Stirling Engine Lab 2023

Welcome to the Stirling Engine Lab! This segment attempts to integrate concepts from different modules you will undertake during your following years to contextualise the real-world usefulness of the theory you will learn. You may be coming across some of these concepts for the first time, but fret not, much following are meant as a surface level introduction and are not examined (yet…). So take this as a gentle romp in the wonderful world of engineering, ask lots of questions, and most importantly, have fun!

Contents

Theoretical Background

Thermodynamics

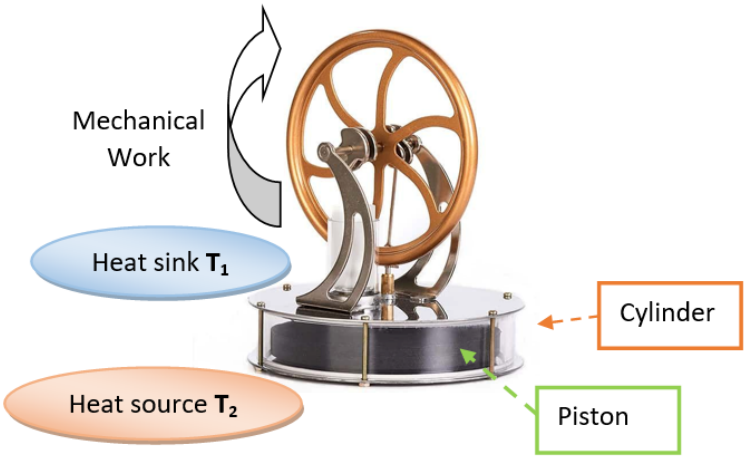


Figure 1. Thermodynamic principles of a Stirling engine. [1]

A Stirling Engine is a heat engine that converts thermal energy to mechanical energy in the form of the rotating motion of the flywheel. The thermal energy comes from a heat source T_2 placed under the cylinder, and is driven by the temperature differential with a heat sink T_1 on top of the cylinder (in this case atmospheric air). It operates on a closed cycle, meaning the working fluid remains contained in the cylinder throughout the entire thermodynamic cycle. We can monitor the effect of the temperature differential on the efficiency of the engine by measuring the RPM of the flywheel. This is achieved by taking measurements from transducers, or sensors as the engine is running.

Instrumentation

Sensors

The sensors used in this setup are: 2 x **TMP36 temperature sensors** to measure temperature at the top and bottom plates of the cylinder, and 1 **infrared (IR) beam sensor**.

The temperature sensor is made of a resistive element whose resistance is a function of temperature: it is connected in a potential divider circuit and the voltage output is measured. A conversion is required to change this to a temperature reading.

The IR sensor consists of an IR emitter and receiver. The receiver normally receives IR light from the emitter, but when something passes between them (like spoke on the flywheel), the beam is broken and this can be registered on a computer as a signal.

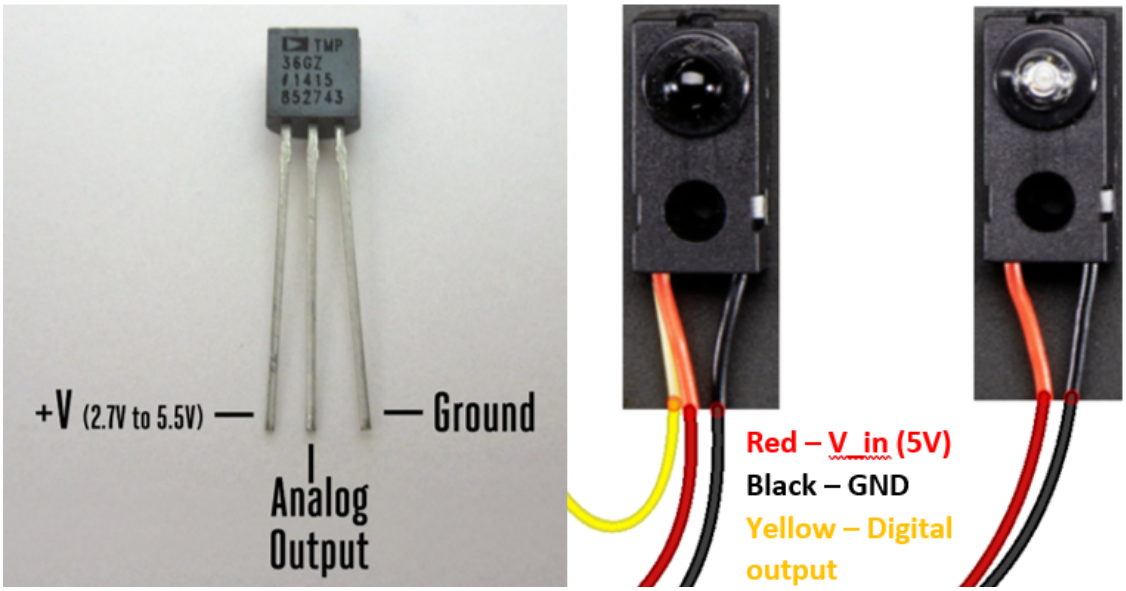


Figure 2. TMP36 sensor and IR beam break sensor consisting of receiver (left) and emitter (right) [2] [3]

Controller

The sensors are connected to an **Arduino Uno**, which is known as a microcontroller and acts as the 'brains' of the operation. It provides input/output (I/O) connections for these sensors so that your personal computer can read/write data to the sensors, and is effectively a mini-computer itself.

Arduinos are widely used in hobby control systems, such as remote control (RC) cars and drones, and data acquisition systems, etc. (here are some cool projects you can build with Arduinos). Arduinos make building robotic systems economical and easily accessible for hobbyists possessing basic coding and electronics knowledge.

Control

Sample frequency

The Arduino queries the sensors and notes down the values they read multiple times within a fixed period of time. This is the concept of sampling, and the rate at which this is done is known as sampling frequency.

Analogue and digital

The sensor inputs to the Arduino can be either **digital** or **analogue**. Digital signals are discrete (only taking on fixed values), and they are useful for, say, a trigger mechanism that has 2 possible states such as the beam break sensor (broken, or unbroken). Analogue signals can take on a continuous value between a range, which is representative of real-life phenomenon such as temperature.

However, most computers nowadays are digital, so we need an interface between the digital world of computer software, to the analogue world around us. This is achieved with an ADC (analogue-to-digital converter), which divides or *quantizes* the continuous range of values into discrete ‘steps’. It then rounds the actual value to the nearest discrete value. The number of steps this range is converted to is determined by the resolution of an ADC, which is the number of bits (a 0 or a 1 in binary) used to represent the signal. A 12-bit ADC can represent $2^{12} = 4096$ discrete values. You don’t have to completely understand how this works at this stage as the conversions are given to you in this handout, but it is handy to keep in mind that this is what goes on “under the hood”.

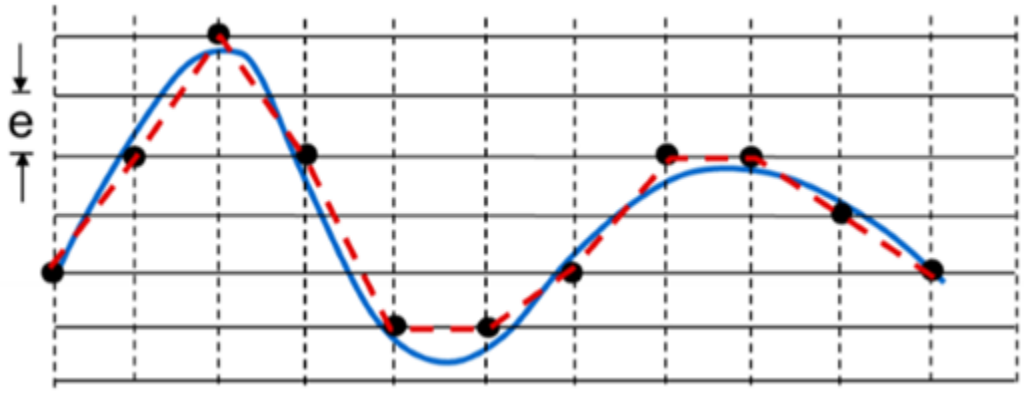
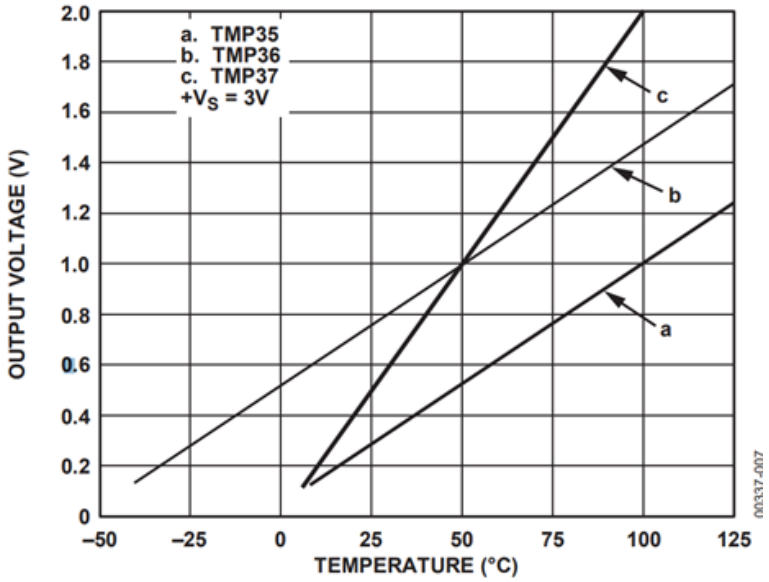


Figure 3. Quantization of a range of continuous values, and conversion of a continuous signal to digital values. [Taken from Ben Hanson MECH0023 Notes]

Signal processing: voltage to temperature conversion

Often when you take a reading from a sensor, it will not be in a form that is useful for your analysis. A physical quantity can be measured directly (such as measuring length with a ruler), or when it is difficult to do so, correlated to another more easily measurable or processable quantity. The TMP36 temperature sensor we are working with do not measure temperature directly, but outputs a voltage as this can be read and processed

by a digital computer. The scaling to be used to convert this to a temperature reading is usually stated on its technical data sheet, as shown in Figure 4 (always read the data sheets!). Again, this conversion is done for you below.



Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage at 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500

Figure 4. TMP35/36/37 output characteristics. [4]

Experimental Setup

The physical setup for the lab is shown below. Familiarise yourself with the different elements present in the hardware side to better understand how this interfaces with the software side later.

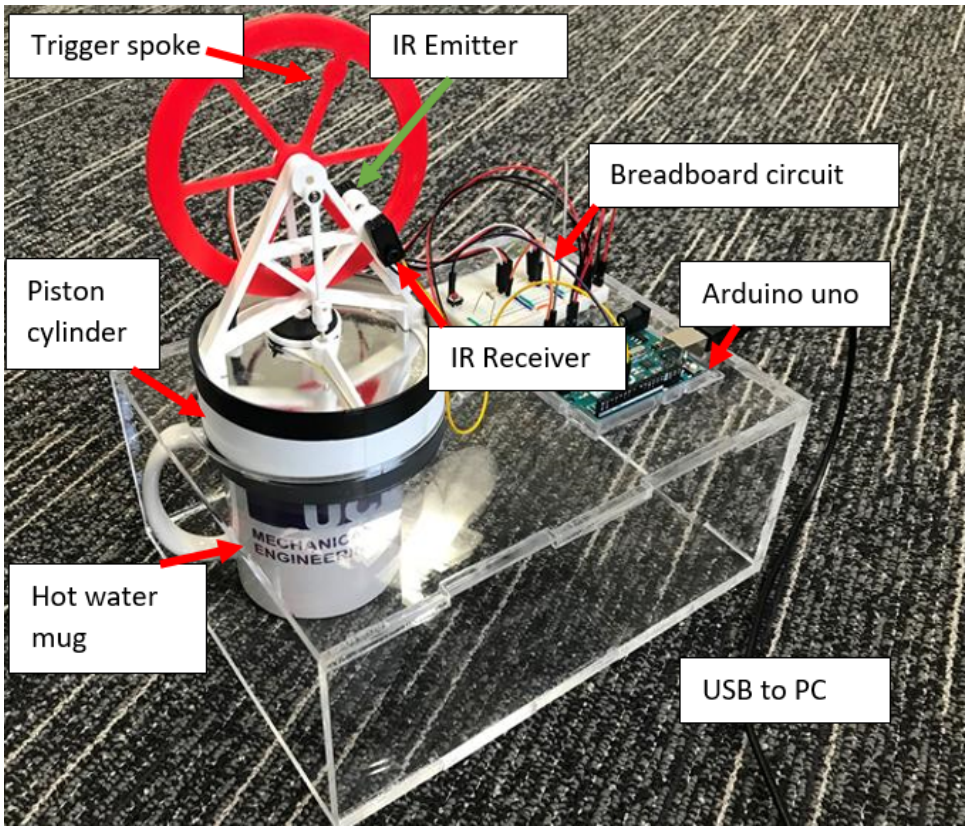


Figure 5. Stirling engine setup (front)

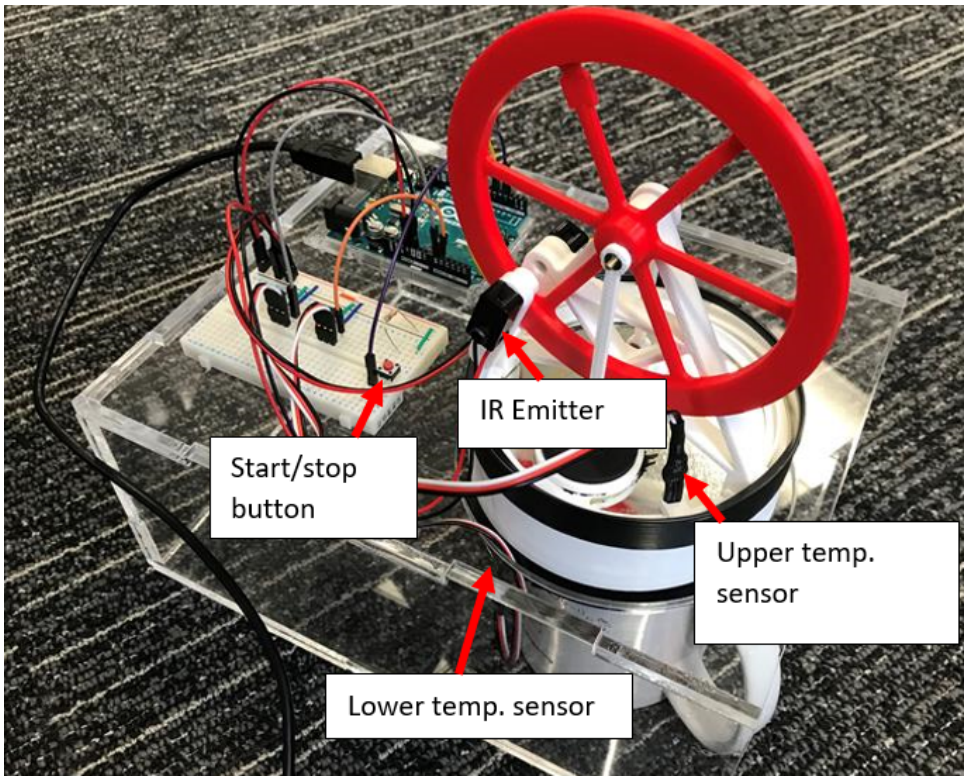


Figure 6. Stirling engine setup (back)

Software

Description

We need instructions for the computer to tell it when to take measurements from the sensors, and what to do with those measurements. In this case, we want to save it to a .csv file so that we can post-process it.

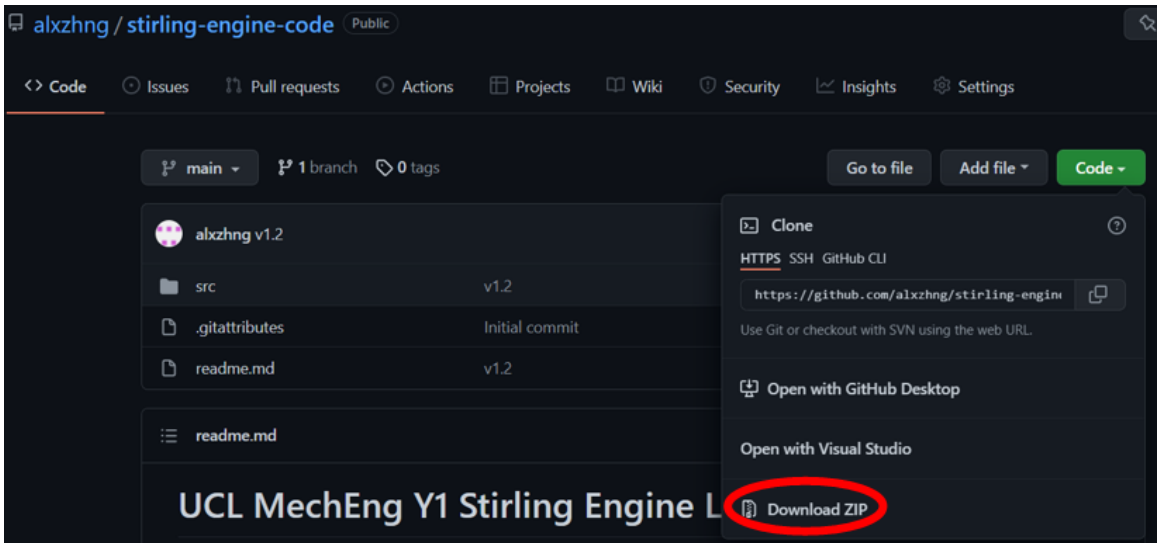
This is achieved on the software side with a two-part code, `stirling_engine_arduino.ino` written in C/C++ which is uploaded to the Arduino to control it, and `stirling_engine_csvwrite.ipynb` that is written in Python and runs in Jupyter Notebook on the PC to save the data to a .csv file.

These two separate scripts communicate with each other over the serial bus – the Arduino sends data values to the computer over a USB cable at the sampling frequency, and the computer enters this into the file. This operation is performed over a time period of 10 minutes (the code runs for 10 minutes by default, but you can interrupt

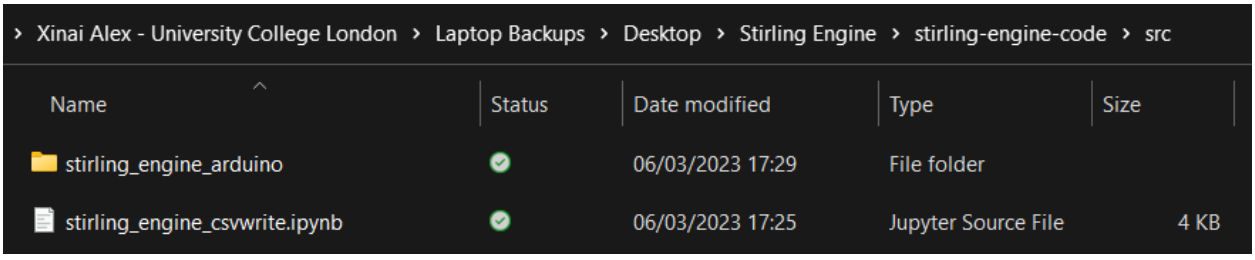
the code by pushing the button again while it is running if for example your engine stops running).

Instructions for running the lab

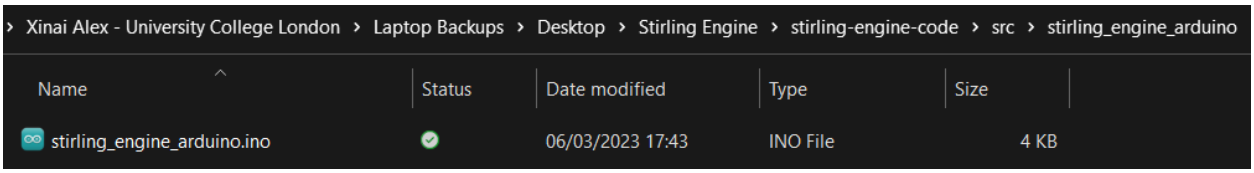
1. Download the code you need from https://github.com/alxzhng/stirling-engine-code by clicking on Code > Download ZIP.



2. Unzip the folder to Desktop and rename to '`stirling-engine-code-main-groupXX`'. (For example, for group 1 it will be '`stirling-engine-code-main-group1`'
3. Go into the `src` folder within the main folder. There should be 2 pieces of code here, the '`stirling_engine_arduino`.`ino`' script within a folder of the same name and '`stirling_engine_csvwrite.ipynb`' script.

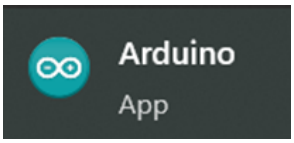


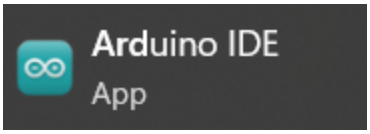
In the /src folder



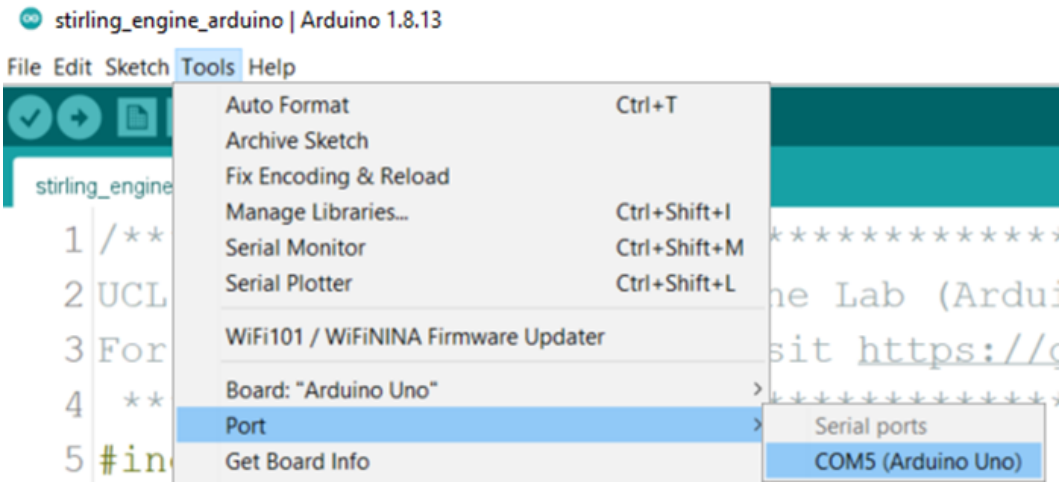
In the /src/stirling_engine_arduino folder

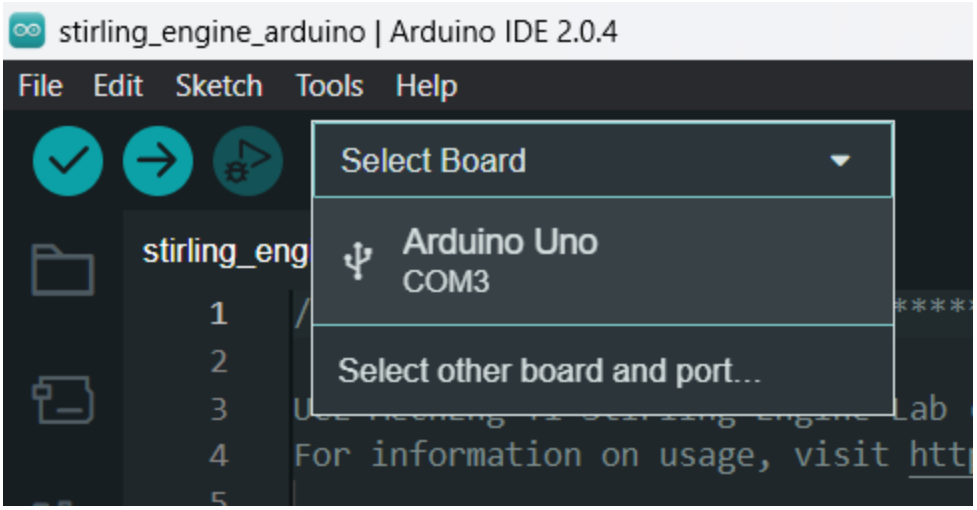
4. Launch the `.ino` file in Arduino IDE.



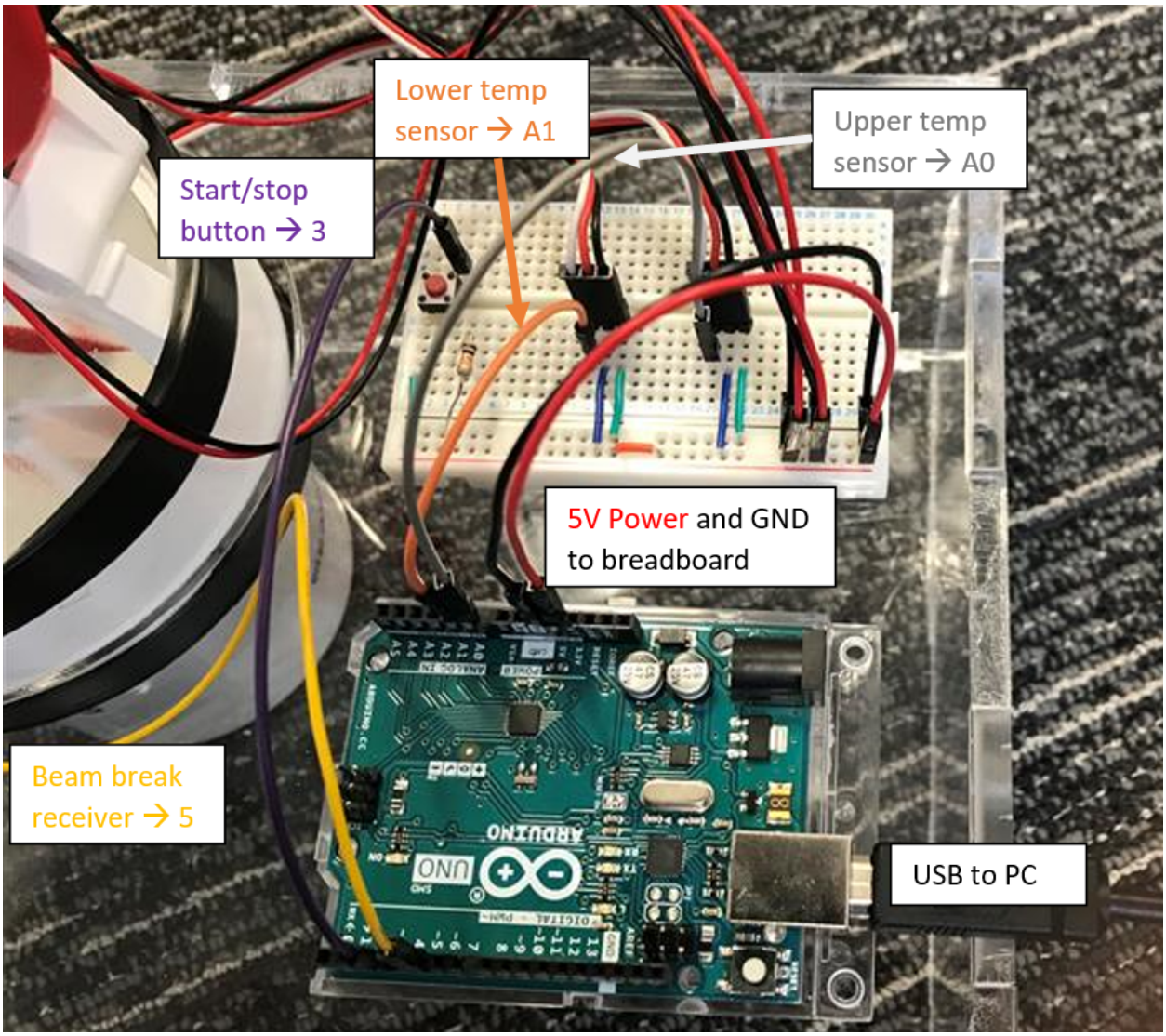


5. Under Tools > Port, check that the correct port has been selected. If there are multiple ports, pick the one that says Arduino Uno. Take note of the COM port number.



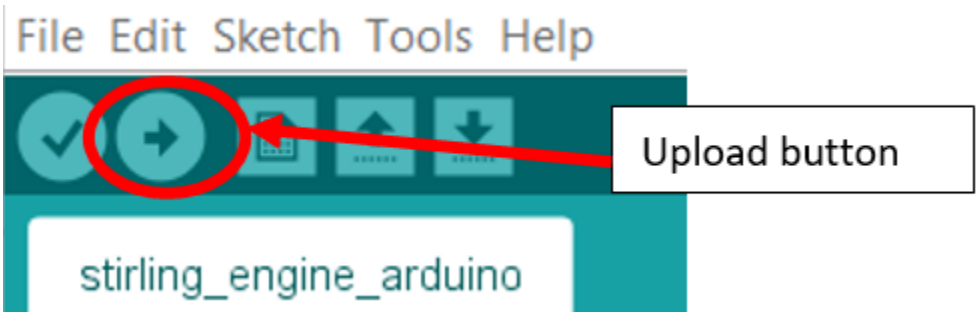


6. Check the pin definitions in .ino with the real physical setup that the pins are defined correctly.

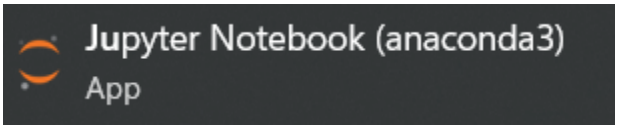


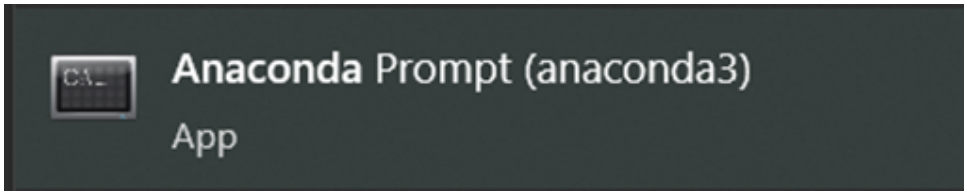

```
8   //---------------------- SET UP ----------------------//
9   // Pin Definitions
10  #define buttonPin 3          // pin connected to button
11  #define irSensorPin 5        // pin connected to IR sensor
12  #define ledPin 13            // pin connected to green LED (internal Arduino connection)
13  #define tempPin1 A0          // pin connected to temp sensor 1 (top)
14  #define tempPin2 A1          // pin connected to temp sensor 2 (bottom)
```

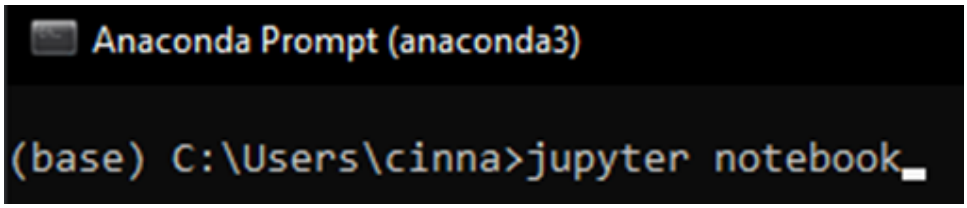
7. Upload the code to the Arduino. If you receive any errors, check the troubleshooting section, or call for help.



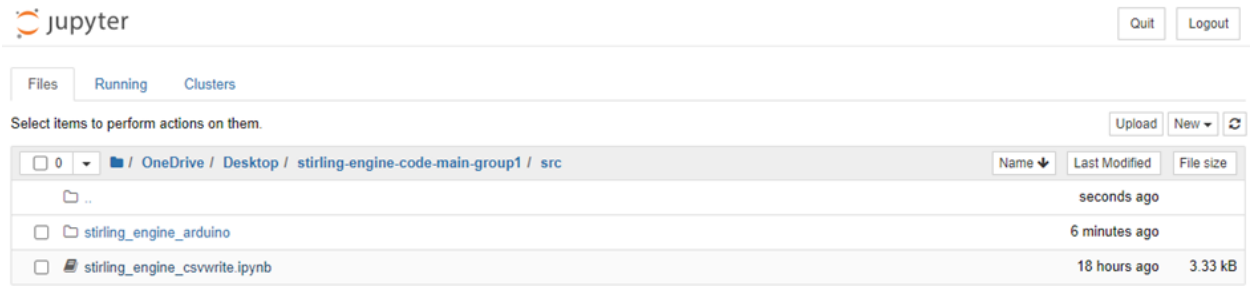
8. Launch Jupyter Notebook. If using Anaconda Prompt: type in ‘jupyter notebook’ and enter.







9. Navigate to your group’s saved main/src folder and open the .ipynb script.



10. Edit the port number to the correct one if necessary. Change the file name to what you want to save it as, keeping the .csv extension. Do not use spaces. Run the ‘Settings’ cell.

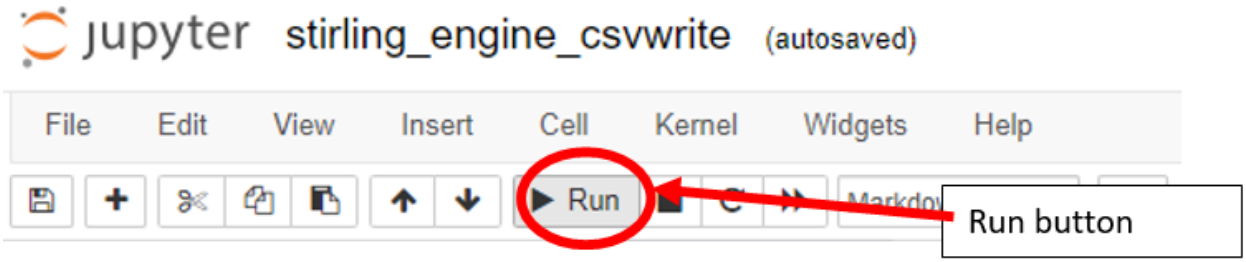
Settings

Check that the port name matches with what is shown in the Arduino IDE. (Case sensitive) The baud rate should be the same as that which was chosen in the Arduino code. Change the file name to what you want the .csv file to be saved as.

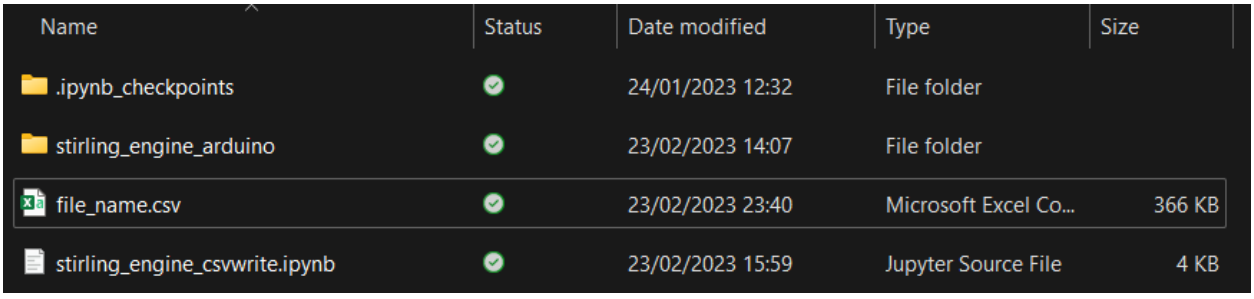
In [ ]:
```
# Settings
arduinoPort = "COM6"
fileName = "file_name.csv"

print("Connected to Arduino on port: {}, and writing to path: stirling-engine-code/src/{}".format(arduinoPort, fileName))
```

```
Connected to Arduino on port: COM6, and writing to path: stirling-engine-code/src/file_name.csv
```

11. Run the next cell and follow instructions printed onto the console to take measurements. The code will take measurements for a full 10 minutes by default, but can be stopped if the red button is pressed at any time.



12. Check that the file has been created in the `main/src` folder, and that data is being written. (The size of the file should be increasing, but DO NOT open the folder while it is being written)



Name	Status	Date modified	Type	Size
.ipynb_checkpoints	✓	24/01/2023 12:32	File folder	
stirling_engine_arduino	✓	23/02/2023 14:07	File folder	
file_name.csv	✓	23/02/2023 23:40	Microsoft Excel Co...	366 KB
stirling_engine_csvwrite.ipynb	✓	23/02/2023 15:59	Jupyter Source File	4 KB

14. When done with the measurements, navigate again to the group's main/src folder to save and export the result .csv file.
15. If another reading is required, press the reset button on the Arduino or reupload the .ino code, change the file name and run the two cells again.

Post-processing

TMP36 Voltage to temperature conversion

An Arduino Uno reads voltage at its I/O pins. It operates at 5V and has a 10-bit ADC. This means that it can represent a continuous voltage signal as one of $2^{10} = 1024$ values (from 0 to 1023) — the temperature reading from the pins A0 and A1 are therefore within this range. Each discrete 'step' corresponds to 5V/1024 = 4.883mV. So to convert the discrete signal read in to a voltage, it is multiplied by this value.

Then to convert this voltage into a temperature, we refer to the TMP36 datasheet values in Figure 4. The offset voltage of 0.5V is subtracted, and the results scaled using a scaling of 10mV/degC.

Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage at 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500

Figure 4. TMP35/36/37 output characteristics. [4]

The conversion from the temperature pin inputs `tPin` to a temperature in degree Celsius is:

```
# Convert ADC to voltage in mV - 5V(5000mv), 1024 adc
volt1 = float(tPin1) * 5000.0 / 1024.0  # cast into float, then multiply by resolution
volt2 = float(tPin2) * 5000.0 / 1024.0

# Refer to TMP36 datasheet for output characteristics
temp1 = (volt1 - 500) / 10.0  # offset voltage 0.5v(500mV), scaling 10mV/degC
temp2 = (volt2 - 500) / 10.0
```

Troubleshooting and help

▼ Error when running Main Code in Jupyter Notebook: `SerialException : could not open port 'COM3': FileNotFoundError(2, 'The system cannot find the file specified.', None, 2)`

Check the COM port in Jupyter corresponds to Arduino code.

▼ **The serial outputs strange symbols.**

Check that the baud rate of the .ipynb code/serial monitor matches that of the .ino code.

▼ **The program starts or stops without physical button press.**

1. Check the physical connection of the button on the breadboard.
2. Check the connection from breadboard to Arduino.

References

[1] https://homeplusexpress.com/stirling-engine-for-home-power-generation/

[2] https://microcontrollerslab.com/tmp36-temperature-sensor/

[3] https://learn.adafruit.com/ir-breakbeam-sensors/circuitpython

[4] https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf