LABORATORY EXERCISE 1: INTERFACING OF ANALOG SIGNALS, SENSORS, AND ACTUATORS

OBJECTIVES

Objectives of this laboratory are:

- To provide practical experience of interfacing a variety of signal sources and sensors to the Arduino Mega 2560 using various peripheral boards
- To give further experience of using the Arduino's dialect of the C language
- To provide experience of the interfacing of a servo motor and incremental encoder using an H bridge, the LS7366R up/down counter and the Arduino itself
- To enable students to see in practice the waveforms associated with pulse width modulation and quadrature encoding
- To give a "sneak preview" of the use of finite state machines and of the use of interrupts

SAFETY

In practice this is a low hazard laboratory, but it does involve the rotating shaft on a small motor. To avoid any risk of entrapment, **sleeves should be rolled up and long hair tied back**. No personal protective equipment is required.

UNDERSTAND THE LAB KIT

A labelled picture of the lab exercise setup is shown in Figure 1. The Lab Kit has been developed to make it very easy for you to set it up, and the focus is on understanding the methodology and process. Consequently, all connections between different sensors/buttons/peripherals and the Arduino are facilitated via a custom wiring hereness with mating plugs and sockets. This is different from the "take-home" kits where the skill to develop your own electrical circuit using breadboard and jumper wires is important.

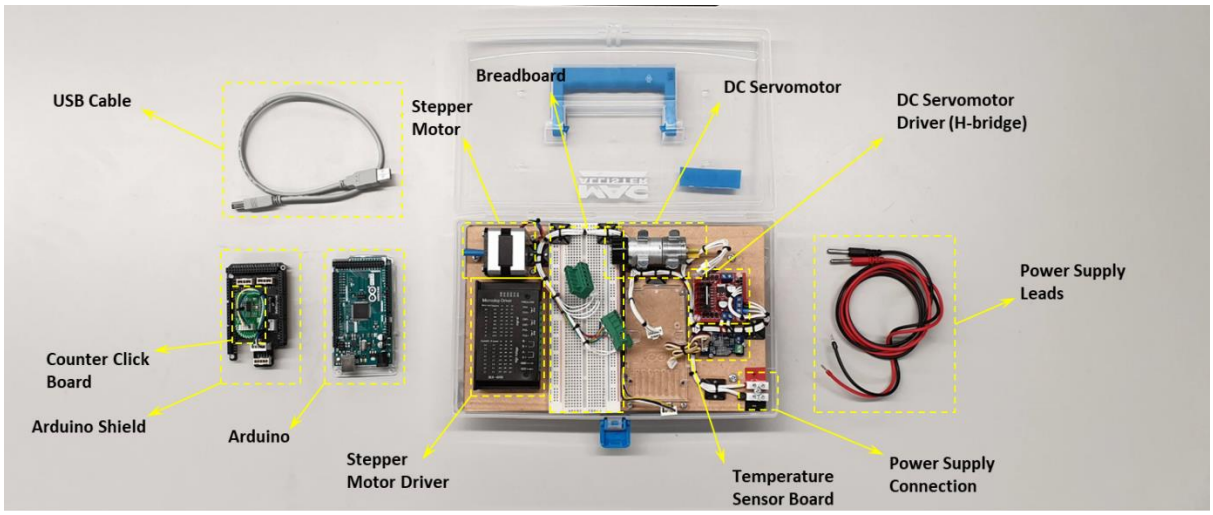


Figure 1 All components in the Lab-Kit

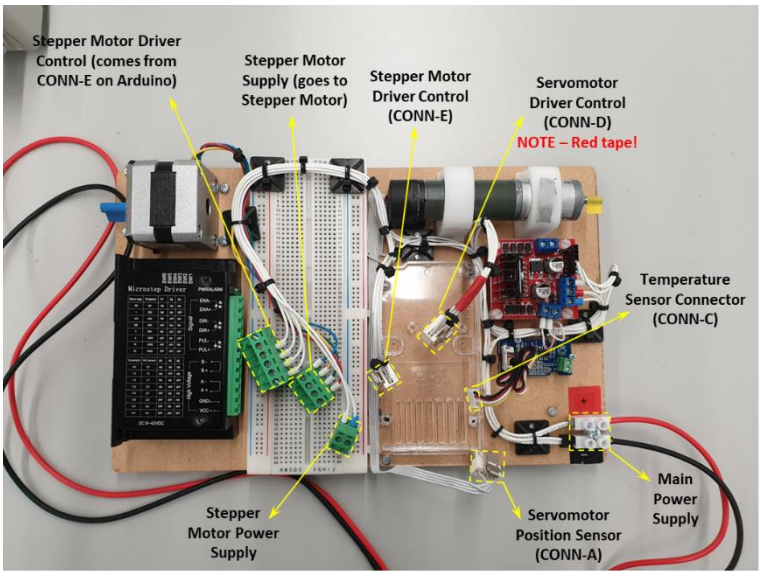
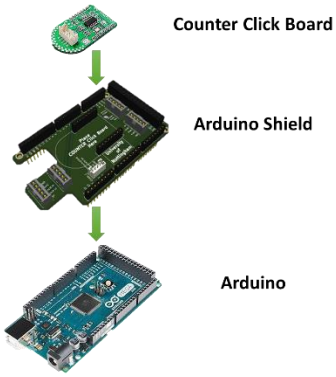


Figure 2 Description of Connectors in the Lab Kit

1. **USB Cable** – Used to connect the Arduino with your laptop or PC.
2. **Arduino** – Microcontroller device that you can program to do various things.
3. **Arduino Shield** – Hardware interface that has been specifically developed to work for these experiments. All connections you require to make with the Arduino (for these experiments) can be made via connectors (no bread board and jumper wires!). The Shield slots on top of the Arduino.
4. **Counter Click Board** – Daughter board required to measure shaft rotation speed. This board slots on top of the Shield.
5. **Stepper Motor** – Motor for applications where precise control of shaft angle position is required.
6. **Stepper Motor Driver** – Produces the signals to drive the stepper motor.
7. **Breadboard** – Prototyping board using jumper wires. You will not require this for your experiments.
8. **DC Servomotor** – General purpose motor for cheap and low power applications where position of the shaft is controlled via closed-loop feedback mechanism.
9. **DC Servomotor Driver (H-Bridge)** – Controls the DC Servomotor. A constant DC voltage across the motor's terminals would also spin the motor, but you cannot control direction or torque produced. The H-Bridge allows that.

10. **Temperature Sensor Board** – Daughter board with a thermistor and connection for a thermocouple.
11. **Power Supply Connection** – DC power supply leads should be screwed in these terminals. Care must be exercised with the polarity (i.e., red lead goes with the red connection point).
12. **Power Supply Leads** – The leads to power the lab kit (using the programmable power supply unit available for the experiments).



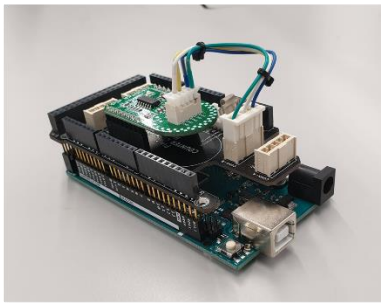


Figure 3 Three boards (Arduino + Arduino Shield + Counter Click Board) together

UNDERSTAND AN OSCILLOSCOPE

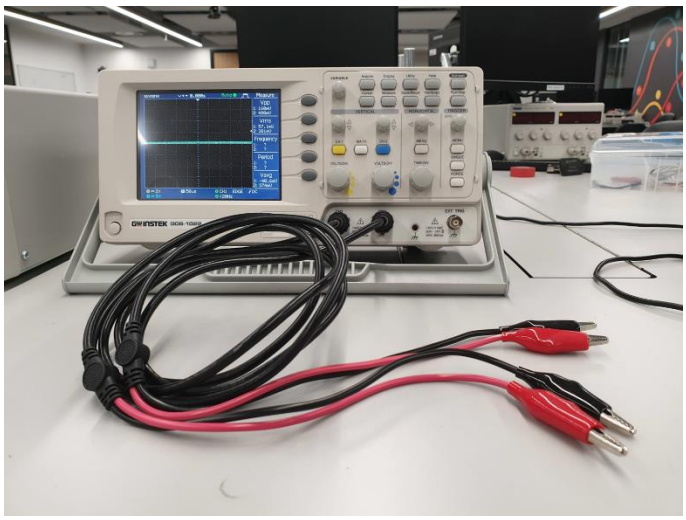


Figure 4 GW-INSTEK GDS-1022 Digital Storage Oscilloscope

WHAT IS IT?

An oscilloscope is an instrument to measure, visualise and record a voltage signal. It has a probe that is required to be connected to the source of voltage (it may be the output of an electronic circuit you have designed or something as simple as an AA-size battery!) and the oscilloscope "set" to read the signal. Usually there are multiple channels (or signals to probe) that can be visualised simultaneously, and some simple mathematical functions done live. The speciality of an oscilloscope (compared to other similar voltage measurement devices like a digital multimeter) is its ability to visualise the signal over a time duration (not just an instantaneous measurement) and record it on a USB stick.

WHY DO YOU NEED IT?

Unlike engineers from other disciplines, electrical and electronic engineers cannot "see" the output of their physical circuits and systems (needless to say, its crucial to be able to see what you have created, is it behaving how you expect it to and if not, what bit needs modifying). An oscilloscope usually goes hand in hand with the design and build of any circuit as it allows you to visualise and record the input and output signals.

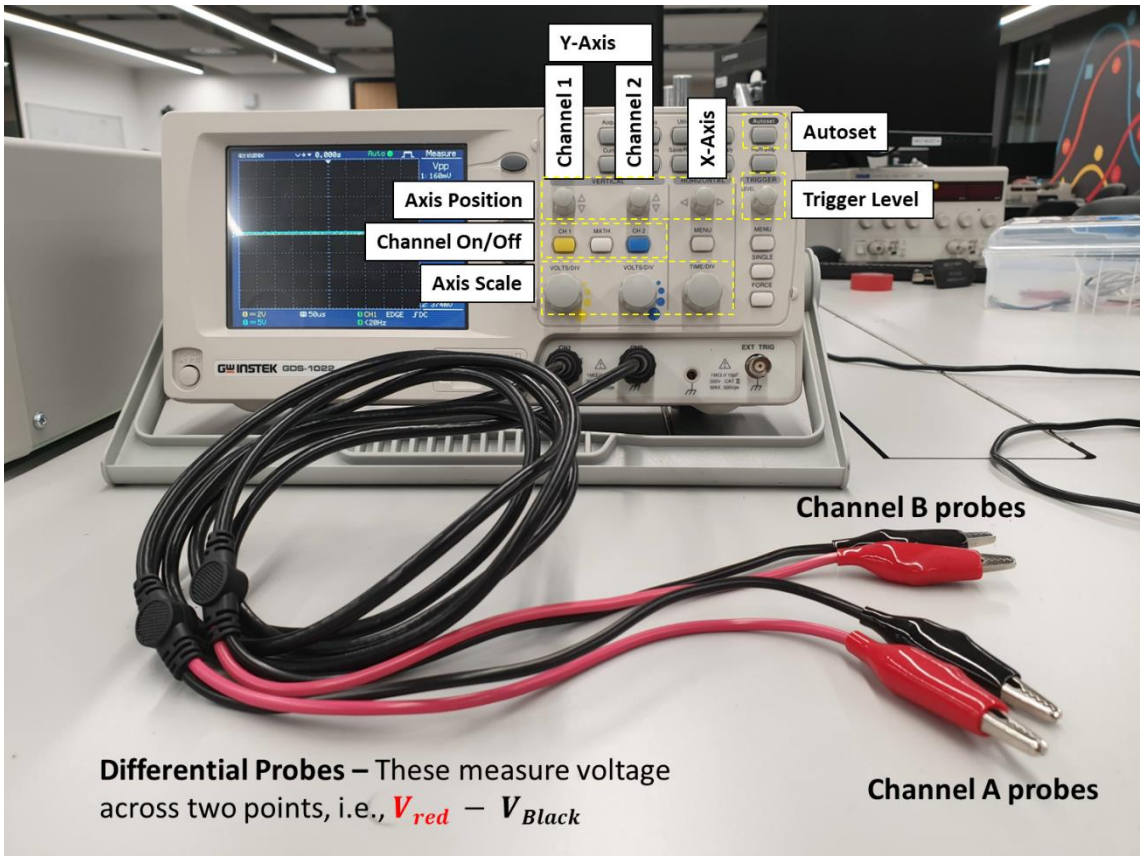


Figure 5 Details of an oscilloscope

THE AXES – HORIZONTAL (TIME) AND VERTICAL (VOLTAGE AMPLITUDE)

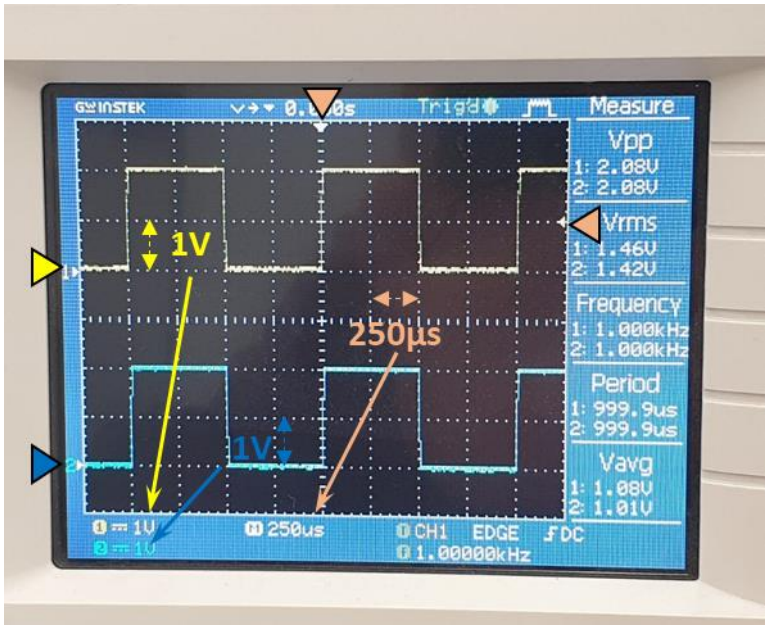


Figure 6 Standard square wave shown on both channels

The information bar at the bottom gives essential information to understand the magnitudes of the visualised signal (Figure 6). The screen is divided in 10 divisions on both axes (illustration not accurate) and the 1st number depicts the signal voltage per sub-division. The second number depicts the time per sub-division, i.e.,

the entire axis is 10x of this value. The third number is the frequency of triggering (read more about triggering in the next section) of the signal and can be interpreted as the frequency of the signal itself.

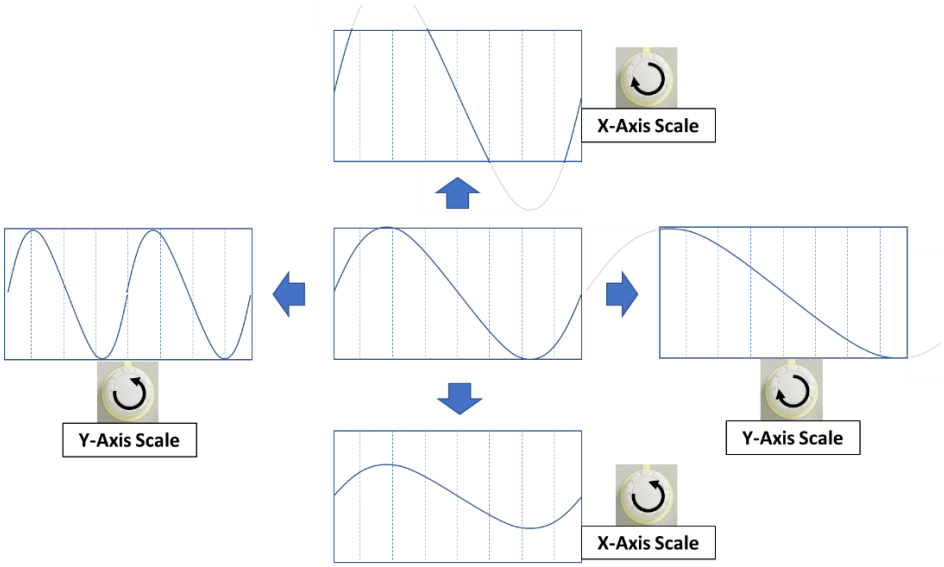


Figure 7 "Scale" functionality (Large knobs) for two channels (Y Axes) and time (X Axis)

The dial knobs called SCALE (both axes) are used to make the visualised signal smaller or larger on the display area (Figure 7). The two channels have their individual Y-Axis scaling dials, but they are both on the same X-Axis, i.e., time axis, and hence there is only one X-Axis scale dial. This means both channels would scale on the time-axis.

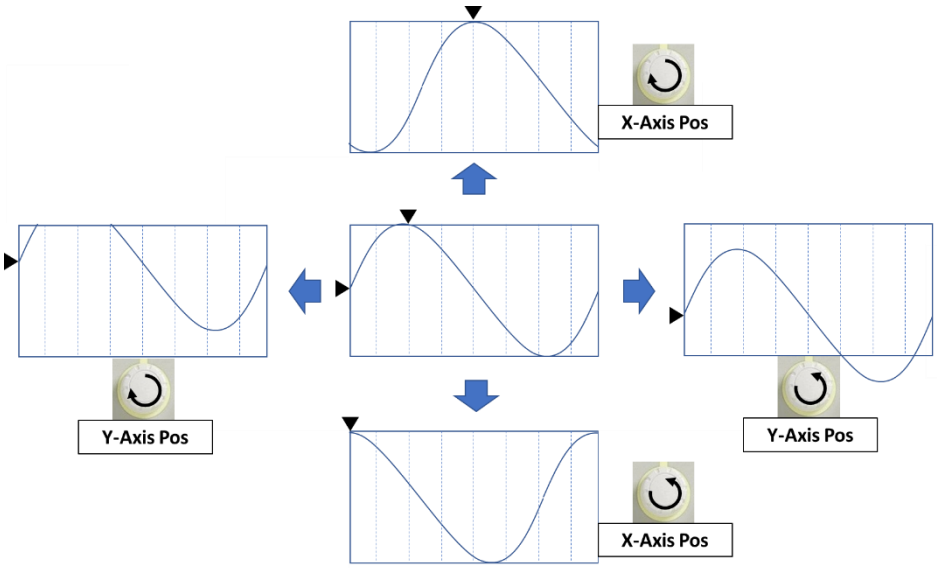


Figure 8 "Position" functionality (Small knobs) for two channels (Y Axes) and time (X Axis)

POSITION dials can be used the move the visualised signal up, down, left, or right. This is very useful when you want to compare two channels visually, you can align them as you want or overlap them on top of each other.

TRIGGERING

This is a bit complicated to understand in the first go, but once you have grasped the concept it is very straightforward. The way an oscilloscope works, it rapidly clicks “snapshots” of the signal (for the entire duration as you have set up using the X-Axes). Now we need to tell the oscilloscope at what instances should it click these snapshots. This is done using the TRIGGER Level and Menu functions. You can tell the oscilloscope to continually detect for “Falling Edge” (selected from the Trigger Menu) at a particular trigger voltage “Level” (set using the dial), e.g., 0V.

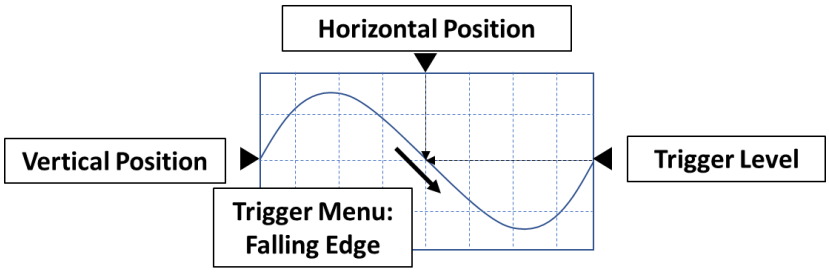


Figure 9 "Triggering" functionality

AUTOSET BUTTON

This is a very useful function; it automatically sets all the above settings to produce a usable display of the input signals.

EXERCISE 1 – INTERFACING WITH DIFFERENT SENSORS

1. **Go to the last page of this document and read all the questions you need to answer for the coursework submission. Keep them in the back of your mind while performing the exercises.**
2. Have a good look at the Arduino Shield (see Figure 10). Notice there are 5 sockets on this board (we will go through which connector to use for each socket as we go through the experiment):
    a. CONN-A – This connects to the DC servomotor (position sensor – encoder)
    b. CONN-B – This connects to the Counter Click Board
    c. CONN-C – This connects to the Temperature Sensor Board
    d. CONN-D – This connects to the DC servomotor Driver (H-Bridge)
    e. CONN-E – This connects to the Stepper Motor Driver

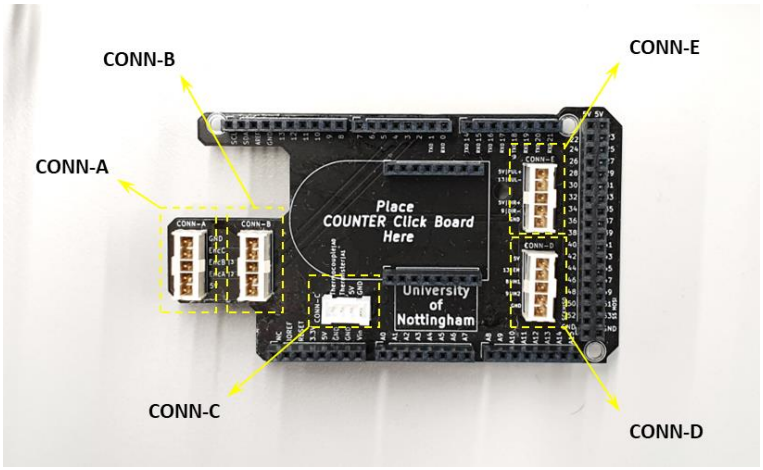


Figure 10 All sockets in the Arduino Shield (CONN-A to CONN-E – relates to Figure 2)

3. Connect the **Arduino Shield** with the **Arduino** (see Figure 11 – **IF IT IS ALREADY DONE, SKIP THIS STEP**):
    a. Align the appropriate pins (on the bottom of Arduino Shield) and sockets (on the top of Arduino)
    b. Press them down lightly and steadily, do not force
    c. Do not press the pins all the way into the sockets (see "after" picture below)
    d. Please exercise utmost caution in this process, you may accidentally bend a pin if all pins are not aligned properly!
    e. **Please seek a demonstrator to check this if you find resistance while connecting them together.**

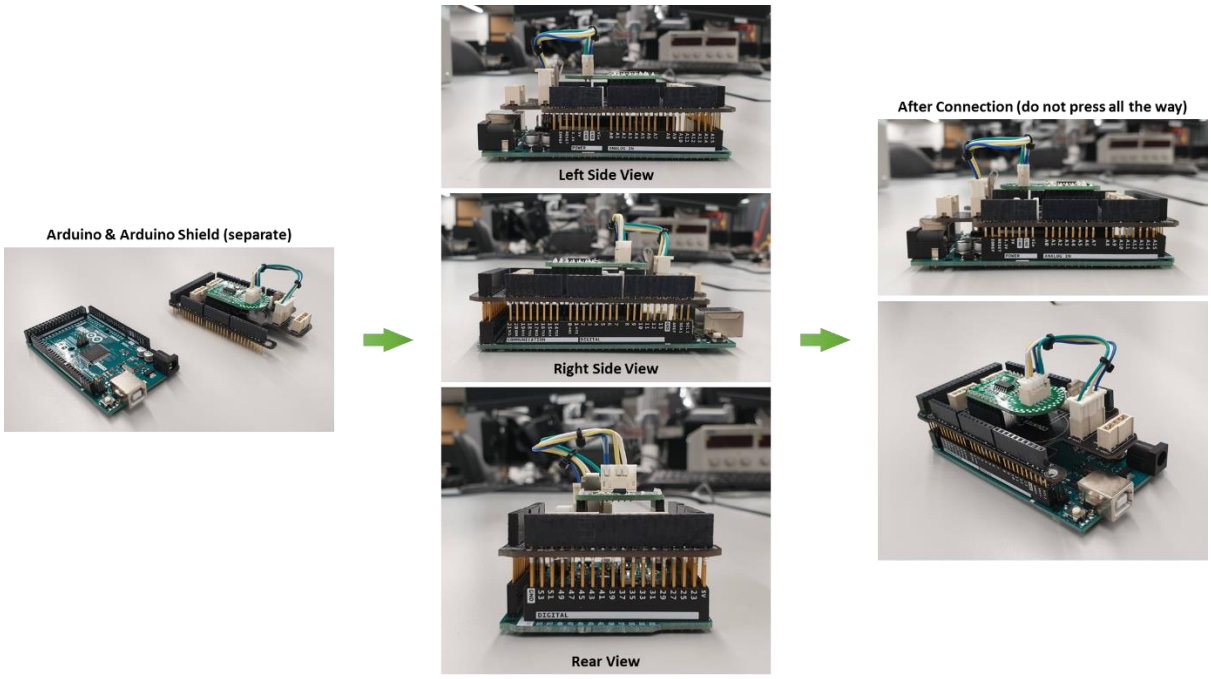


Figure 11 Steps to connect the Arduino and Arduino Shield

4. Place the Arduino in the middle of the lab kit (in between the breadboard and DC Motor Driver).
5. Using the small screwdriver provided in your experiment kit, wire up the thermocouple to the temperature sensor board, ensuring that the **red lead** is connected to **+** and the **white lead** to **–** on the green terminal block. **Take care not to over-tighten** as the terminal block and its attachment to the board are quite fragile.
6. Connect the multicolour 4-wire cable (coming from the Temperature Sensor Board – see picture below) connector to CONN-C on the Arduino Shield. Do not connect anything else (CONN-B may be left connected to the Counter Click Board).

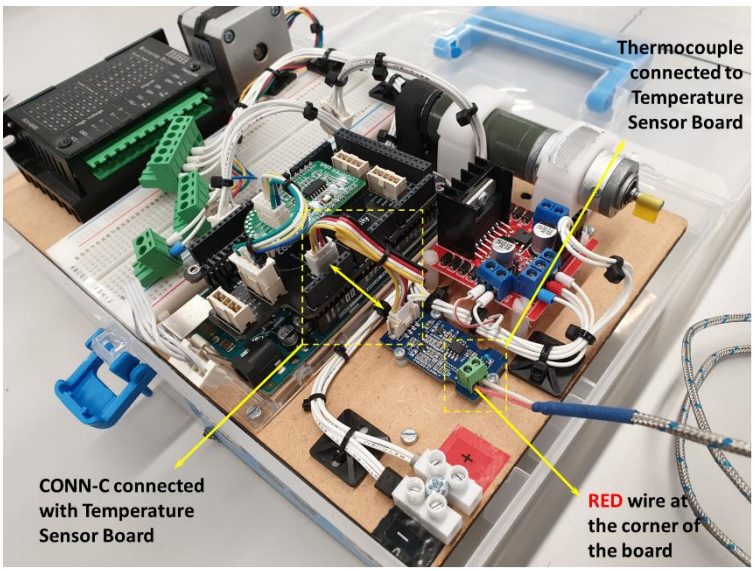


Figure 12 Hardware setup for Lab-1 Exercise-1

7. **Get a demonstrator to check your wiring!** It should look like Figure 12.

8. Connect the Arduino to your PC/Laptop using the USB cable. You should see the light appear on the Arduino. This indicates there is 5V power from the USB.
9. Load the program you have modified from **TwoSensorsSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions. **If you have not been able to get a working code – seek a demonstrator and ask for the solution code.**
10. Ensure that the correct port and board type are set by checking **Tool | Board** and **Tools | Port**.
11. Now compile and upload your program. Open the Serial Monitor from the Tools menu. You should see the room temperature; the thermocouple temperature and the displacement displayed every second.

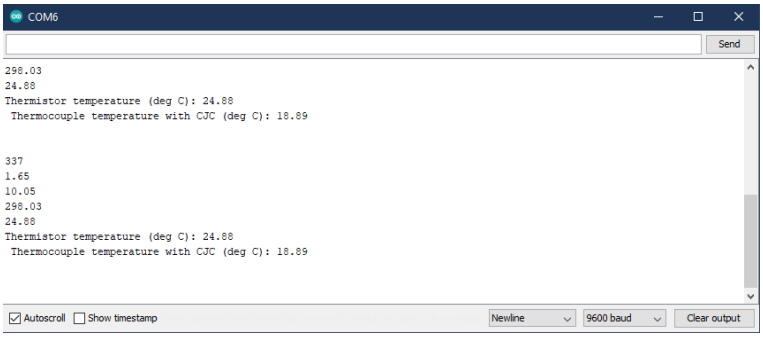


Figure 13 Serial Monitor view for Lab-1 Exercise-1

12. You can test the sensor by placing a finger on the thermistor (it is the blue "bulb" like feature on the Temperature Sensor Board – see Figure 14). You should see the temperature rise on the Serial Monitor.

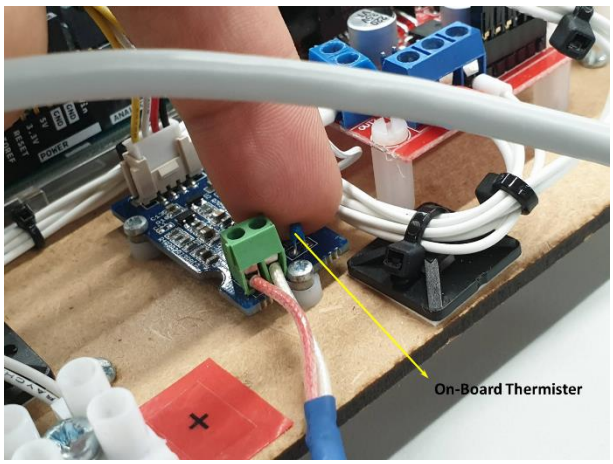


Figure 14 Location of on-board thermistor on Temperature Sensor Board (and how you can test it by placing your finger)

13. You may test the thermocouple by placing it near the exhaust vent of your PC/Laptop (see Figure 15).

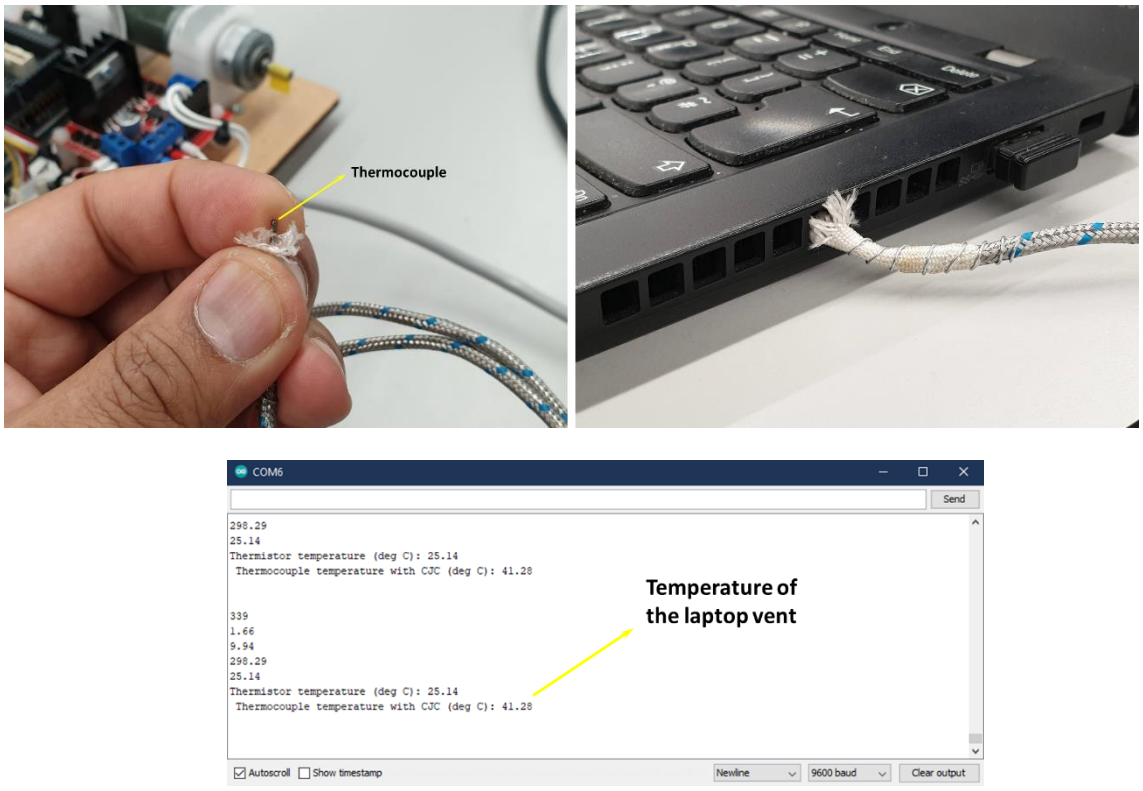


Figure 15 Thermocouple testing by placing it near PC/Laptop vent and display temperature rise in Serial Monitor

14. When you have finished, disconnect the Arduino from the PC.
15. Disconnect CONN-C on the Arduino Shield, Arduino USB cable from the PC/laptop, and the thermocouple from the Temperature Sensor Board.

EXERCISE 2 – PULSE WIDTH MODULATION: INTERFACING WITH A SERVOMOTOR

DC Servomotors of different types are widely used for positioning, for instance within machine tools and robotics. Unlike stepper motors, which (if desynchronisation and resonance do not occur) will move under open-loop control to a defined position and will stay there, servomotors rely on closed-loop control to achieve the desired position:

- Actual position is measured continuously and compared with the desired position.
- The difference (error) is used via a control algorithm (decision-making process) to determine the corrective action, typically in the form of a voltage or current which is fed into the motor.
- The current flowing through the motor results in a torque which causes the motor to move to correct for the error.

This process is repeated many times a second to achieve rapid and stable response to changes in the required position.

The interface to a servomotor requires the following:

- A means of varying the voltage and/or current fed to the motor – in the present case, this is achieved using pulse width modulation, which (conveniently) is the form of output provided by the Arduino when the loosely-named **`analogWrite()`** function is used.
- A means of measuring the position – in the present case, this is achieved using an incremental encoder – this will be examined in Exercise 3, "Quadrature Encoder Decoded".

Within this lab, you will examine the forms of the signal going to and from the motor from this module under open-loop conditions – within lab 2 we will explore the motor's closed-loop behaviour in more detail.

The lab itself involves driving the motor from an L298N H-bridge driver IC ("DC Servomotor Driver"), which in turn is powered from a power supply.

1. **Go to the last page of this document and read all the questions you need to answer for the coursework submission. Keep them in the back of your mind while performing the exercises.**
2. Ensure the large bench power supply unit (PSU) is connected to the mains.
3. Switch the PSU on and ensure the ON/OFF button is in OFF. When it is OFF, the screen displays the set-point voltage and current limits (see Figure 16).

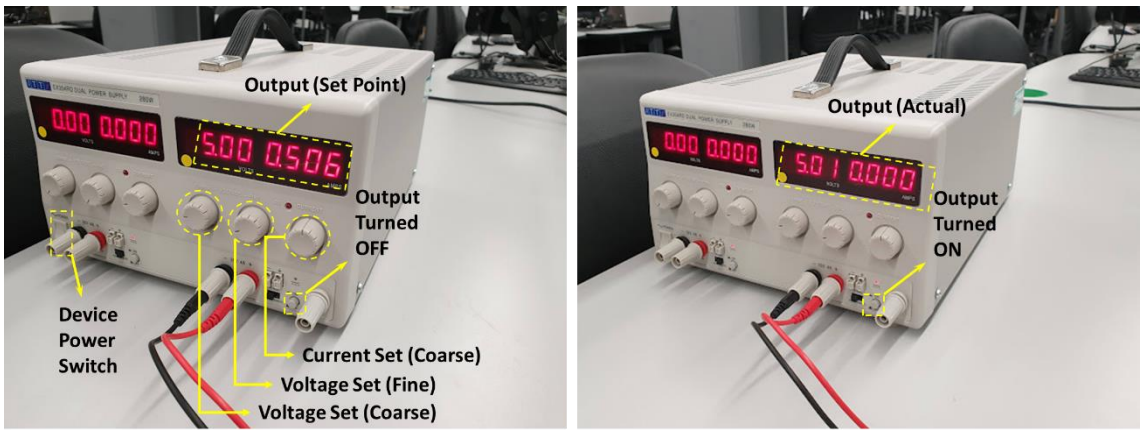


Figure 16 Using the Power Supply Device

4. Adjust the voltage to read 5V and the current limiter to read ~0.5 A. Use the three knobs:

a. Voltage Set Point – Coarse (Left)
b. Voltage Set Point – Fine (Middle)
c. Current Limit Set Point – Coarse (Right)

5. Turn ON the supply. The displays now read the actual voltage and current. The voltage should remain around 5V, but the current should be approximately zero as no load is connected.
6. Turn OFF the output and switch off the device.
7. Ensure the Arduino is disconnected from the PC.
8. Make the following connections:
    a. **CONN-D** (on Arduino Shield) with the "**Servomotor Driver Control**" connector (see Figure 2 a few pages above – this is the plug/connector with a red tape wrapped around the cables)
    b. **CONN-A** (on Arduino Shield) with "**Servomotor Position Sensor**" connector (see Figure 2 a few pages above)
    c. **CONN-B** (on Arduino Shield) with "**Counter Click Board**" connector (see Figure 3 a few pages above)
    d. **Main Power Supply** (bottom right of the lab kit base board) using the **Power Supply Leads** (see Figure 1 and Figure 2 a few pages above – ensure polarity is correct, i.e., **Red** with **Red**, and **Black** with **Black** – see Figure 17)

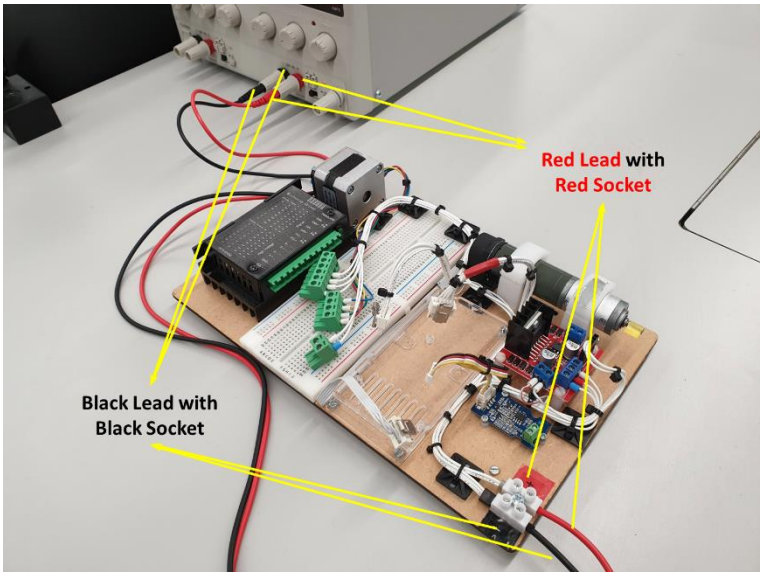


Figure 17 Polarity matching when connecting Power Supply

9. **Get a demonstrator to check your wiring!** It should look like Figure 18.

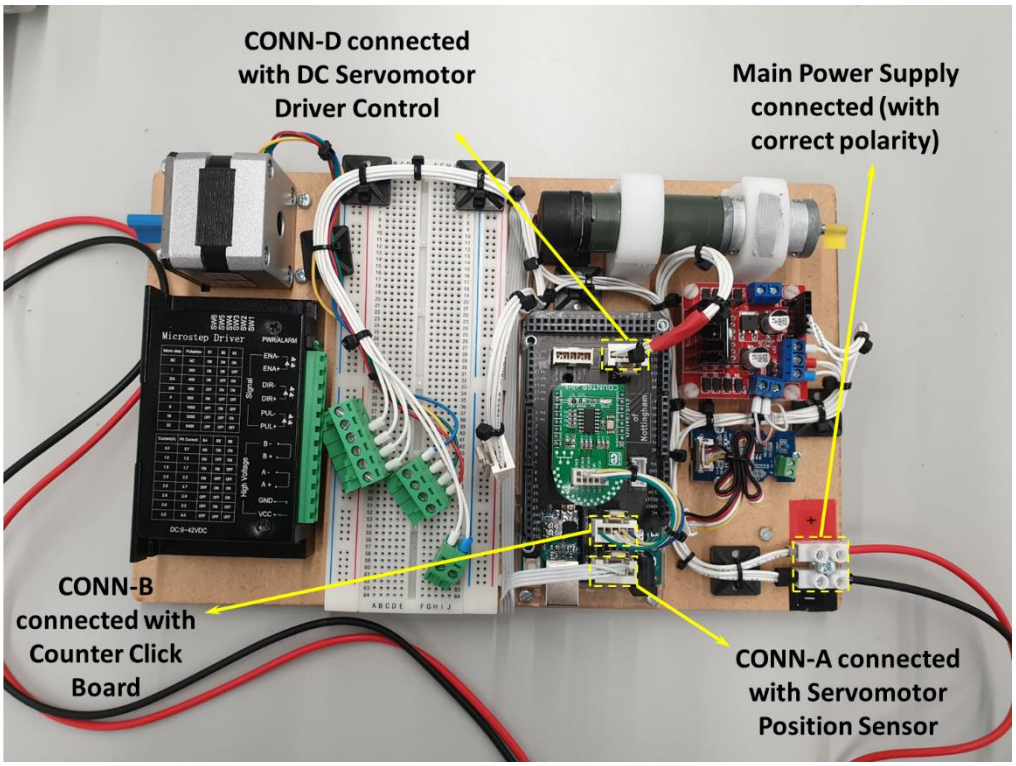


Figure 18 Hardware setup for Lab-1 Exercise-2

10. Connect the Arduino to your PC/Laptop using the USB cable. You should see the light appear on the Arduino. This indicates there is 5V power from the USB.
11. Turn ON the PSU and turn ON its output. Ensure you start at 5V.
12. Load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions. **If you have not been able to get a working code – seek a demonstrator and ask for the solution code.**
13. Ensure that the correct port and board type are set by checking **Tool | Board** and **Tools | Port**.
14. Now compile and upload your program. Open the Serial Monitor and check the box in the bottom right-hand corner of the monitor reads "Both NL and CR" to ensure that your program will understand when a line of text has been sent.
15. Type different values of PWM duty cycle percentage in the range -100 to 100 into the input field. The motor should run forward or backwards for positive and negative values. **You now have a working DC Motor with variable torque and interchangeable direction!**
16. It would be very interesting to see the voltage waveform output from the H-Bridge (Motor Driver). In order to do this, we shall use an Oscilloscope! Please revisit Figure 4 to Figure 9 above to understand how an oscilloscope works.
17. Make the following connections to read the signal on the oscilloscope:
    a. **OUT1** on the H-Bridge (bottom edge of the Driver Board to the left) with **CH1 Signal** (**Red** crocodile clip) via a jumper cable
    b. **OUT2** on the H-Bridge (bottom edge of the Driver Board to the left) with **CH1 Reference** (**Black** crocodile clip) via a jumper cable
18. Ensure that the scope is switched on and set to 5V/div sensitivity on Channels 1 and 2 and 0.5 ms/div on the time base.
19. **Get a demonstrator to check your wiring! The final setup should look like Figure 19. (If you do not understand how to operate the oscilloscope, ask a demonstrator).**

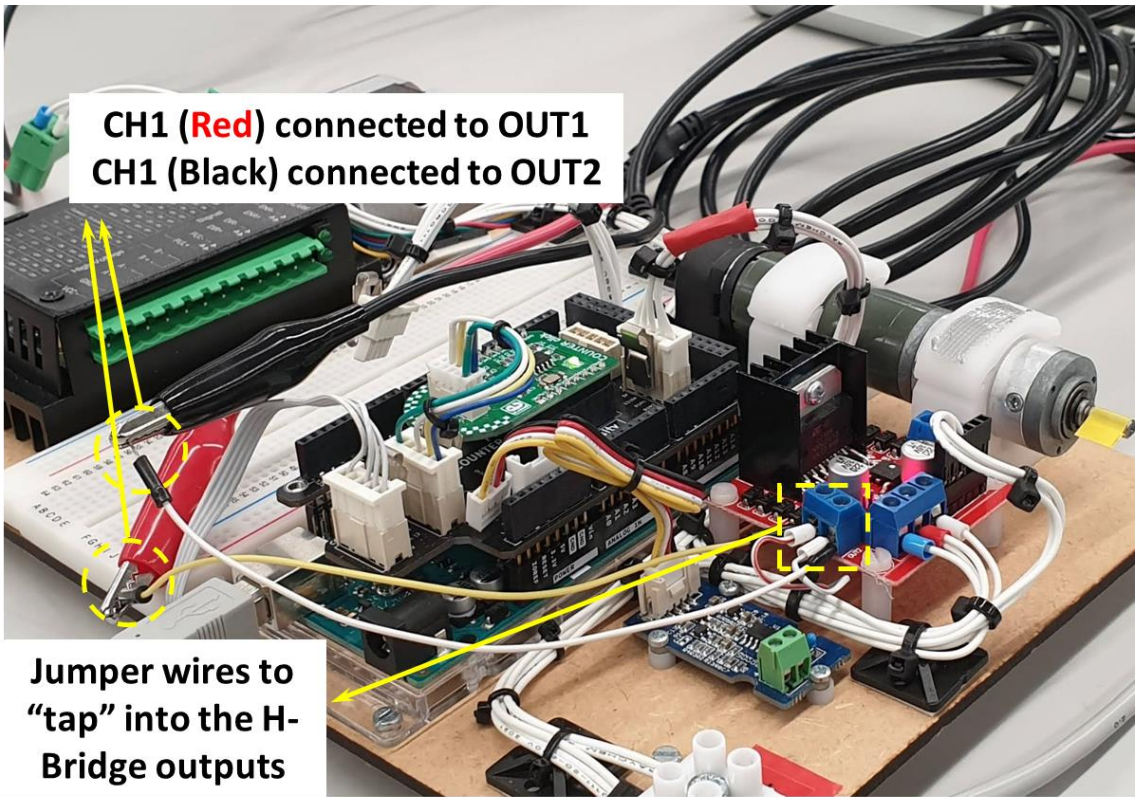


Figure 19 Hardware setup for Lab-1 Exercise-2 (after probing)

20. Connect the Arduino to your PC/Laptop using the USB cable. You should see the light appear on the Arduino. This indicates there is 5V power from the USB.
21. Load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions. **If you have not been able to get a working code – seek a demonstrator and ask for the solution code.**
22. Switch the PSU on and ensure it is activated by pressing the small button to lock it in. Ensure you start at 5V.
23. Open the Serial Monitor and ensure that the box in the bottom right-hand corner of the monitor reads **"Both NL and CR**" to ensure that your program will understand when a line of text has been sent.
24. Type different values of PWM duty cycle percentage in the range -100 to 100 into the input field. The motor should run forward or backwards for positive and negative values. Sketch carefully the waveforms on channels 1 and 2 of the scope (which are the signals seen by each side of the motor with respect to ground) and the math trace (which is the potential difference across the motor) for (say) 60% and -60%. You will need to draw these neatly on your answer.
25. Now increase the PSU voltage up to 15V for higher speed. **Do not go higher than that, you may blow up the H-bridge.** Observe the traces again.
26. When finished, close the monitor, switch off the PSU and disconnect the Arduino but do not disconnect anything else.

EXERCISE 3 – QUADRATURE ENCODER DECODED

After having set up the DC motor, the "servo" functionality (i.e., using closed-loop circuit to control the angular position of the motor shaft is called servomotor) shall be setup in this exercise. Notice that the DC motor has a motor position sensor (called an "encoder") attached to its non-shaft end. This sensor provides the angular position of the motor shaft that can be used to "close the loop".

1. **Go to the last page of this document and read all the questions you need to answer for the coursework submission. Keep them in the back of your mind while performing the exercises.**
2. Ensure the power supply is switched off.
3. You should already have CONN-B connected to the Counter Click Board connector and CONN-A connected to the "Servomotor Position Sensor" connector (see Figure 18 from previous exercise).
4. If all the connectors (CONN-A, CONN-B, and CONN-D) are connected properly, the following logic connections have been made:
    a. **Encoder Channel A** connected to **Counter Click Board** and **Arduino pin 47 (timer/counter input pin T5)**
    b. **Encoder Channel B** connected to **Counter Click Board**
    c. **Counter Click Board 5V** supply and **ground**
    d. **Counter Click Board SCK (system clock)** connected to Arduino pin 52
    e. **Counter Click Board CS (chip select)** connected to Arduino pin 10
    f. **Counter Click Board SDI (slave data input)** connected to **Arduino pin 51 (MOSI – Master Output Slave Input)**
    g. **Counter Click Board SDO (slave data output)** connected to Arduino pin 50 (**MISO – Master Input Slave Output)**

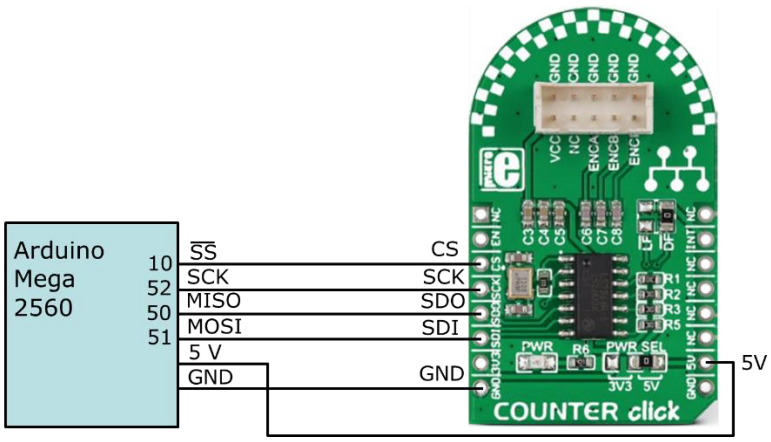


Figure 20 Counter Click Board logic connections (already implemented in the Arduino Shield, completed by plugging in CONN-A, CONN-B, and CONN-D)

5. In your program, you would be reading the position/speed of the motor shaft in three ways:
    a. Using the Counter Click Board (reading via the serial communication – CLK, CS, SDI, SDO)
    b. Using Arduino Timer to count Encoder Channel A pulses
    c. Using your State Machine code
6. Make the following connections to read the signal on the oscilloscope (please see Figure 21):
    a. **Pin 4 (Encoder Channel A)** on CONN-B (yellow cable) with **CH1 Signal** (**Red** crocodile clip) via a jumper cable
    b. **Pin 3 (Encoder Channel B)** on CONN-B (white cable) with **CH2 Signal** (**Red** crocodile clip) via a jumper cable

c. **Pin 5 (Encoder Ground)** on CONN-B (green cable) with **CH1** and **CH2** Reference (**Black** crocodile clip) via a jumper cable

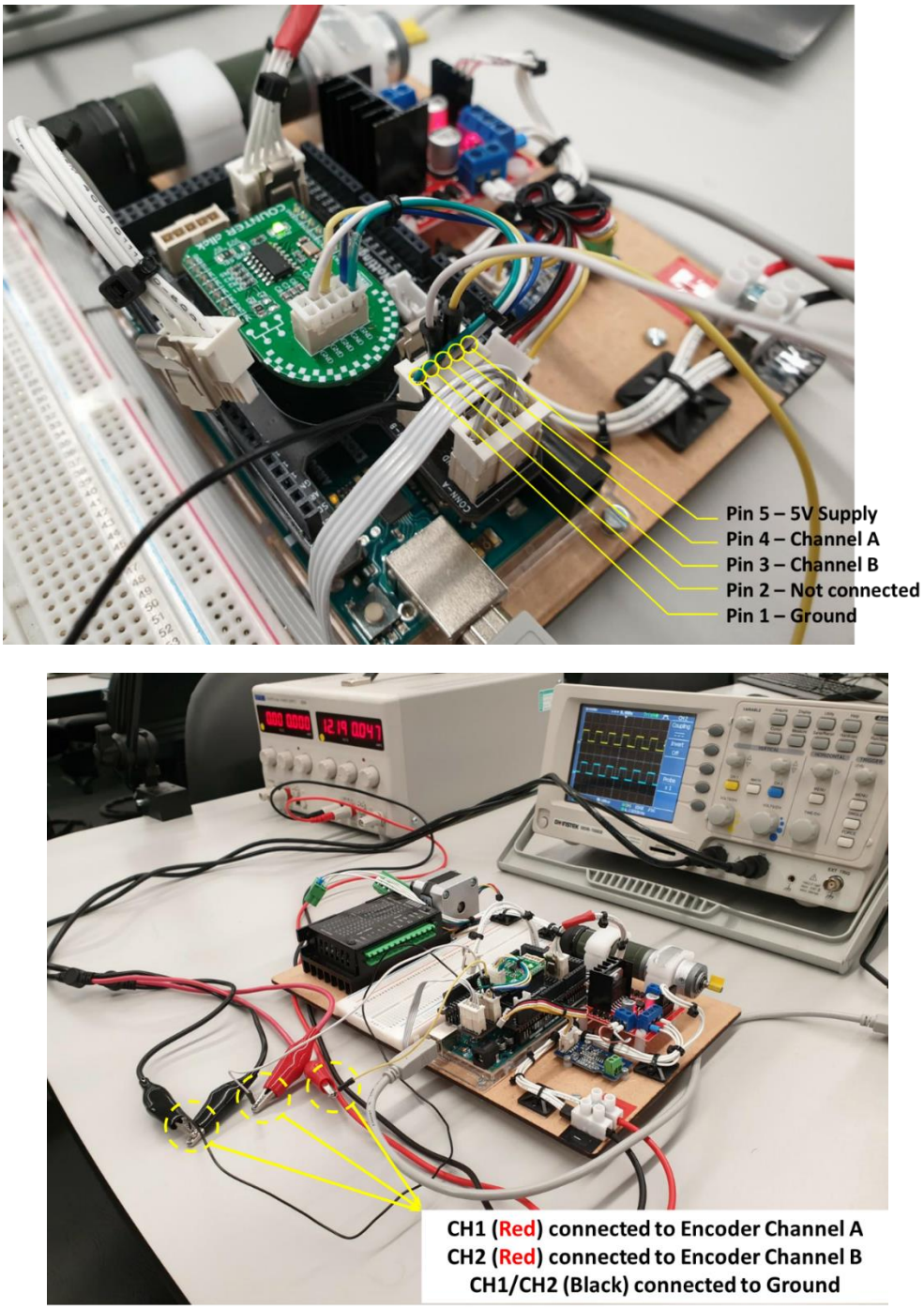


Figure 21 Hardware setup for Lab-1 Exercise-3 (after probing)

7. **Get a demonstrator to check your wiring to avoid expensive damage!**
8. Connect the Arduino to your PC/Laptop using the USB cable. You should see the light appear on the Arduino. This indicates there is 5V power from the USB.
9. You should already have the working code loaded onto your Arduino from the previous exercise. If not, load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it

should have already been verified in your programming sessions. **If you have not been able to get a working code – seek a demonstrator and ask for the solution code.**

10. **Enter values 100** (to run motor fully forward) **or -100** (fully in reverse) and see whether the encoder counts recorded by your state machine program agree with the values from the LS7366R hardware counter on the Counter Click board. **Do you get any errors recorded?** Note that your state machine code "polls" (looks at) the encoder inputs every time the `loop()` code executes, which is at a rate determined by software. **Can you identify any reason why the microprocessor might miss long sequences of encoder pulses even though not many errors are recorded?**
11. You should also note that the internal counter (using Timer 5, counting pulses received on pin 47) on the Arduino is giving a count value that should be a quarter of the value from the hardware counter. **Does that behave as expected at high-speed running?**
12. Carefully observe the way the numbers are changing when the motor is running forward and when it is running backward. **What is different between the two?**
13. How does the error look like at higher speeds? Is it different compared with lower speeds?
14. **BONUS:** Now comment out the line `updateEncoderStateMachine()` in `loop()` and uncomment the following two lines in `setup()`:

```
attachInterrupt(digitalPinToInterrupt(channelA),
updateEncoderStateMachine, CHANGE;

attachInterrupt(digitalPinToInterrupt(channelB),
updateEncoderStateMachine, CHANGE;
```

15. These call the `updateEncoderStateMachine()` function whenever there is a change of state on channel A or channel B from the encoder – in other words, exactly when (and only when) the state needs to change, regardless of anything else the Arduino is doing. Test this with motor PWM values up to (say) 30%. Then try at faster and faster speeds**. Do you get any errors? Does something unexpected happen at higher speeds? Why?**
16. When you have completed the experiment, please proceed with the **shutdown procedure**:
    a. Disconnect the Arduino USB cable to the PC/laptop (this removes 5V supply and immediately mitigates against any accidental shorting).
    b. Switch off the PSU, disconnect the power supply leads from the lab kit using the screwdriver.
    c. Disconnect all jumper wires you used for oscilloscope probing.
    d. Disconnect all the connectors that you plugged in on the Arduino Shield (CONN-A to CONN-E).
    e. Remove the Arduino from the Arduino Shield (exercise caution in this step, you may accidentally bend some pins if you are not careful!).
    f. Replace all components in the Lab Kit container as you found it.
    g. Be careful when shutting the lid of the container, make sure no cable gets trapped.

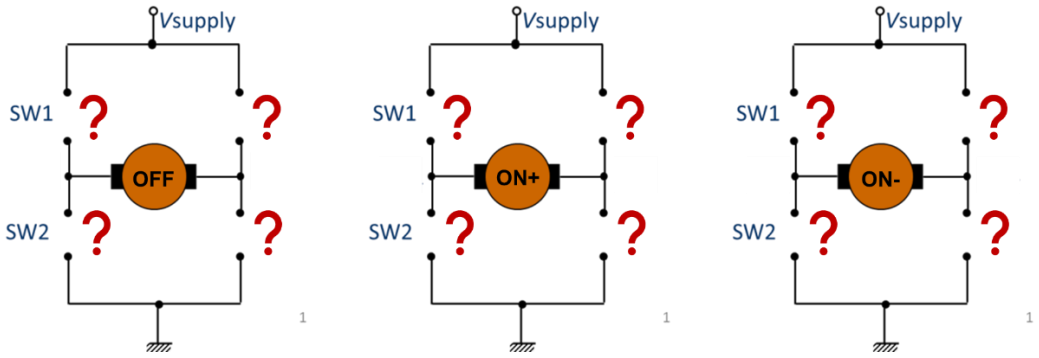
Hope you enjoyed the session! Please work on your lab exercise and submit on Thursday 16th November by 1500 hours

COURSEWORK SUBMISSION

Well formatted and organised submission: A zip file containing the answers to your questions, ideally as a PDF, including listings of the programs you have written and submitted as preparatory work, including any corrections made in class. Also please include in the zip file all the programs themselves as Arduino source (.ino) files. **[20 marks].**

Questions (please limit each answer to a single side of an A4 page – maximum size of submission should not be more than 6 pages):

1. The temperature sensor kit (thermocouple and thermistor daughter board) has a 3.3V amplifier, i.e., the output voltage is between 0 and 3.3V for the complete range of detected temperature (equation 4 in the programming lab sheet). Why do we still use 5V as the *Vref* when we convert the ADC-converted value back to it analog value (equation 1 in the programming lab sheet)? Why is this not the most optimum use of the ADC pin and how can we increase the resolution of the detected signal? **[10 marks].**
2. Paste camera clicks of the PWM waveform (oscilloscope readings) as you observed in exercise 2 for the following duty cycle settings: 10%, 25%, 50%, 75%, 100%. What is the distinction between 100% PWM and all others? Clearly mark which region is the ON region and which region is the OFF region. Why is the voltage not zero in the OFF region? **[10 marks].**
3. Sketch the switch configuration for ON and OFF regions in the H-bridge circuit diagram below. What is the difference between ON (positive) and ON (negative)? What is the PWM frequency by default in an Arduino? What is the frequency of the audible note that we hear? Please provide evidence through the oscilloscope readings of the PWM waveform. **[10 marks].**



4. Paste camera clicks of the quadrature signals (oscilloscope), when running in the positive and negative directions. Remember to mark the zero voltage datum marks. **[10 marks].**
5. Although it only counts a quarter of the pulses (as it only looks at rising edges of one pulse train, not all four transitions per cycle) the internal counter (Timer 5 configured to count pulses on pin 47) seems very reliable, at least under some circumstances. Could this be used as an alternative to the LS7366R quadrature decoder? Explain your reasoning. **[10 marks].**
6. When you implemented your quadrature decoder coding to run as software on the Arduino, what shortcoming did you observe? Can you identify what the program might be doing when this problem occurs? **[10 marks]**