14   //AFMS.begin(1000);  // OR with a different frequency, say 1KHz
15 }
16
```

```
17 void loop() {
18   myMotor->setSpeed(80);  //  speed betwen 0 and 255 max
19   myMotor->run(FORWARD);  //  turn on (other
20         // values: BACKWARD for reverse, RELEASE to stop)
21   delay(3600000);         // wait for an hour
22 }
```

6.5 Instructions

- Connect the motor shield with the Arduino board.
- Make sure the Vin jumper is on (creating a short-circuit of the two pins). If the jumper is not on, the shield circuit will not be powered.
- Cut / rip a small piece of the paper tape, and stick it on the shaft of the DC motor in a configuration that resembles a propeller. This will help with detecting the speed of the motor. Next, connect the two wires on the motor to the two sockets labelled M1.
- Connect the battery holder and USB cable to the Arduino.
- Open the Arduino IDE. Verify that the Adafruit motor shield library is installed in the Arduino IDE by following the steps described at `https://learn.adafruit.com/adafruit-motor-shield-v2-for-install-software`
- Download the code for this activity from Moodle. Make sure the IDE uses the correct serial port to communicate with the Arduino.
- Upload the code and observe the behaviour of the DC motor.

6.6 Exercises

1. Using the code from Listing 6.1 as starting point, modify the program to achieve the following tasks. For each of the tasks listed, save the program in a separate file.
   (a) The motor turns in the opposite direction.
   (b) The motor doubles its speed of rotation.
   (c) The motor runs at its maximum speed, in the FORWARD direction.
   (d) Try to find the smallest speed value that allows the motor to move, and set the motor to turn in the FORWARD direction at that speed. Explain your strategy for finding the minimum value for rotation.
   (e) Using the speed of 100, turn the motor on for 1 sec and off for 1 sec; repeat.
   (f) Using the speed of 80, Turn the motor on for 1 sec and change the direction of rotation for 1 sec; repeat.

2. Add a second motor to the circuit and modify the program from Exercise 1 so that one motor is on for 1 sec and the other is off, after which the roles of the motors are exchanged. Repeat.

3. Change the program from Listing 6.1 so that the motor increases its speed from 0 to maximum gradually, in speed increments of 5 units (each speed should be maintained for 20 ms). Once the maximum speed is reached, the motor maintains its speed for 1 sec, after which the motor should be turned off for 1 sec. This process then repeats.

4. Add a potentiometer and, with the help of a breadboard, connect the circuit like in Chapter 3. Use the potentiometer to control the speed of the motor.

5. Add a a button to the circuit from Exercise 4. Pressing the button will cycle the motor from off, forward, and backward rotation states, while the potentiometer will still control the rotation speed.

6. Add a second motor to the shield. Complete Exercise 3 but making sure the second motor turns in opposite direction from the first but with the same speed of rotation.

7. Add a second motor to the shield. Complete Exercise 5 but making sure the second motor turns in opposite direction from the first but with the same speed of rotation.

8. Add a second motor to the shield and arrange them one to the left and the other to the right. Complete the circuit from Exercise 4 and program the following behaviour: When the value read of the potentiometer is 512, both motors should turn with the same speed, equal to 128. If the value read on the potentiometer equals x (x ranges between 0 and 1023), the motor on the right should run at a speed y, proportional to x, while the motor on the left should run at speed $255 - y$.

Part II

Robotics Project

Chapter 7

Project challenges

In Part I of this manual we covered four basic operations on that the Arduino micro-controller can perform: input and output of digital signals, and input and output of analog signals. We learned in the lectures and homework assignments about a basic programming template that allows our micro-controller to process input and generate signals, practically at the same time. We are now ready begin work on our robotics project.

In the remaining weeks of this course, students will program a small two wheel robot to complete the following task. The robot must approach a target box no farther than 20 cm away, but without touching it. Several challenges with increasing degrees of difficulty will be proposed and students will need to meet them in order to get graded.

In addition, the students are asked to maintain a project log which will also be graded. The structure of the log is described in the lecture notes.

7.1 Component list

Each lab kit should contain the following basic set of components that must accompany every project activity:
- 1 x Arduino Uno micro-controller
- 1 x USB cable
- 1 x Adafruit motor shield version 2.3
- 1 x battery holder with 4 AA batteries
- 1 x three wheel robot.
- 1 x Ultrasonic distance sensor.
- 2 x Light sensor.
- An assortment of connection wires: male-male, male-female, and female-female, say 5 of each type.

In addition, the students can choose to add, from a special project supply box, various other components that they might wish to incorporate:
- Resistors (220 Ω).
- Potentiometers (10 kΩ).
- LED-s.
- Pushbuttons.
- Breadboards.

7.2 Challenge 1: Straight line trajectory

For this challenge, the robot will be placed at a distance of exactly 1 m, measured from the front of the robot to the target box. The robot is facing the box. The robot should be stationary initially, to allow this placement. In order to reach the target, the robot must travel on a straight line for at least 80 cm, but for no more than 100 cm. The robot should remain stationary at the end of its travel, to determine whether the challenge was met or not.

7.2.1 Ideas and suggestions

1) Before starting any activity for the project, record your work in the project log as discussed in the lectures.
2) Connect the shield and motors as for Exercise 6 from Chapter 6.
3) Download the sample Arduino code which contains the configuration code for the motor shield.
4) Add code that makes the robot move straight for exactly one second. Choose the speed of movement by experimenting. A high speed might drain the batteries too quickly, a low speed might take too long to move the robot for a given distance. Remember, to upload your code you need to connect the Arduino to the workstation using the USB cable, but to test the code, you should un-tether the Arduino and place the robot on the ground.
5) It is possible that the trajectory of the robot is not perfectly straight as the motors are not identical. If the trajectory deviates from a straight line too significantly for your taste, consider compensating by adjusting your code to feed different PWM signals to the motors.
6) You can use a ruler to measure the distance travelled by the robot during one second. Perform three or four experiments and record the distance travelled during each experiment. Let x be the average distance travelled by the robot in one second, measured in metres. Then, to make the robot move forward for approximately 1 metre, keep the motors on for time $t = \frac{1}{x}$ seconds. Use this information to write the code that will move the robot within 20 cm of the target box.
7) In order to facilitate the positioning of the robot in front of the target box, the robot must be stationary at first. One can use delays to implement this behaviour, but a more reliable approach is to add a pushbutton to the robot. The robot is initially stopped. Pushing the button will make the robot move straight and stop no farther than 20 cm from the target box and without touching the target box. After this is completed, the robot should be stopped. Pushing the button, will make the robot repeat its behaviour. Consider using the programming template discussed in class.

7.3 Challenge 2: straight line trajectory, but variable distance

For this challenge, the robot will be placed at a distance that can vary between 1 and 2 m, measured from the front of the robot to the target box. The robot is facing the box. The robot should be stationary initially, to allow this placement. In order to reach the target, the robot must travel on a straight line and stop no farther than 20 cm from the target box and without touching the target box. The robot should remain stationary at the end of its travel, to determine whether the challenge was met or not.

7.3.1 Ideas and suggestions

1) Before starting any activity for the project, record your work in the project log as discussed in the lectures.
2) Use the ultrasonic distance sensor to detect when the box is within an acceptable range.
3) To write your program, use the development branch idea discussed in class to create a function which returns the distance measured by the distance sensor.

7.4 Challenge 3: straight line, arbitrary distance, and arbitrary orientation

For this challenge, the robot will be placed at a distance that can vary between 1 and 2 m, measured from the front of the robot to the target box. The robot is facing an arbitrary direction. The robot should be stationary initially, to allow this placement. To help the robot locate the target, a source of light will be placed on the object. The robot will need to determine the direction towards the target, and then it will need to approach the target within 20 cm from it, but without touching it. The robot should remain stationary at the end of its travel, to determine whether the challenge was met or not.

7.4.1 Ideas and suggestions

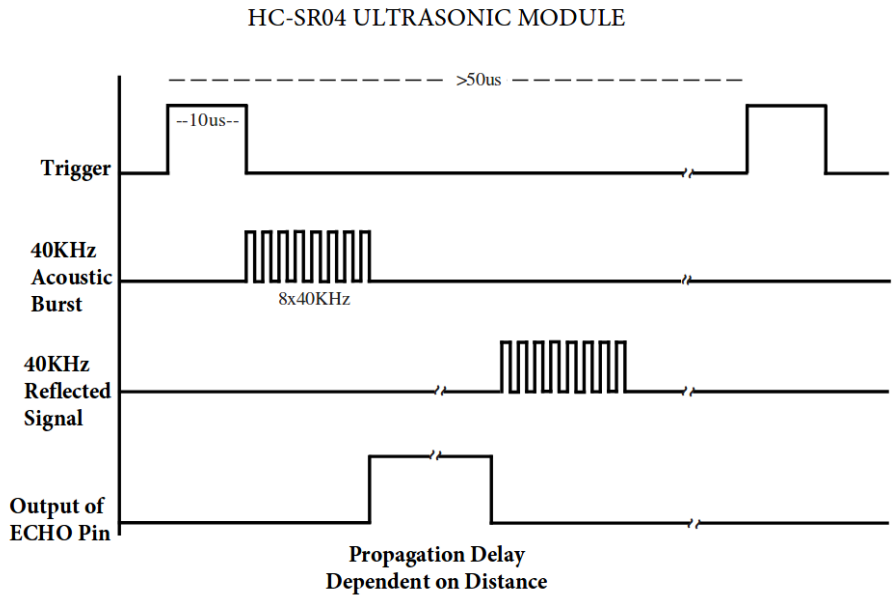
1) Before starting any activity for the project, record your work in the project log as discussed in the lectures.
2) To detect the direction towards the source of light placed on the target, use two light sensors as suggested in Chapter 8.

Chapter 8

Project resources

8.1 Distance sensor

The distance sensor emits a sonic pulse and measures the delay of the echoed pulse to determine the distance to the obstacle that reflected the sonic pulse. The diagram below illustrates the signals used to control the sensor and its output.



Coding:

- To measure the duration of the pulse on the ECHO pin, we can use the function pulseIn(pin, value):

pin	the digital pin where the signal is connected (eg: ECHO)
value	HIGH to measure the duration of the interval the signal is HIGH LOW to measure the duration of the interval the signal is LOW
return value	duration in μs (10^{-6} sec).

- For the speed of sound, you can use 331.5 m/s at 0 deg Celsius. Add 0.6 m/s for each degree Celsius above 0. A typical room temperature is 20 degrees Celsius.
- Program: set the trigger signal like in the figure (make sure the trigger pulse is preceded by a 40 μs period when the signal is low. Immediately after generating the trigger pulse, measure the pulse on the ECHO line. Use the speed of sound to determine the distance.

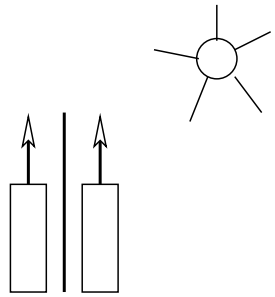


Figure 8.1: Light sensors to detect the direction of light.

– To generate pulses of the order of microseconds, one cannot use the delay() function like in Chapter 2. However, a similar function does exactly what we need: it pauses the execution of the program for thr number of microseconds passed as its argument:
delayMicroseconds(n): will pause the program for n microseconds, where n is an unsigned integer.

8.2 Using the light sensors

Two light sensors can be used to detect the direction for a source of light. Fasten the two sensors together, but separate them by a protruding sheet of paper (see Fig. 8.1). If the sensors receive the same amount to light, it means the source of light is in front of the pack of light sensors. If the source of light is to the left of the pack of sensors, then the sensor on the left will measure a more intense light than the sensor on the right because the sheet of paper creates a shadow. This information can be used to rotate the robot in the appropriate direction until the source of light is in front of the robot.

For your program, consider using the probe/action template, where one of the state variables is the difference between the measurements from the left and the right sensors.

Appendix A

Revision history

v 2.0: Compared to version 1.0, Activity 0 was added and the order of activities was changed to improve the flow.

v 2.1: Exercise 4 from Chapter 4 was added.

v 2.2: The lab activity from Chapter 5 was simplified and the original activity was assigned as an exercise, and new exercises were added.

v 2.3: Exercise 3 on page 25 was revised and simplified.

v 2.4 The code for the activity from Chapter 6 was simplified. Two exercises were added.

v 3.0 Part II completely revised. Project challenges introduced.