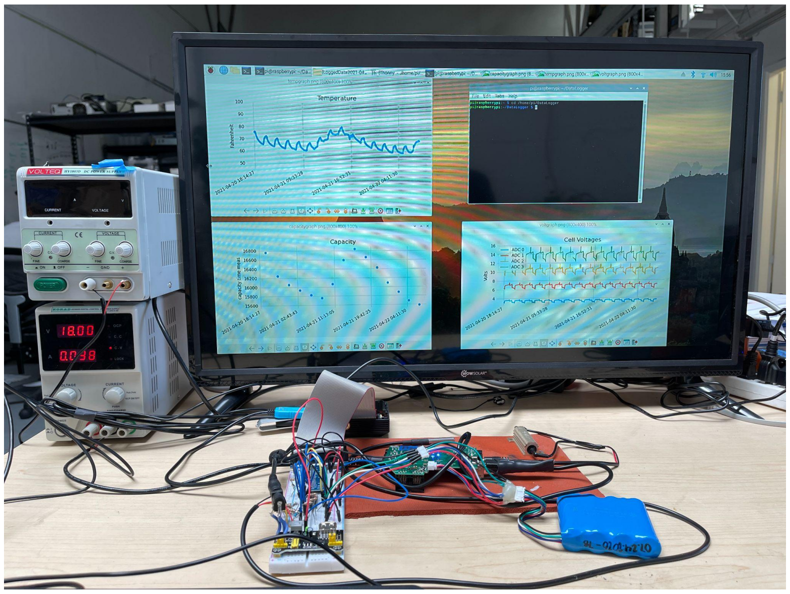
RASPBERRY Pi DATA LOGGER INSTRUCTIONS



Setting up a new Raspberry Pi

- Downloading libraries
    1. at the command prompt, enter **sudo pip3 install keyboard**
    2. at the command prompt, enter **sudo apt install python3-pandas**
    3. at the command prompt, enter **sudo pip3 install adafruit-blinka**
    4. at the command prompt, enter **sudo pip3 install adafruit-circuitpython-mcp3xxx**
- SPI Enable
    - at the command prompt, enter **sudo raspi-config**
    - select “Interfacing Options”
    - select and enable “SPI”
    - if prompted to reboot select “Yes”
- 1-wire thermocouple setup:
    1. at the command prompt, enter **sudo nano /boot/config.txt**
    2. then add or modify existing to be **dtoverlay=w1-gpio,gpiopin=17** to the bottom of the file
    3. Exit Nano, and reboot the Pi with **sudo reboot**
- disable screen blanking
    1. In RPi Configuratio nWindow, click on Display Tab
    2. Look for the Screen Blanking row, click Disable. Then click OK.
    3. A window will open and asks if you would like to reboot the RPi. Click Yes.
- don’t forget to calibrate the ADC, instructions are below

Running the program:

First, before starting to data log is to *configure 'main.py'.* Here the user configures different variables controlling sample rates, which ADC channels to read, file paths, and much MUCH more!

Next, the user must run the desired data logging program:

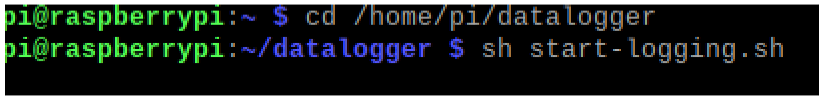
at the command prompt, cd to the directory where the 'main.py', 'plotter.py' and 'start-logging.sh' are located

Option 1: Running DataQ with Live Plotting

at the command prompt, enter:

sh start-logging.sh

- this is a bash shell script that does all the same functions as DataQ from Option 1, but ALSO runs a live plotting program to display the data in real time on the display



```
pi@raspberrypi:~ $ cd /home/pi/datalogger
pi@raspberrypi:~/datalogger $ sh start-logging.sh
```

Option 2: Running DataQ standalone

at the command prompt, enter:

sudo python3 -c 'import main; main.run_main()'

- this will acquire data, live save the data to a CSV, and upon termination save a copy of the data CSV in a folder with graphs of temp, capacity, and voltage

LAST, after the data has been acquired the user ends the program

- at the command terminal running the program, *hold 'alt+s'*
- this action completes the data logging and saves the CSV file and data graphs
- after acquiring large amounts of data it may take a few minutes to plot and save the graphs, be patient

Structure and notes:

Data Input:

- Temperature Sensor: collected using DS18B20 sensor, 1-wire interface, probe is on 1 meter cable

- Serial Communication: collected using smart cable connected between RPi USB port and SHS USB port. *Note that this data is cleaned and processed specifically for the capacity printouts in life cycler mode.* This capacity number is calculated on the SHS and not by the RPi.
- Voltage Data: collected using MCP3008, the RPi Hat has 8 female sockets where jumper wires are connected for voltage input.

File Structure:

- 'currentloggeddata.csv' is created and gets written to in real time as the program runs
- A folder created at the beginning of the program. Once the termination sequence has been initiated by pressing 'alt+s', a copy of 'currentloggeddata' and the data plots are saved in this folder

  Important Notes:
  - 'currentloggeddata' is overwritten each time the program is run
  - The location of the saved files is specified by the user in 'main.py'
  - Notice the name of the folder has the exact time the program began running and the individual files saved inside have the time the program was completed
  - if the data acquisition were to be abruptly interrupted(like by a blown circuit breaker LOL) the data up to that point will be saved in 'currentloggeddata.csv'. Before starting data logging again, this data could be manually copied by the user to a new CSV to be saved

Circuit Notes:

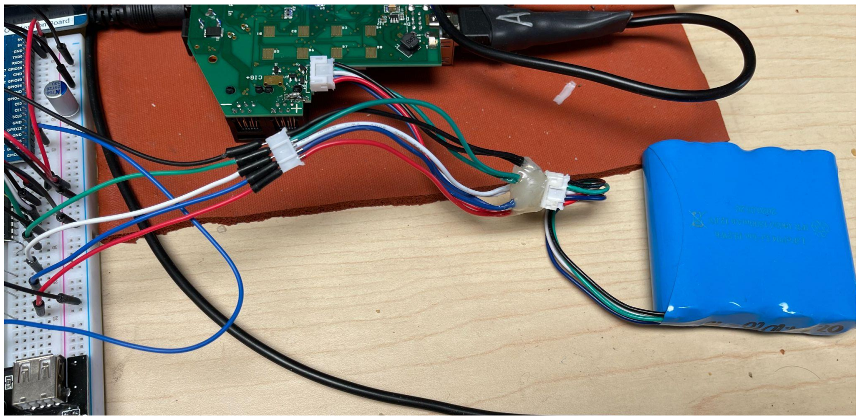
- The MAX input to the ADC is 20V for this set up. Damage will be caused by inputting larger voltages to the ADC.
  - 3:1 voltage dividers are used to reduce the input voltage to within the 0-5V range. 5V is the operating voltage on the MCP3008 and thus 5V is used as the reference voltage.
  - To increase the input voltage the ADC resistor pack values must be changed. Don't forget to change the voltage divider coefficient inside main.py after making this change!
- 3.3V is the max input to the RPi's GPIO Pins. Voltage dividers are necessary. You will notice a 2:1 voltage divider being used between the MISO pin on the RPi and the ADC. This is used because the ADC is running on 5V.

- The 5V power produced by the Raspberry Pi is noisy and unreliable. Powering the circuit by an external power source is highly recommended

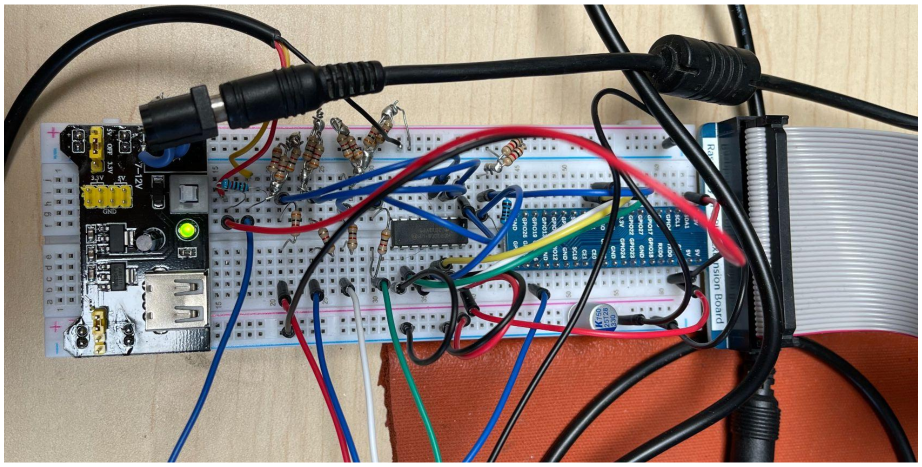
- Serial connection most easily made using USB UART cable

- Individual Cell Voltages:
    - if you want to find individual cell voltages turn v_sub True in ‘main.py’
        - this feature subtracts the differences between each ADC channel to report individual cell voltage
            - For example, with v_sub = True a pack with a total voltage of 13.8V, 3.45V per each cell would be graphed and recorded in the CSV. If v_sub = False in this scenario, 3.45, 6.9, 10.35, and 13.8 would be graphed and recorded in the CSV.
        - When v_sub = True make sure the lowest voltage input is connected to the lowest ADC channel you are using. The second lowest voltage input is connected to the second lowest ADC channel you are using, and so on...

- The adapter pictured below is what is used to sample the cell voltages



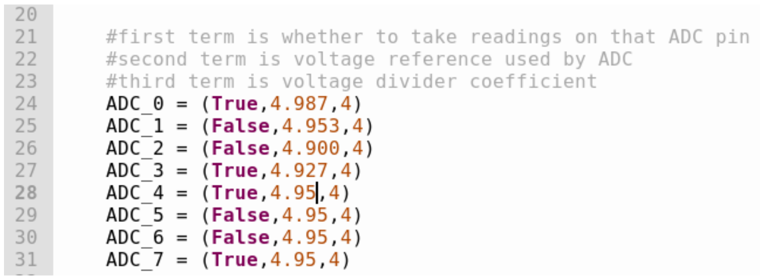
- Pictured below is the layout of the breadboard circuit



Code:

- The code to run the data logger can be found here: https://gitlab.com/ampedinnovation/datalogger.git
    - dataQ.py → temperature, voltage, and serial data collected and saved into a CSV file
    - plotter.py → live plotting and plots at end of data logging run created and saved here
    - main.py → functions as a config file for the user to specify their settings
    - start-logging.sh → sh file necessary for running the plotter and dataQ files simultaneously

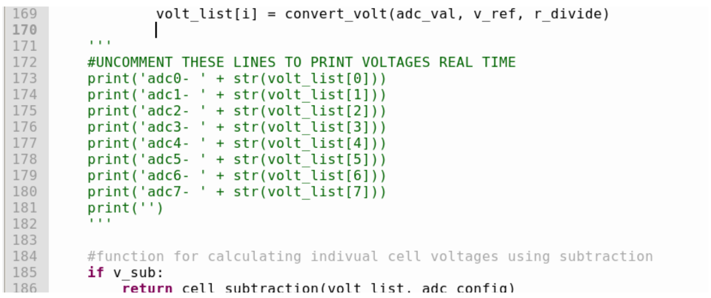
Calibrating ADC Input:

- Due to variation between the RPi Hat circuits, **calibrating the ADC is necessary**
- The middle term(v_ref) of the ADC tuples in 'main.py' is used for calibration. In DataQ.py, v_ref is multiplied by the raw ADC value.
- 

```
    #first term is whether to take readings on that ADC pin
    #second term is voltage reference used by ADC
    #third term is voltage divider coefficient
    ADC_0 = (True,4.987,4)
    ADC_1 = (False,4.953,4)
    ADC_2 = (False,4.900,4)
    ADC_3 = (True,4.927,4)
    ADC_4 = (True,4.95,4)
    ADC_5 = (False,4.95,4)
    ADC_6 = (False,4.95,4)
    ADC_7 = (True,4.95,4)
```

Below is the process I used to find an accurate value for v_ref:

- Using a trusted power supply, apply a voltage near the expected level you will be measuring on that specific ADC input channel.
- Turn on printing by uncommenting the lines of code in dataQ.py seen below



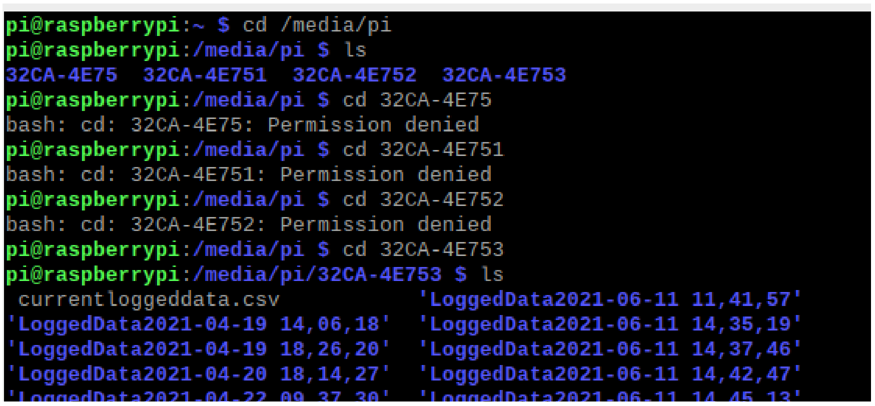
```
            volt_list[i] = convert_volt(adc_val, v_ref, r_divide)
            |
        '''
        #UNCOMMENT THESE LINES TO PRINT VOLTAGES REAL TIME
        print('adc0- ' + str(volt_list[0]))
        print('adc1- ' + str(volt_list[1]))
        print('adc2- ' + str(volt_list[2]))
        print('adc3- ' + str(volt_list[3]))
        print('adc4- ' + str(volt_list[4]))
        print('adc5- ' + str(volt_list[5]))
        print('adc6- ' + str(volt_list[6]))
        print('adc7- ' + str(volt_list[7]))
        print('')
        '''

        #function for calculating indivual cell voltages using subtraction
        if v_sub:
            return cell_subtraction(volt_list, adc_config)
```

- at the command prompt, enter:
  sudo python3 -c ‘import main; main.run_main()’
    - by running this and not the bash script, live plotting does not take place. This is useful because when calibrating the ADC you must run the program many times to hone in on the right v_ref value. Closing all the live plotter windows gets annoying when doing this. Hopefully Michael and Will add auto closing of the plots, so this step is no longer necessary
- adjust v_ref in main.py until the value that is printed matches the power supply

USB Drive:

- path_to_save in ‘main.py’ is where the user specifies the the location that all files will be saved
- Saving to a USB drive is useful but finding the correct path can be tricky
- Here are the steps to manually find the path: (An example image of this can be seen below)
    1. at the command prompt, enter **cd /media/pi**
    2. at the command prompt, enter **ls**
       this lists all the USB devices
    3. at the command prompt, enter **cd __fill in USB device name__**
       now keep on entering device names until Permission is not denied
    4. at the command prompt, enter **ls**
       if the files listed match the ones in your desired location you know you’ve found the name of the correct USB device
- The last step is to change path_to_save in ‘main.py’ to
  ‘/media/pi/__fill in USB device name__’



```
pi@raspberrypi:~ $ cd /media/pi
pi@raspberrypi:/media/pi $ ls
32CA-4E75  32CA-4E751  32CA-4E752  32CA-4E753
pi@raspberrypi:/media/pi $ cd 32CA-4E75
bash: cd: 32CA-4E75: Permission denied
pi@raspberrypi:/media/pi $ cd 32CA-4E751
bash: cd: 32CA-4E751: Permission denied
pi@raspberrypi:/media/pi $ cd 32CA-4E752
bash: cd: 32CA-4E752: Permission denied
pi@raspberrypi:/media/pi $ cd 32CA-4E753
pi@raspberrypi:/media/pi/32CA-4E753 $ ls
 currentloggeddata.csv           'LoggedData2021-06-11 11,41,57'
'LoggedData2021-04-19 14,06,18'  'LoggedData2021-06-11 14,35,19'
'LoggedData2021-04-19 18,26,20'  'LoggedData2021-06-11 14,37,46'
'LoggedData2021-04-20 18,14,27'  'LoggedData2021-06-11 14,42,47'
'LoggedData2021-04-22 09,37,30'  'LoggedData2021-06-11 14,45,13'
```

- if still experiencing problems saving to the USB drive try manually mounting the USB drive to the Raspberry Pi. The link below walks you through this. http://raspi.tv/2012/mount-a-usb-flash-drive-on-raspberry-pi