# Data Aqusition and Handling: Remote Weather Sensing System

Project G

Project done by Hannah Nicholson and Elian Ruijter

## Contents

# 1. Introduction

Have you ever been frustrated with the weather forecaster getting the forecast wrong? Well, here is how you can do it yourself. Our primary goal was to build a sensing system that measured temperature, humidity and water level and uploaded this data to a wifi hotspot. The personal Arduino weather station can record measurements in real-time right outside your house. The next step to build upon this system was to take data from the Met Office and other local weather stations to complement the data taken with the Arduino. This extra data will give a broader perspective of the weather for the whole month. If the user is not home nearby the hotspot but still wants to know the weather local to their house, an email or text could also be sent to them from the Arduino, using google's API or Twilio's API, respectively. A warning system can also send an email in case of ice if the temperature goes below four degrees Celsius.

This collective system would be helpful for people in remote places without internet access or who want Instant local weather updates as frequently as they like.

## (a) The Three Main Parts to the Project

### (i) Sensing and wifi micro-controller components

This collective system recorded and uploaded the data to a local wifi hotspot, which could be accessed from any device that knew the IP and was connected to it. It was made up of six main parts. An Adafruit HUZZAH ESP8266 WiFi micro-controller breakout (WMC), DHT22 Temperature Humidity Sensor (THS), Water Level sensor (WLS), Breadboard, USB to TTL UART 6PIN CP2102 Module Serial Converter (USB) and a DC power supply set. This setup was tested in the lab, but it can be operated anywhere with a local hotspot setup, for example, at home, work, or university, provided the IP in the driver code is set correctly.

### (ii) Retrieving data from the hotspot

This is a python script to retrieve the data from the local server once uploaded by the WMC. It returns the sensors' three readings, temperature, humidity and water level. The device running the code must also be connected to the hotspot for this to work.

### (iii) Gathering data from other local weather stations

Two datasets from online weather stations were chosen to be incorporated into this project, one with data from previous years of the same month and the other with individual day data from the past year. This python script plots the data from each day of the current month and calculates the average per day compared to that month's average for previous years. As for the temperature and humidity data, a CSV will be saved to the file's directory, and each time a measurement is made on the Arduino, the new data will be saved to that file. Both these data points are plotted on a graph against days of the month, so the user can see the broader picture.

# 2. Materials

## (a) Adafruit HUZZAH ESP8266 WiFi micro-controller breakout (WMC)

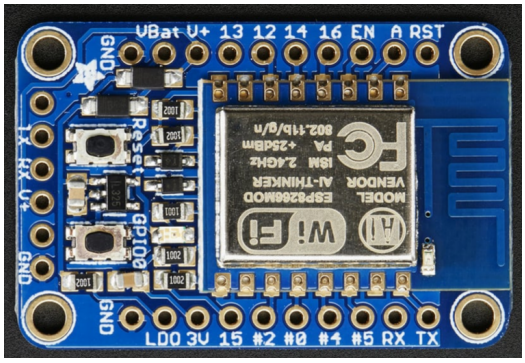



*Figure 1: Adafruit HUZZAH ESP8266 Wifi micro-controller breakout without pins. R*etrieved *from [1].*

This component was the most crucial part, it took the data from all the other parts and, along with some code, uploaded the data to the local server in the lab. It has two buttons a reset button that would be pressed before uploading the driver code and a user button to put the chip into bootloading mode. The red LED was used for a blink test and 3.3V out connected to the ground. The pins used include the analogue input pin, A; ground, GND; a 3V output; the power supply, UBat, set to 4V and 250mA and two 3.3V GPIO pins, #0 and 12. [1], [2].

## (b) USB to TTL UART 6PIN CP2102 Module Serial Converter

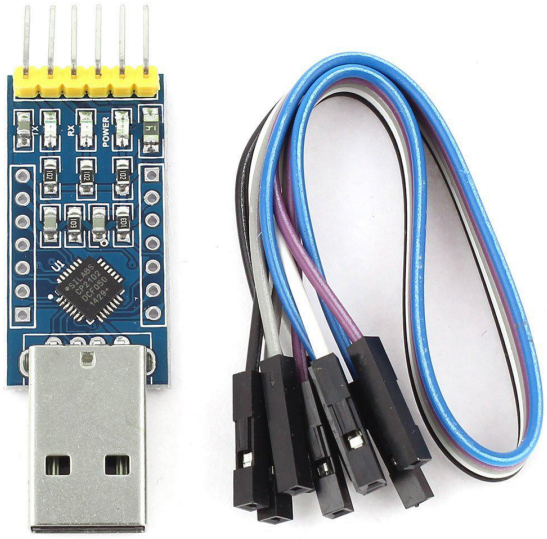


*Figure 2: USB 6-pin CP2102 module serial converter. R*etrieved *[5]*

This component is a USB connection to upload the driver code to the WMC. Four of the USB pins were connected to four pins on the left side of the WMC. See Figure 1. [3], [4].

## (c) DHT22 Temperature Humidity Sensor (THS)

This sensor was connected to the WMC 3.3V power supply, a GPIO pin and ground via the breadboard. No analouge pins are needed for this component because it has its own internal ADC inside. It only has one data pin, however, is not Dallas One Wire compatible, but as this is the only sensor going to a particular GPIO pin we did not need to worry about conflicting data signals. See [7], [6] for more information.

The range of the sensor is 0-100% in temperatures of -40 to 80 degrees celsius and has an accuracy of 0.1% RH (relative humidity) and ＜+/- 0.5 Celsius. The resolution for the humidity

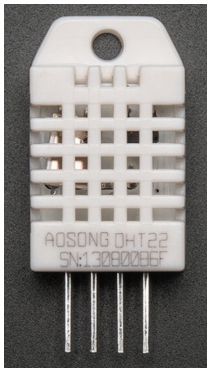



*Figure 3: DHT22 Temperature Humidity Sensor. R*etrieved *[6].*

is +/- its accuracy and +/-0.2 Celsius for the temperature.

With these specifications, the sensor is more than capable of recording measuremnts for the job of recording data fro the weather as the range in which the accuracy functions in comfortably contains the temperatures of the UK. If the device were to be used in a locations that go well below freezing, it need to be ensured that if the temperature goes below -40 Celsius the user is aware.

Extension wires can also be used for example for the sensor to reach outside of a window (but protected from rain damadge).

## (d) The water level sensor (WLS)

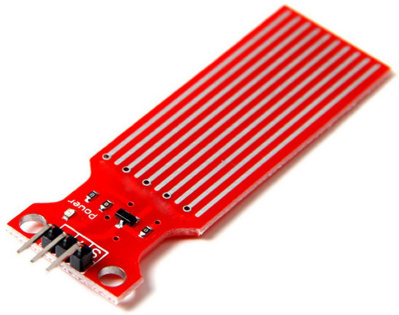


*Figure 4: Water level sensor made by Arduino. Retrieved [8]*

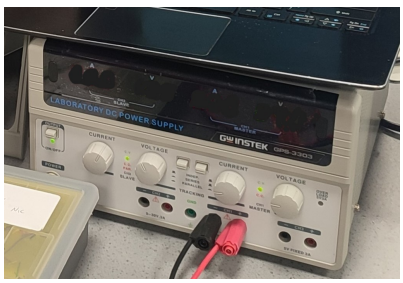
The primary purpose of this component is to track the level of rainfall in the environment, which is meant to be emptied daily to record total rainfall per day. It only has three pins; signal, power and ground, however, it needs to be depth calibrated. The maximum depth it can record is 4cm. There are ten copper strips going down, five power traces and five sense traces going across alternativley. When the sensor is submerged in water this connects the two different strips to each other and allows a current to flow. The more it is submerged the less resistance it needs to overcome, acting like a variable resistor. This varying voltage can be converted into a depth when callibrated. Imformation found here [9].

## (e) Rasberry Pi 4

*Figure 5: Rasberry Pi 4. Retrieved [10]*

The Rasberry Pi was connected to a monitor in the lab and used as a simple computer to run python scrips and retrieve data from the local server. A mouse and keyboard were also connected to the Rasberry Pi to control it. A simple python text editing software was used to run and execute the code [10].

## (f) DC power supply




A DC supply in the laboratory was used to power the components on the breadboard for testing. It was used rather than connecting the USB to the Rasberry Pi or a PC because it is a more reliable source. If connected to a PC or the Rasberry Pi, the reliability of the water level sensor would be significantly affected.

*Figure 6: A photo taken in the lab on the 24th of Nov 2022 of the DC power supply when testing the components once assembled.*

### (g) Other Equiptment

- Windows 7 PC

  A Windows PC in the lab was used with the Arduino programme already installed [11] to upload and edit the driver code to the WMC.

- Wifi hotspot

  A hotspot in the lab was used to upload the data recorded by the sensors to a particular IP address which could then be retrieved later.

- Aduino Breadboard

  This is the standard breadboard, so all the components can easily connect with pins.

- Personal laptop

  A Personal laptop was used to write the code to retrieve data online from weather stations in Edinburgh. Visual Studio Code (VSC) was the chosen text editor for the Python scipt.

# 3. Mehods

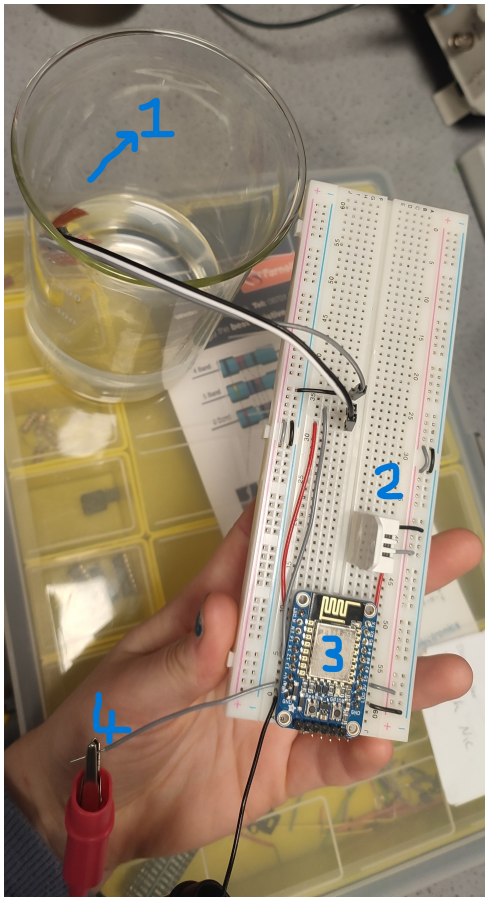



*Figure 8: The fully assembled components of the weather sensor. The WLS (1), THS (2), WMC (3), and DC power supply (4).*

### (a) Connecting the THS and WLS to the WMC


*Figure 7: THS required connections to the WMC. Retrieved [12].*

Figure 7 shows the three THS connections: data, ground and a 5V power supply. These were connected to GPIO #0, GND and the power supply in the WMC, respectively. It is shown in figure 8. Here is information on connecting the THS and its needs [12]. A key attribute to remember later is that the THS can only measure a maximum of once every two seconds.

The WLS has three pins connected to a data pin, power (3.3-5V) and ground. Note that the data pin for the WLS must be an analogue input pin in the WMC. Figure 1 the analogue pin in the top right is labelled “A”. It was necessary to connect the WLS power pin to a GPIO in the WMC, which only provides power when a reading is taken, to increase longevity. Quoted from the official Arduino dashboard: “If you leave them constantly powered-up, then they will basically be electro-etching themselves the whole time they’re submerged and have a very short life.” [13]. Figure 8 shows that we used longer wires for the WLS so it could reach the beaker without the other components being at risk of water damage.

### (b) Uploading and writing of the Driver Code

This code was uploaded to the MWC using the USB connection to a Windows PC. The FDTI pin out was used to upload the code to the breakout.

Four pins out of the six from the USB connections in figure 9 were connected to another set on the WMC. These were GND to GND, 5V to V+, RX to TX and TX to RX with USB to WMC respectively.

First the Arduino application had to be set up for the components we were using, following the instructions in the dah-course-booklet.pdf that can be found in Learn the ESP8266 package was installed as well as setting the correct board, CPU frequency, baud upload, and USB port. More detailed instrctions can be found here including the C++ code for the blink test [14].

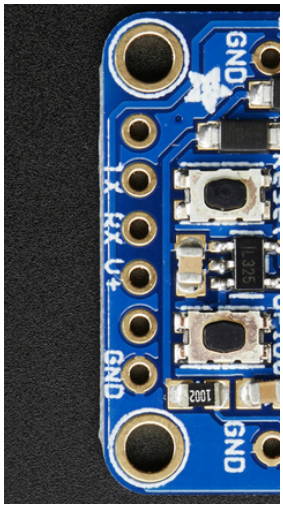



*Figure 9: Showing the four connections to the USB on the WMC. The labels are TX, RX, V+ and GND from top to bottom.*

Bottloading mode (a mode used for receiving firmware updates) was activated everytime code was uploaded to the WMC, including for the blink test. The blink test indicated weather the board’s main fucntions were in working order. Once this was successful, the code to connect the WMC to the wifi hotspot was uploaded, which can be found here [14]. The two variables in the code were updated to the specific details of the lab’s wifi:

```
const char* ssid     = "yourssid";
const char* password = "yourpassword";
```

The text “yourssid” was changed to DAH-local-wifi with its corresponding password, this is easily changeable for personal use and remote locations. The hotspot does not need a connection, just a local server to connect to when close by, which makes this sensing system ideal for places without an internet connection. The only drawback to

remote places without internet is that they will not get data from other weather stations online, only the Arduino sensors. The sensor's data is saved to a CSV, each with a date and timestamp (more detail in **6. The code**). Therefore, the user will only lose out on data when they do not make measurements themselves.

The driver code to access the hotspot was combined with code to read and upload the THS and WLS data. Examples for the code were taken from here [15] and here [12] and the combined C++ scrip explicitly written for this setup can be found here [17] with the name: weatherstation arduino.cpp.

The length of the WLS was measured to be 40mm long, the limit of rainfall it can measure until the beaker is emptied. The code for the WLS [15] was therefore edited to have 40 increments, allowing the value to be easily converted into millimetres as each increment was 1mm. Shown below;

```
water_level = map(water_value, SENSOR_MIN, SENSOR_MAX, 0, 40); // 40 levels
```

The values 0 and 40 are set to be the website's minimum and maximum display values.

To combine the code for the THS and WLS, a function to read the WLS was added to the THS script returning the value in millimetres (cal). Then this function was called in `setup(void)` that makes a unique address within the server's IP for each of the three readings. For example, "the IP address" + ("/temp" or "/humidity" or "/water_level") which takes you to a page with the writing "Temperature:" or "Humidity:" or "Water level:" with the value and its units after. Go to the beginning of 5. Calibration and Testing to find out how to get the IP address.

Once the driver code was uploaded, the USB could be ejected and unplugged from the Windows computer. The breadboard was then connected to a DC power source set to 4V and 250mA, recommended on the Adafruit website [1]. The DC source used is in figure 6, and figure 8 shows the breadboard connected to it (labelled 4 in the figure). Once connected to the power, the circuit will continuously take measurements and upload them to the server until unplugged, given that everything is in working order.

# 4. Calibration and Testing

## (a) Accessing the Measurements

The uploaded values can be viewed on any device that is able to connect to the local server. In the lab this was done with the Rasberry Pi in Google Chrome. While the WMC was plugged into the Windows PC a note was made of the specific IP address to access the data. This could be found by going in Tools, then Serial monitor after pressing the reset button on the WMC. When this is done a unique IP address was shown as the following:

> Working to connect .....
> DHT Weather Reading Server
> IP Address: 192.168.1.2
> HTTP server started

The instructions for this were found in the file dah-course-booklet.pdf on the Univerity of Edinburgh's Learn page as explained in chapter **4. (b) The Assembly, Uploading and writing of the Driver Code** and [16], the specific value was obtained by writing the IP and then the measurement: "192.168.1.2/temp". To obtain new values, the page needed to be simply refreshed.

## (b) Testing the THS

The first set of testing was on temperature, values were around 20 degrees Celsius. When a hairdryer was blown on the THS component, as expected this value increased rapidly. The hiardryer was kept on it at about a distance of 30cm untill it reached 50 Celsius, to avoid damadge. To see the values live chrome was refreshed apporximatly every 2 seconds. To check the absoulte temperature the THS was cross checked with others in similar circits. Out of the three in the lab all values agreed with each other within +/-1.3 Celsius, this is expected to be larger than the error of

the sensor due to fluctuations in temperature throughout the room. The same was done for humidity readings that fluctuated +/-5%.

### (c) Testing and Callibrating the WLS

To use the WLS correctly, maximum and minimum values of the variable `water_value = analogRead(SIGNAL_PIN)` in the function `readwater_level()` had to be found and set equal to the variables `SENSOR_MIN` and `SENSOR_MAX`. To find the minimum, the sensor was wiped with a dry cloth taking away any excess droplets that would allow current to flow between the power and sense trace strips. Several analogue values were recorded while the sensor was completely dry. It was discovered while doing this that the precision of the WLS was low, as values would vary a lot across measurements. The lowest value recorded was 1, but as it varied a lot, the `SENSOR_MIN` was set to zero. The maximum value was found with the sensor entirely submerged to the top of the power and sense trace strips. The values recorded varied even more broadly, with +/- 50 of the analogue value recording a maximum value of 549, giving a error of 9%. About twenty measurements were made to ensure this value was closest to the true maximum because there would be problems later if the value exceeded it. According to [18], which uses a very similar sensor, the analogue output ranges from 0-1023 because of internal resistance due to the wires and water and the component itself. This page recommends that the maximum value should be approximately half the analogue output. Considering all the information discussed, a value of 550 was set as the `SENSOR_MAX` value.

When using the WLS, you must be aware of its significant inaccuracy of 9%, which varies the 40mm increments by +/-4mm. Other users of this component also found that the precision was poor, according to [19] “the water level sensor is not very accurate, but it is good for water detection”. After finding this out, the code was changed slightly to display values to the nearest centimetre. Shown below:

```
water_level = map(water_value, SENSOR_MIN, SENSOR_MAX, 0, 4); // 4 levels
```

This would ensure more reliable values for the readings. The code was also changed to ensure the webpage output would remain in mm, the unit convention for rainfall. To check the WLS calibration, water was filled at each centimetre increment up, and as expected, the webpage displayed 10mm, 20mm, 30mm and 40mm for each. The WLS is not very accurate, but because even 10mm of rain is significant for just a day, it will give the user a good idea of the rainfall.

## 5. The Python code

### (a) Retrieving Sensor Data

There is a Python file in [16] named our_sensor_data.py that contains two functions that take the data uploaded to the hotspot and returns the temperature, humidity and water level respectively. To ensure there is a high accuracy ten values are taken every two seconds (two because of the THS limit), added to a list and then averaged. These averaged values are the values returned to the user. The next section talks about how this data is used and displayed to the user.

### (b) Plotting Data

To complement the data recorded by the sensing system, a Python script (named project.py in [16]) was written to retrieve data from the Met Office and another weather station in central Edinburgh and compare it to data taken by the WLS. It also saves all previously recorded values in a CSV so the user can look back at it and visualise it with graphs. There are many functions in the Python file, but here I will only describe the motions of the core functions. Each function is described in detail in the file on [16] if more information is needed.

The function `get_online_data(link)` takes the .txt file from the link passed to it [20], saves it in the directory of the Python file as `rainfall_data.txt`, and returns the data as a Pandas DataFrame. The link contains the Met Office monthly data

for Eastern Scotland as far back as 1836. This data is used later to compare the current month's values from the sensors. The `function get_online_json(link, jsonname)` retrieves .json files from the link given and saves it as jsonname to be used later for comparison. The function is called twice, once for daily data [21] and for recent monthly data [22], both from the same weather station, Comiston Rain Gauge (SEPA). More information can be found here [23]. Next, the function `avg_rain_thismonth(jsonfile)` takes the monthly rainfall and finds the average and cumulative sum. Then the Met office .txt file is used in `comapre_previsous_year_rain(cum_rainfall_this_month)` to average the past 50 years of the current months' rainfall. The previous fifty years' average is compared to this month's cumulative sum and prints out how much this month's rainfall is over or under in %. For example:

```
'Already! We are above average rainfall for ths month by 20%'
```

Then using this data, another function, `plot_rainfall_graph()`, takes in all of the rainfall data from online and the WLS and plots it on a graph. An example of the graph in figure 10 shows the daily rainfall, with the averages for this month and the previous fifty years. The WLS's data is the X; if the user takes data every day for the month, the X will show for all the days.

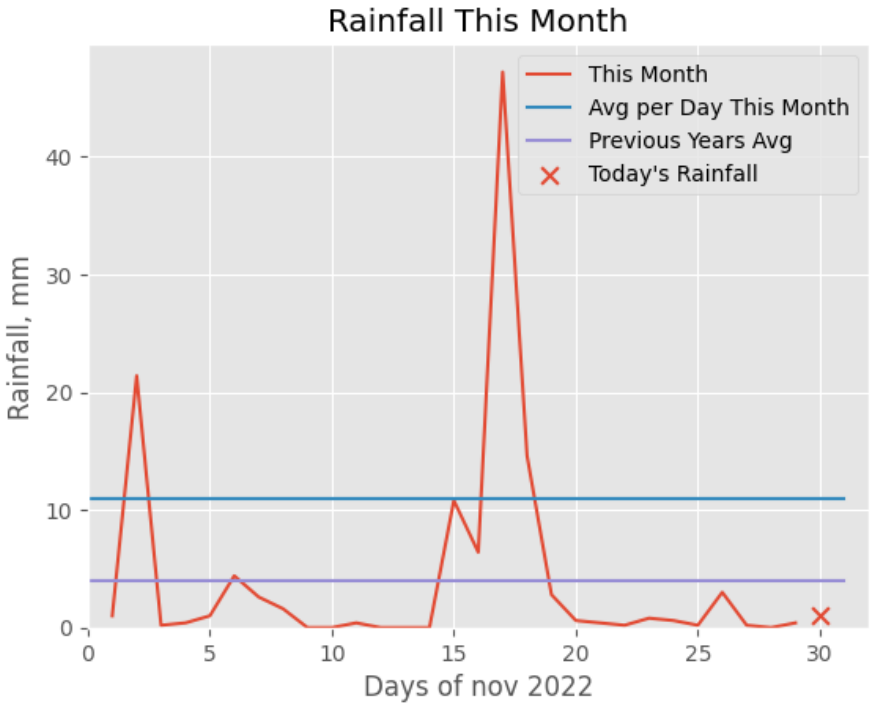



*Figure 10: Shows the daily rainfall with a red line taken from [21] and the WLS's value shown by the red X. The average so far this month per day is in blue from [22], and the previous fifty years' average is in purple from [20].*

Finally, the function `plot_humidntemp(temp, hum, rain, plotmonth, plotyear)` is called in `main()` to plot the temperature and humidity of the month from the THS data (figure 11). New Arduino data is saved to a CSV (sensor_data.csv) with the date and time when the function is called. An example od what the CSV looks like is in table 1 with the columns temp, humid, and rain representing the temperature (in Celsius), humidity (in relative %) and rainfall in mm.

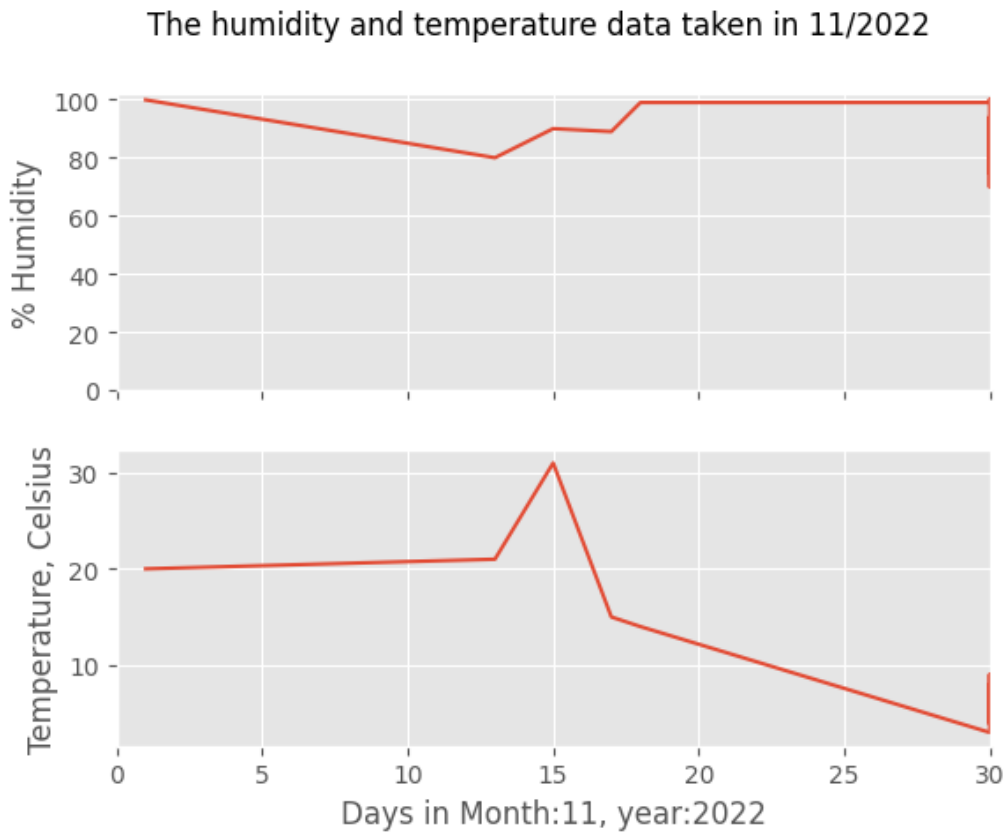



Figure 11: The temperature and humidity as a function of days of the current month. Any year and month can be chosen to be plot by the user, providing they have saved the readings in the CSV.

Both plots in figure 10 and 11 are saved with a date stamp in their name so the user can identify the plots easily, and look back on them. The rainfall data taken from the web are from Edinburgh weather staitons, however they can be changed to other stations where the user is situated. This can easily be done by changing the link in the python file for a a different one.

| temp | humid | rain | day | month | year | time |
|---|---|---|---|---|---|---|
| 0 | 87 | 30 | 26 | 12 | 2022 | 18:40:17 |
| 3 | 99 | 20 | 27 | 12 | 2022 | 18:40:25 |
| 1 | 89 | 20 | 28 | 12 | 2022 | 18:41:08 |
| 4 | 86 | 30 | 29 | 12 | 2022 | 18:43:12 |
| 0 | 87 | 30 | 30 | 12 | 2022 | 18:40:17 |

*Table 1: An example of the CSV saves each time the python script is run. the values are from the THS and WLS.*

# 6. Conclusion

The weather sensing system as a whole completes the purpose of telling the user the current and past temperature, humidity and rainwater levels, along with graphical interpretations of the data. The precisions for the three are +/-1%, +/-0.5 Celsius and +/-4mm, respectively. The temperature and humidity errors are much better than necessary for a

home use system, but the WLS could be better with only four increments. If the WLS had better accuracy, the user would get a better idea of the rainfall per hour. It also can not measure above 40mm, so if there is more than that per day, which is not uncommon, the user would have to empty the WLS's beaker, and if they do not, they risk damaging the component. Despite the inaccuracies, that is why data is taken from the Met Office and SEPA to complement the weather sensor. As well as this, the station is highly customisable with small effort changes to the code. For example, to plot hourly data instead of daily or changing the weather station's location if the user is not situated in Edinburgh. A plot like in figure 10 can also be replicated for temperature as this data is easy to find online.

# 7. Discussion

Given more time, a casing for the breadboard and components could be made to protect them from the weather so the user could leave it for long periods. To remove the need for the user to empty the beaker once daily, a modified siphon device could be used for the WLS beaker so that when the water level goes above 40mm, all of it flows out. This would also prevent damage to the wiring if more than 40mm of rain fell in a day.

Currently, the readings are only saved when the user runs the python file, but this could be changed to run, for example, twice a day automatically. The user also needs to be within range of the hotspot, so an alternative solution would be to upload the data to the general internet, where it can be accessed anywhere. The CSV history of all previous measurements could also be uploaded to the internet alongside the current ones. If the weather station detects more than a certain amount of rain or temperatures at four degrees Celsius or below, the user could be sent a warning text saying, for example, 'flood warning' or 'ice warning'. This would be incredibly useful as it could save the user from accidents when driving on the road.

# 8. Bibliography


1. Ada, L.(2015, April 24). Adafruit huzzah ESP8266 breakout. Retrieved November 28, 2022, from https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/overview
2. Ada, L. (2021, November 15). Adafruit HUZZAH ESP8266 breakout. Retrieved December 01, 2022, from https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah-esp8266-breakout.pdf
3. Labs, S. (2008, August). SINGLE-CHIP USB TO UART BRIDGE. Retrieved November 26, 2022, from https://sisy.name/mymcu_download/produkte/controller/db_cp2102.pdf
4. S. (n.d.). USB-TTL UART module - CP2102. Retrieved November 26, 2022, from https://www.sunrom.com/p/cp2102-usb-ttl-uart-module
5. Antenna, A. (2015, January 22). ANTV Amplified Outdoor HDTV Antenna. Retrieved November 26, 2022, from https://www.ubuy.com.bh/en/catalog/product/view/id/1770281/s/uxcell-cp2102-usb-2-0-to-uart-ttl-6pin-m/
6. Berners-Lee, T. (n.d.). DHT22 temperature-humidity sensor + extras. Retrieved November 26, 2022, from https://www.adafruit.com/product/385
7. Liu, T. (n.d.). Digital-output relative humidity & temperature sensor/module. Retrieved November 26, 2022, from https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
8. A. (n.d.). Water Level Liquid Detection Sensor. Retrieved November 26, 2022, from https://alltopnotch.co.uk/product/water-level-liquid-detection-sensor/

9. Last Minute Engineers. (2022, October 16). In-depth: How water level sensor works and interface it with Arduino. Retrieved November 26, 2022, from https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/
10. R. (n.d.). Raspberry pi 4 model B specifications. Retrieved November 26, 2022, from https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
11. Söderby, K. (2022, November 23). Downloading and installing the Arduino IDE 2.0. Retrieved November 26, 2022, https://www.arduino.cc/en/software
12. Maleki, M. (2021, June 19). Interfacing DHT22 temperature humidity sensor with Arduino. Retrieved November 26, 2022, from https://electropeak.com/learn/interfacing-dht22-temperature-humidity-sensor-with-arduino/
13. Adiadi. (2018, October 04). Is the water level sensor waterproof? Retrieved November 26, 2022, from https://forum.arduino.cc/t/is-the-water-level-sensor-waterproof/549372/4
14. Ada, L. (2015, April 24). Adafruit huzzah ESP8266 breakout: Using Arduino IDE. Retrieved November 30, 2022, from https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide
15. Last Minute Engineers. (2022, October 16). In-depth: How water level sensor works and interface it with Arduino. Retrieved November 26, 2022, from https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/
16. Nicholson, H. (2022, November/December). Dah-report-2022: The Suplimentry material for the remote sensing system. Retrieved December 1, 2022, from https://github.com/isabelnic/DAH-Report-2022
17. Barela, A. (2015, May 6). Using the Webserver. Retrieved November 29, 2022, from https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/using-the-webserver
18. Haiges, S. (2017, October 24). Water level sensor. Retrieved November 29, 2022, from https://calliope.cc/en/examples/water-level-sensor
19. Rdagger. (2016, October 11). Raspberry pi analog water sensor tutorial. Retrieved December 3, 2022, from https://www.rototron.info/raspberry-pi-analog-water-sensor-tutorial/
20. M. (n.d.). Areal values from HadUK-Grid 1km gridded climate data from land surface network. Retrieved December 04, 2022, from https://www.metoffice.gov.uk/pub/data/weather/uk/climate/datasets/Rainfall/date/Scotland_E.txt.
21. Station nummber: 525510. SEPA. Retrieved December 4, 2022, from https://www2.sepa.org.uk/rainfall/api/Daily/525510?all=true.
22. Station number: 525510. SEPA. Retrieved December 04, 2022, from https://www2.sepa.org.uk/rainfall/api/Month/525510.
23. SEPA. (n.d.). Data Download for Comiston Rain Gauge. Retrieved December 4, 2022, from https://www2.sepa.org.uk/rainfall//data/index/525510
24. SEPA. (n.d.). Data Download for Comiston Rain Gauge. Retrieved December 4, 2022, from https://www2.sepa.org.uk/rainfall//data/index/525510