BASE Data Logger

Axel Pontén

BASE-CERN Summer Student Program 2019

Table of Contents

Introduction 1
What is BASE? 1
Summary 2
Old Data Logger 2
Memory leak 2
Object Oriented Data Logger 2
..... 6
Temperature fluctuations 9
Day/Night Cycle & Air Conditioning 9
Fluctuations in Experimental Data 11
Relays for Cryogenic Temperature Probes 12

Introduction

What is BASE?

The Baryon Antibaryon Symmetry Experiment (BASE) is a small sized experiment in the Antiproton Decelerator (AD) at CERN which measures the magnetic moment and charge to mass ratio of individual protons and antiprotons to very high precision using a multi-penning trap. The current precision of the measured magnetic moment and charge to mass ratio is 1.5 per billion and 69 parts per trillion, respectively. Since the experiment has such high precision, it is very sensitive to environmental noise and fluctuations. To keep track of the environment in the experimental zone, BASE uses a data logger written in LabVIEW. The data logger is kept on a computer in the experimental zone and is written by the experimentalists themselves. Below is a picture of the data logger front panel.

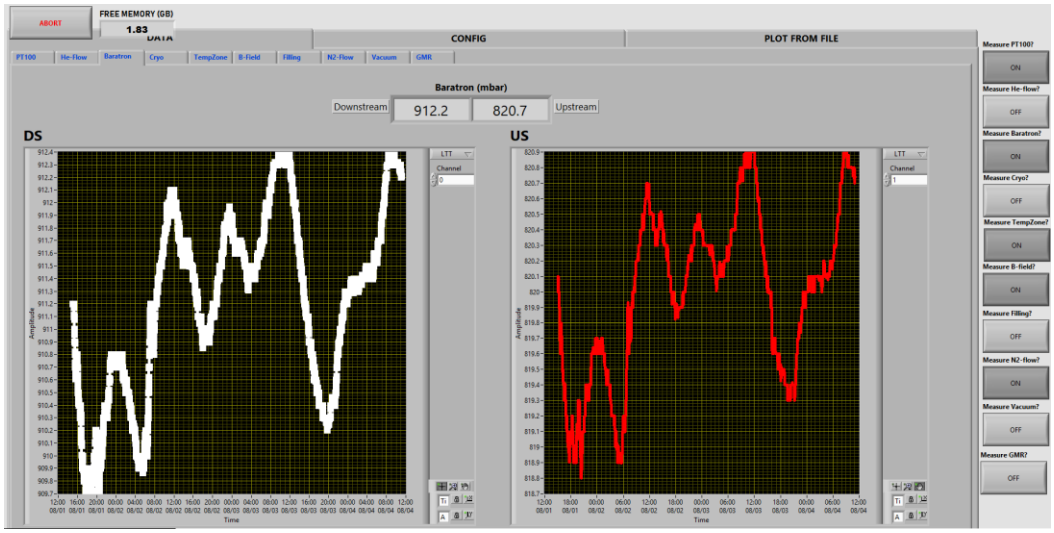

Figure 1. Front panel of data logger

Summary

My work this summer was related to BASEs data logger. Initially my project was to solve a memory leak in the logger code, but after familiarizing myself with the code I ultimately decided to rewrite the entire data logger in order to prevent future errors since the program had low reuse of code and no encapsulation. The new data logger I wrote was object oriented and made inserting new devices and debugging quicker and easier. After finishing the new data logger - which was most of my work this summer - I analyzed temperature fluctuations in the experimental zone and designed and built a relay circuit for our cryogenic temperature probes. Throughout the summer I also worked on troubleshooting a helium flowmeter connected to our cryogenic system and built a lab table, but these are not covered in this report.

Old Data Logger

The old logger had two prominent issues before I started my work:

1. Memory leak which caused the program to slow down over time.
2. Low reuse and encapsulation of code, which made troubleshooting and inserting new instruments more difficult and time consuming.

Memory leak

The cause of the memory leak was related to the arrays which stored the plot data. During data acquisition, new data was added to the plotting arrays using the *Build Array* function in LabVIEW, which allocates new memory when needed for the array to grow dynamically. For some unknown reason, LabVIEW kept allocating excessive memory, and after a few hours the memory of the computer was full and the program slowed down.

To solve this problem, the dynamically growing array was substituted with a pre-allocated one. I initialized an array of fixed size with NaN at the start of the program, and then replaced those values as new data was gathered. This ensures memory is constant. However, this leads to the problem that when the array is full, old data will be lost. To solve this I created two types of plot arrays. The first one is a Real Time (RT) array which will lose old data as new data is gathered, and the second one is a Long Time Trend (LTT) array which is a moving average of the gathered data. Once the LTT array is full, it averages pairs of data points which cuts the size in half. Once the array is averaged, the sample time for the LTT array (not the same as the instrument's sample time) is doubled and new data points will be added to a temporary array which is averaged once it's time for a new point to added to the LTT array, and then the average of that temporary array is inserted into the LTT array. In this way, the plot will always show information since the start of the measurement while still using a fixed size array.

Object Oriented Data Logger

The second problem with the old logger was the low reuse of code and lack of encapsulation. In the data logger, each device does the same three things: Gather data, plot data, write to file. For all devices, the plotting and file writing was done in the same way, the only difference being the data acquisition. However, each device would have its own code for plotting and file writing. This makes it harder to find bugs and errors, since there was not only one function to debug, but several. Additionally, adding new instruments took a long time since all code had to rewritten for every device.

To solve this problem, I decided to implement an object-oriented approach to the data logger. This was done using LabVIEW Object Oriented Programming (LVOOP). The basic unit in this program would be the *Measurement* class. Each device would belong to the same *Measurement* class and would share all its functions. However, since every device gathers data differently, each device would override the data acquisition function. With this design, reuse of code is maximized since the devices will share all common functionalities.

To encapsulate the code, I created a class for each functionality. For example, the data acquisition functions all belong to the *Instrument* class, and the file writing function belong to the *File* class. Each *Measurement* object contains objects from these secondary classes. In this way, if something is wrong with for example file writing, we only need to troubleshoot the *File* class since the other classes don't have access to those functions. To override the data acquisition, each device's *Instrument* object will belong to a unique child of the *Instrument* class which has overridden the DAQ-functions. The class hierarchy of the new logger is shown in the following figure.

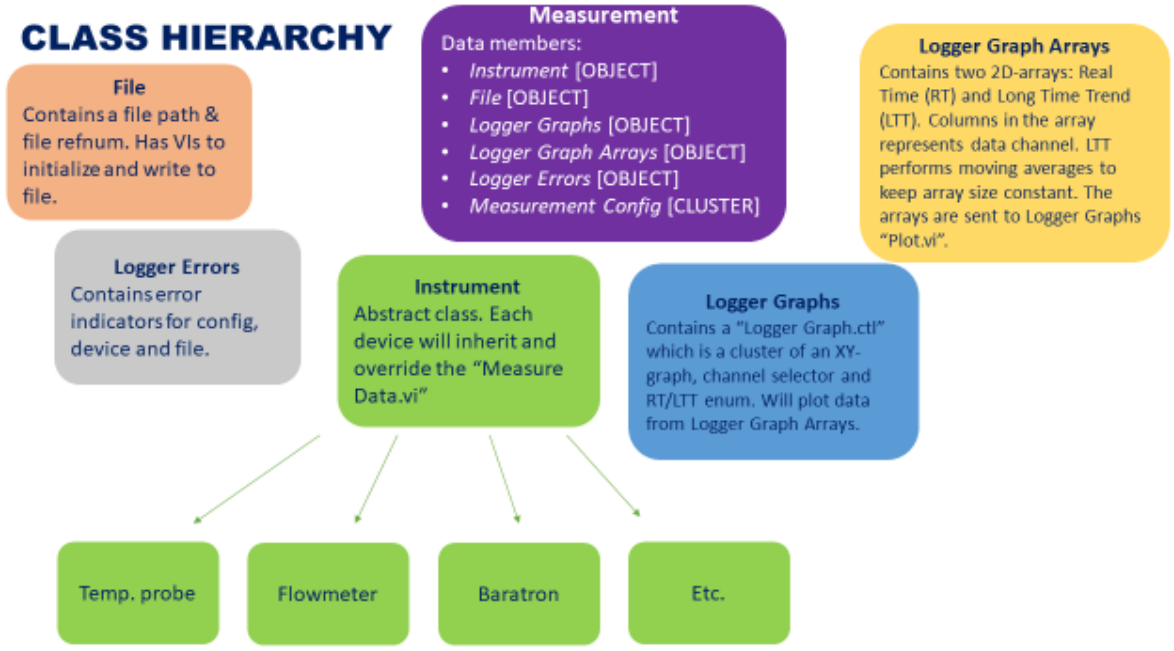

Figure 2. Class hierarchy of new logger

There is a main VI that contains the data logger front panel. There are four customized front panel controls that are used: *Logger Graph Control, Last Value Array, Config Cluster* and *Error Display.* These controls will together give most of the functionality needed on the front panel, if a specific device needs something else, then that can easily be added too (such as a VISA address control). On the block diagram of the main VI, references to the front panel controls together with an instrument object (or rather, child of instrument) will be passed into the DAQ-loop. Below is a schematic of the main VI followed by a screenshot of the block diagram.

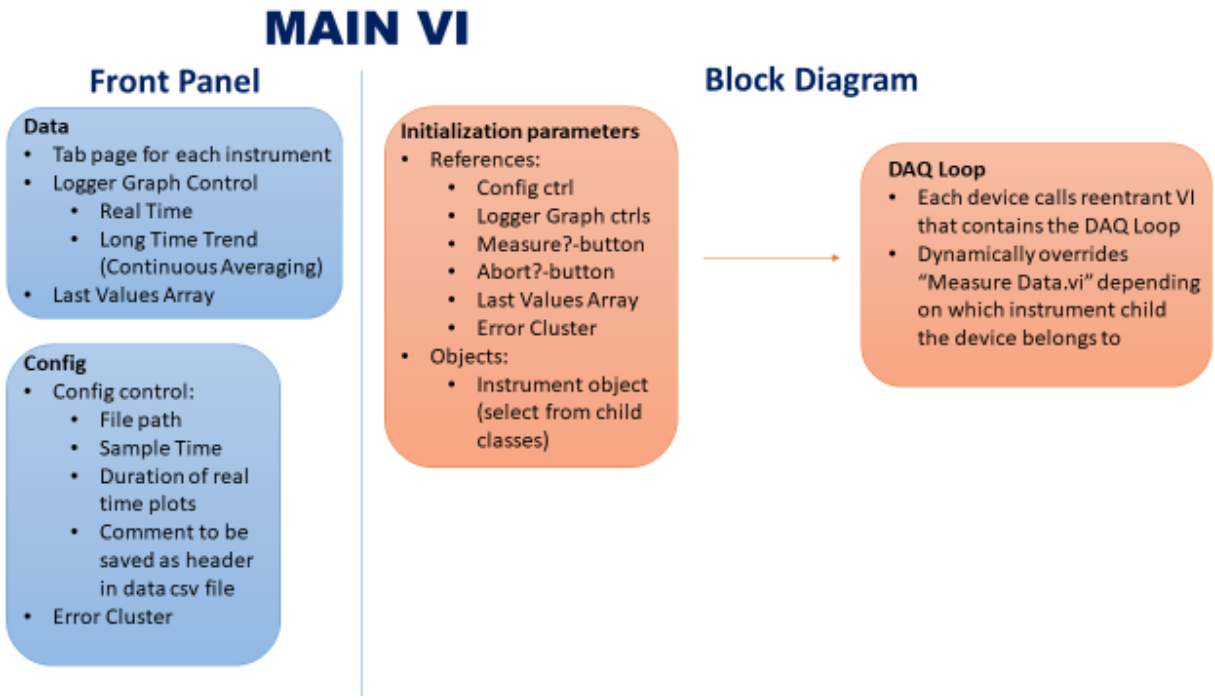

Figure 3. Structure of main VI in data logger

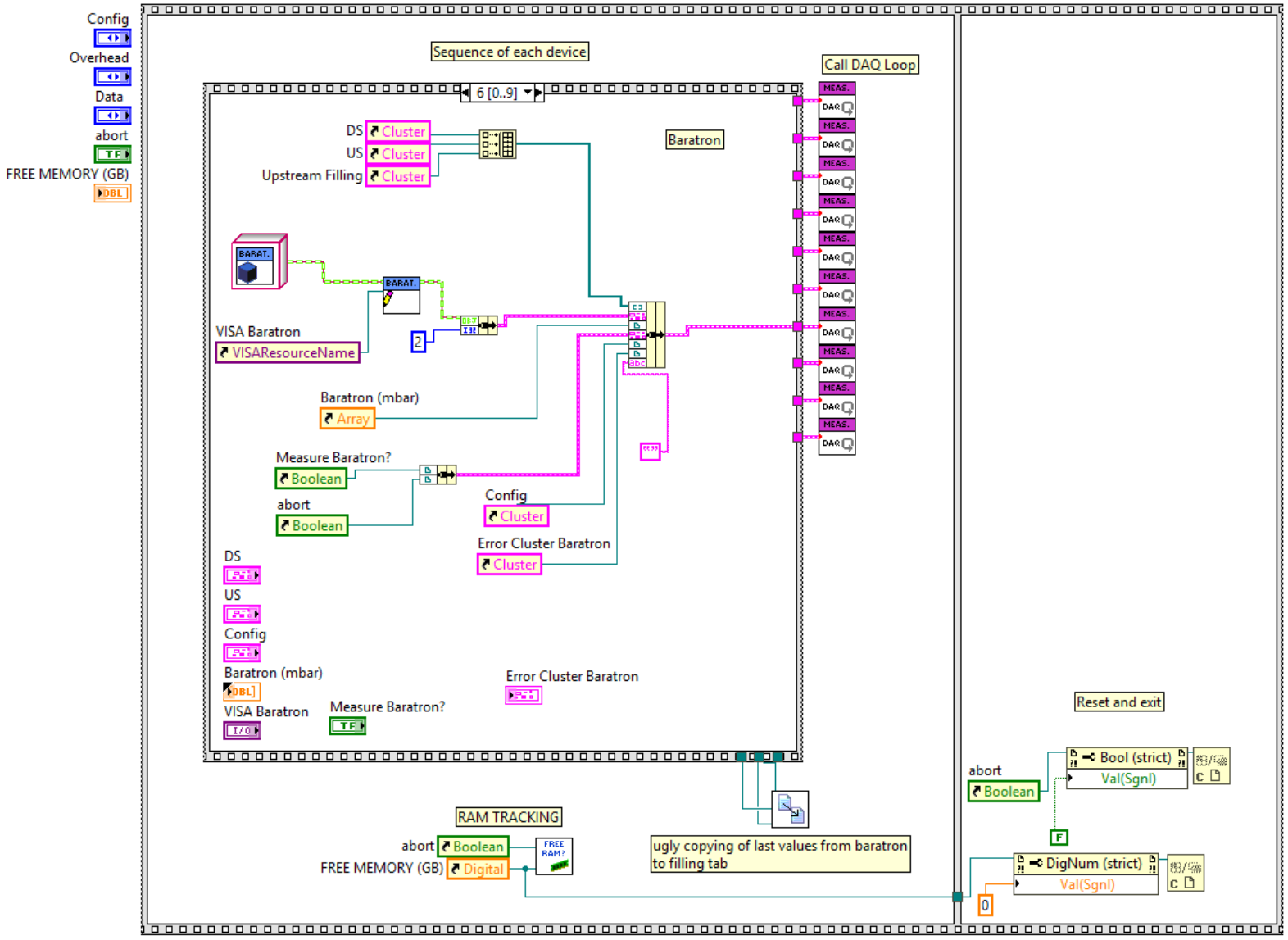

Figure 4. Screenshot of block diagram of main VI

In the DAQ-loop there are two nested loops. The inner loop will only run when *Measure?* is true. The purpose of the outer loop is to allow a measurement to be reset without having to restart the entire logger. When *Measure?* is true, the DAQ-loop will initialize the measurement from the parameters given in the config cluster. If there are any errors, such as “device not found”, config error (sample time set to zero etc.) or file error, then measurement will not start, and the user will have to fix the issues before trying again. If there are no errors, then the DAQ-loop enters the inner loop and data is acquired. During this time, the config is disabled since several bugs appear if the settings are changed throughout the measurement. If config is to be changed: turn off measurement, change config, and turn on measurement again. Below is a schematic of the DAQ-loop followed by screenshots of the DAQ-loop VI and the subVIs in the DAQ-loop.

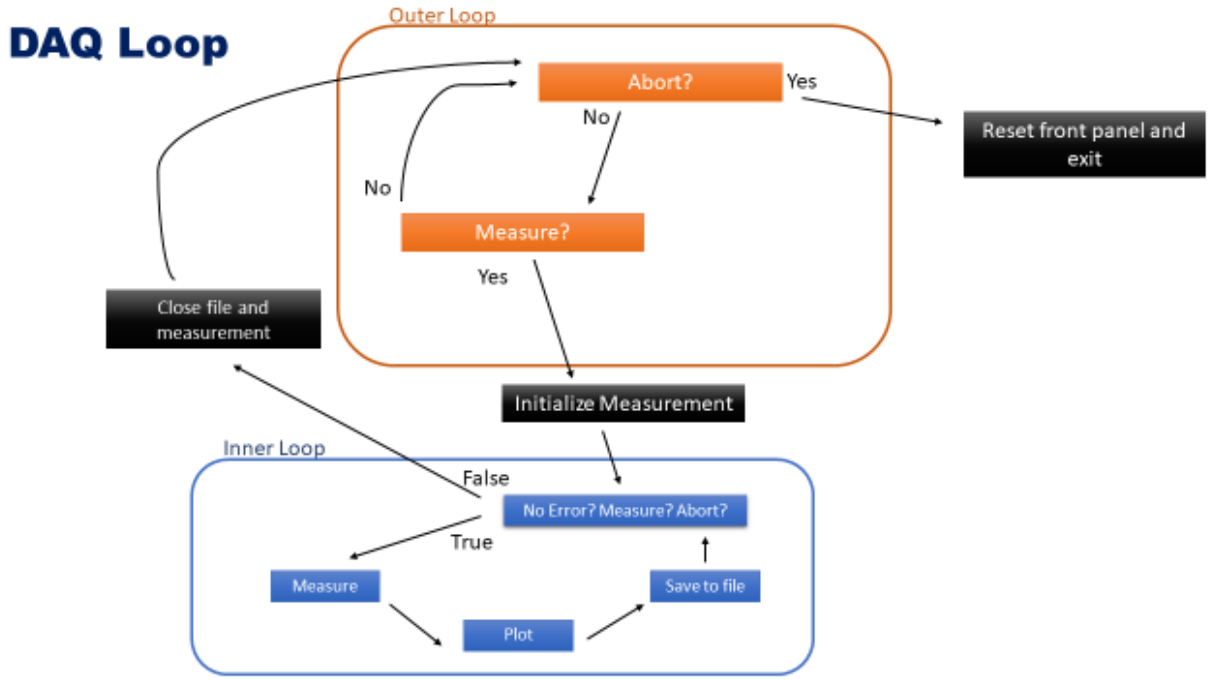

Figure 5. Diagram of DAQ-loop structure

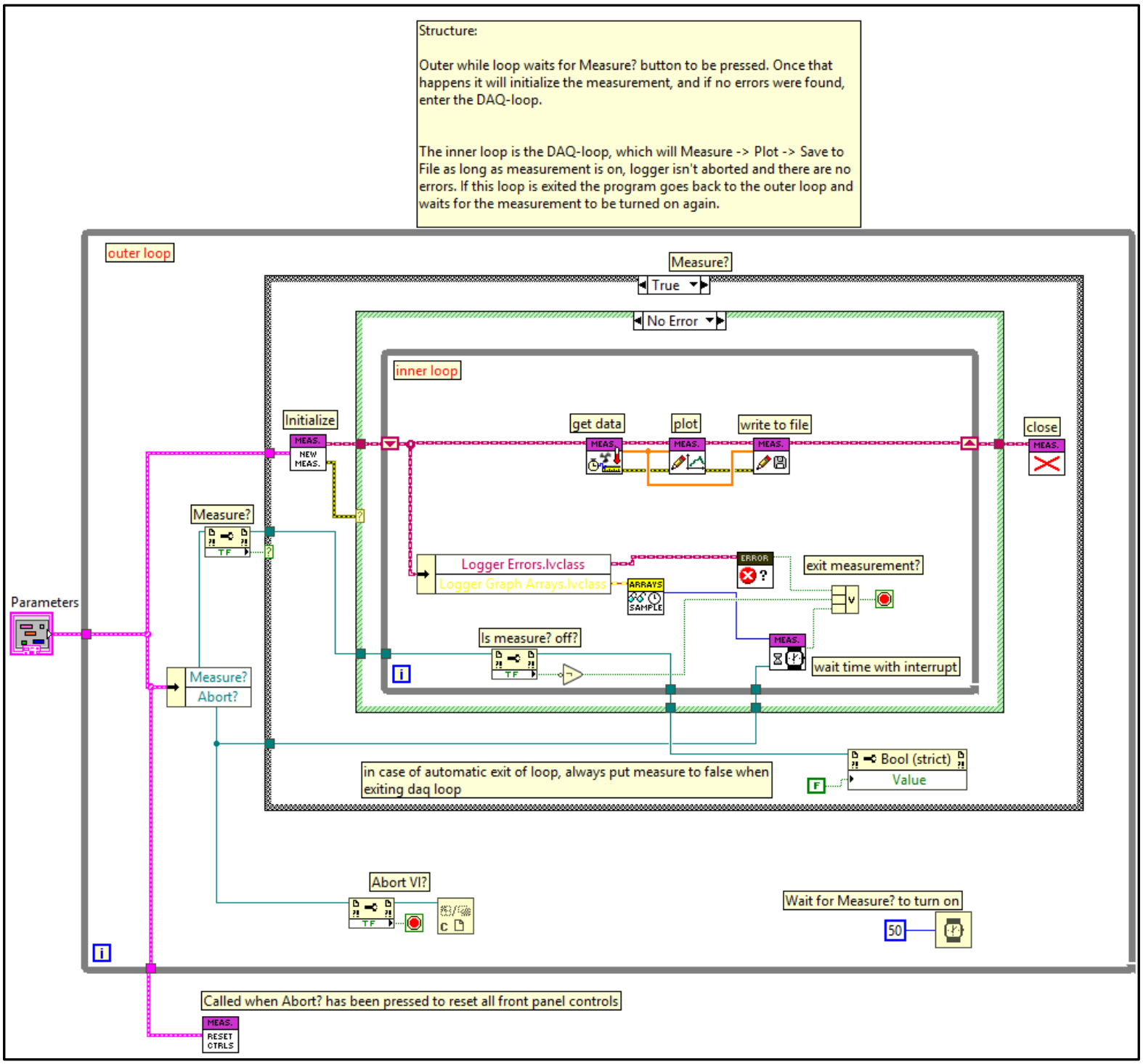

Figure 6. Screenshot of DAQ-loop block diagram

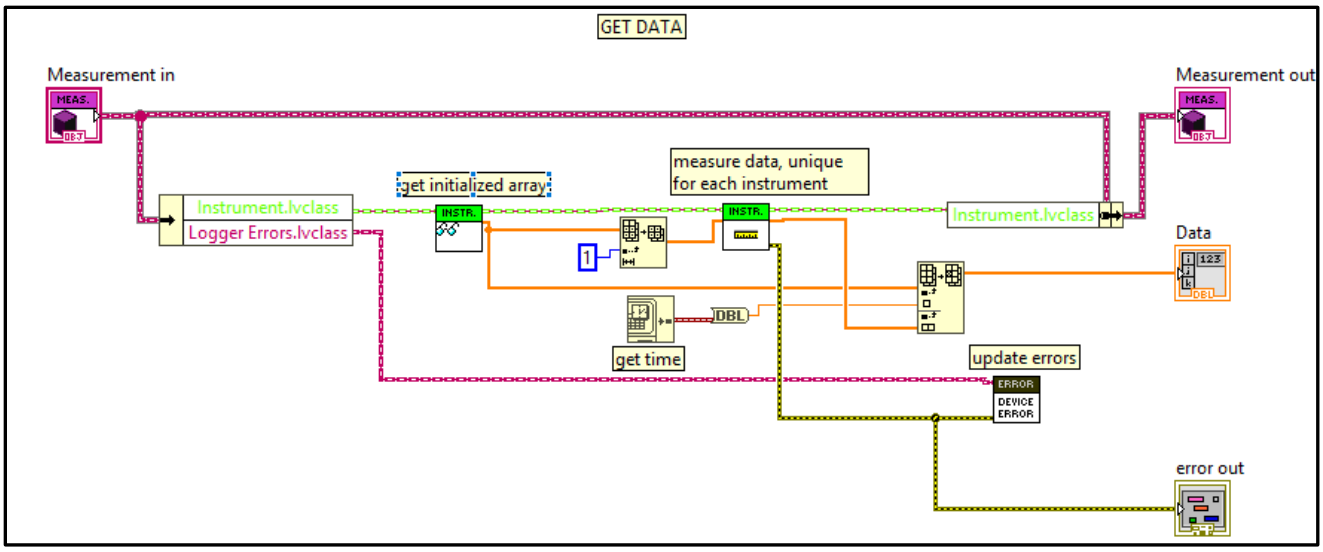

Figure 7. Screenshot of Get Data subVI

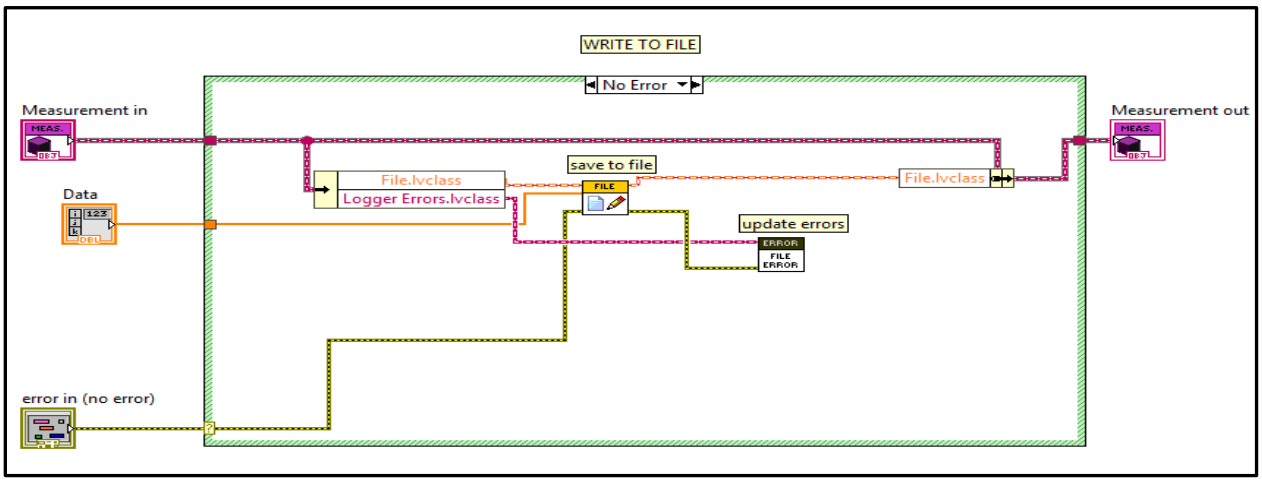

Figure 8. Screenshot of Plot subVI

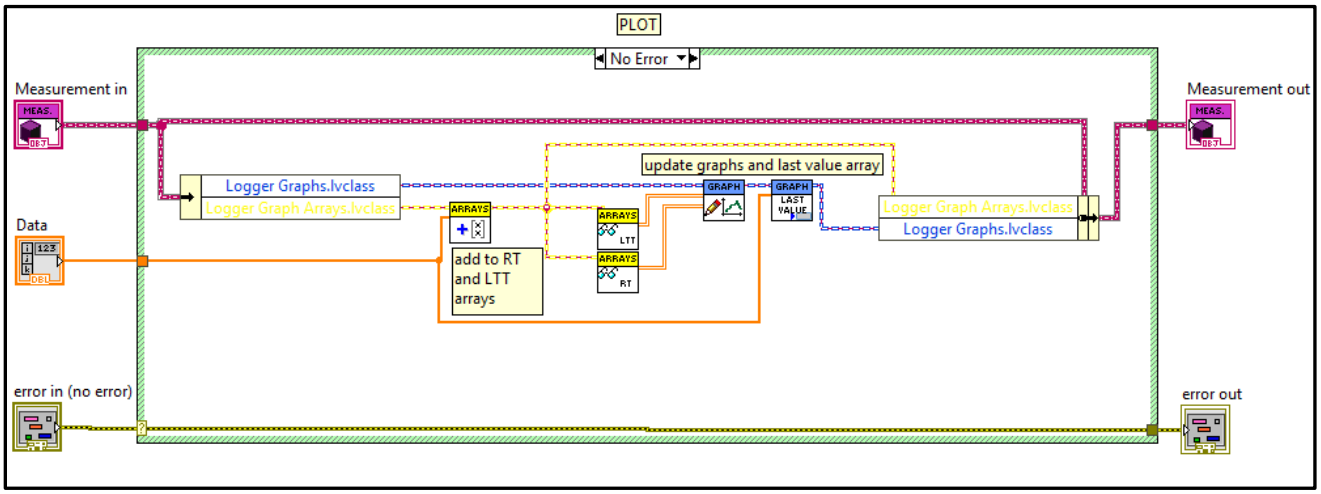

Figure 9. Screenshot of Write to File subVI

Adding a new device is now very convenient and quick! All that needs to be done is to create a new child class of instrument for the device, override Initialization/Measure Data and add a new front panel tab. File writing, error handling, and plotting is already written and taken care of.

Temperature fluctuations

Day/Night Cycle & Air Conditioning

After finishing the new data logger, I started analyzing the room temperature of the experimental zone using the data from the data logger. In particular, I analyzed temperature over the weekend 19-22 July. Below is the raw data.

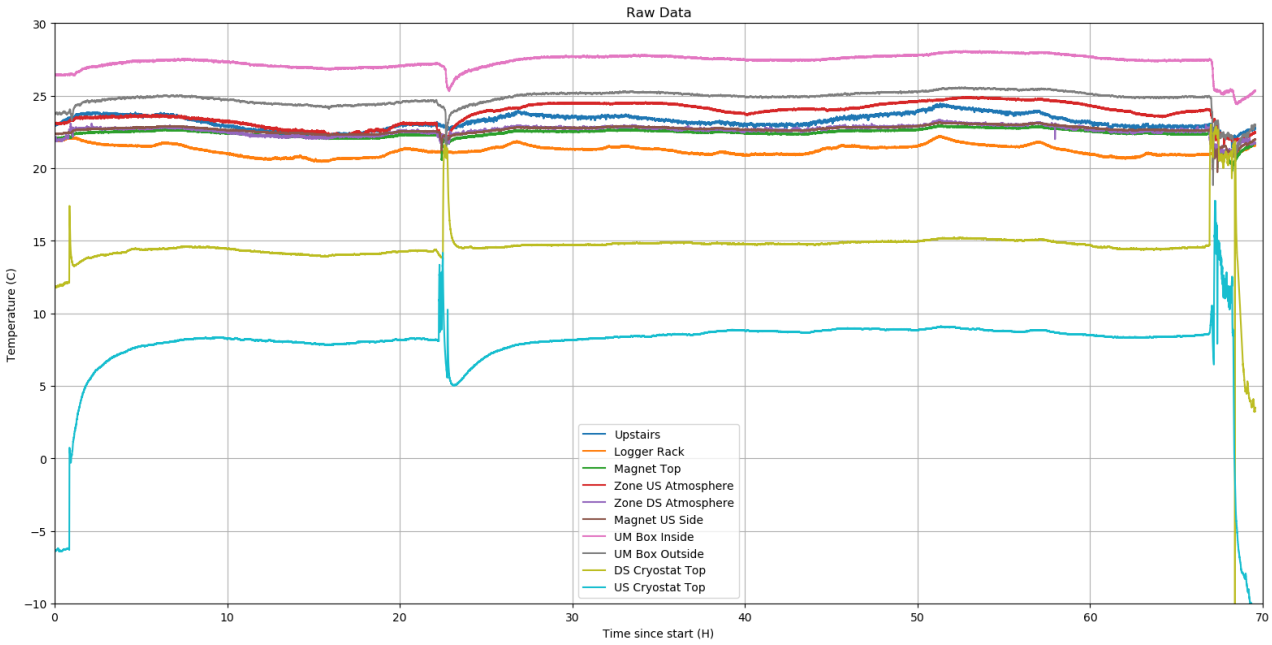

Figure 10. Raw temperature data in zone from Jul 19-22

First, I removed the data from the cryogenic filling time period (22-25 h and 67-70 h) due to the fluctuations that arise from the procedure. To get a better sense of the shape of the data I will remove DS & US Cryostat top from the plot, since they are not on the same temperature scale as the other thermometers.

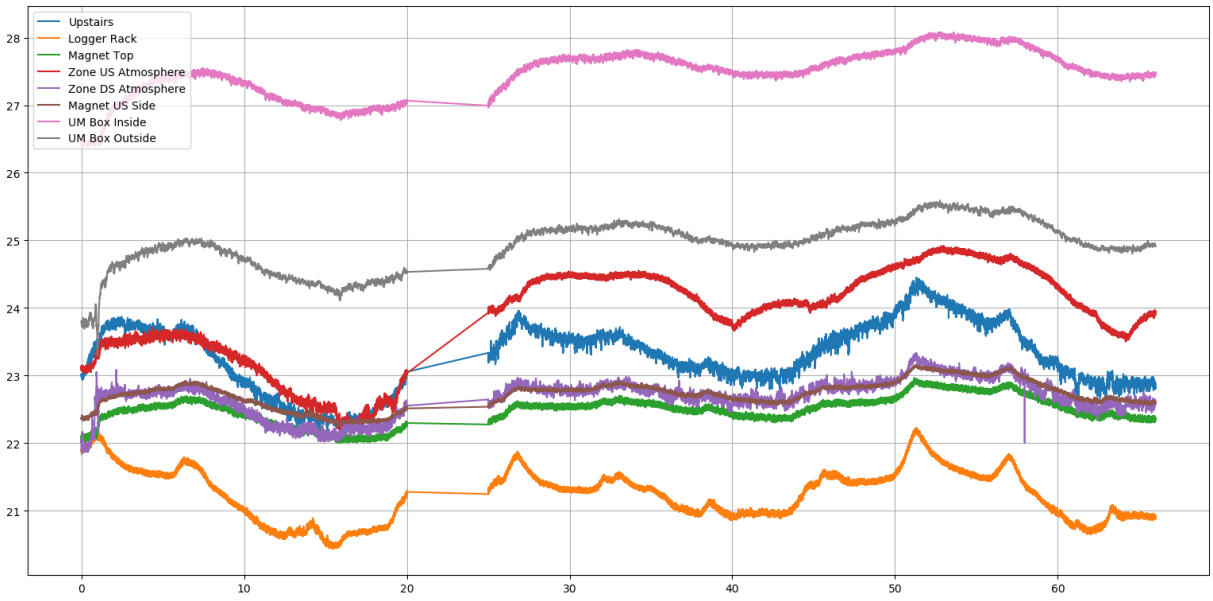

Figure 11. Raw data without fillibng and cryostat temp

I fitted a sine curve with a linear drift to each data set and subtracted that fit from the data to adjust for day/night cycle. Below is a plot of the logger rack data with the fit (blue and green) and after subtracting the fit (orange). Fit parameters are seen in the yellow box. The period of the sine curve is 24.6 hours, which roughly agrees with a day/night cycle.

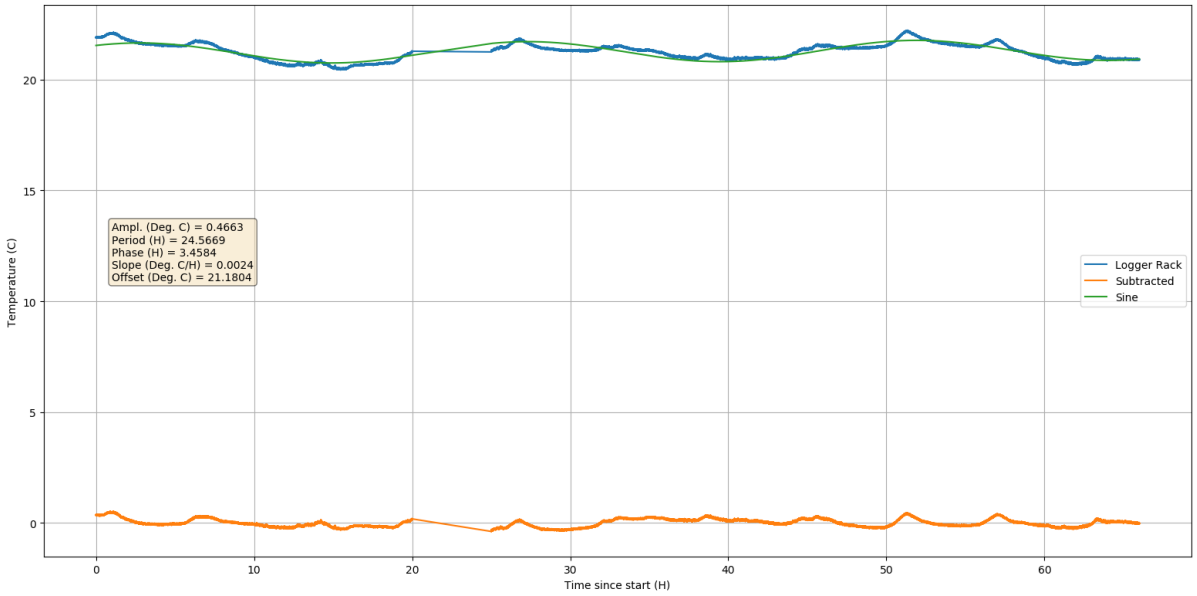

Figure 12. Logger Rack data fitted and subtracted fit

Zooming in on the adjusted data (orange) we see the following.

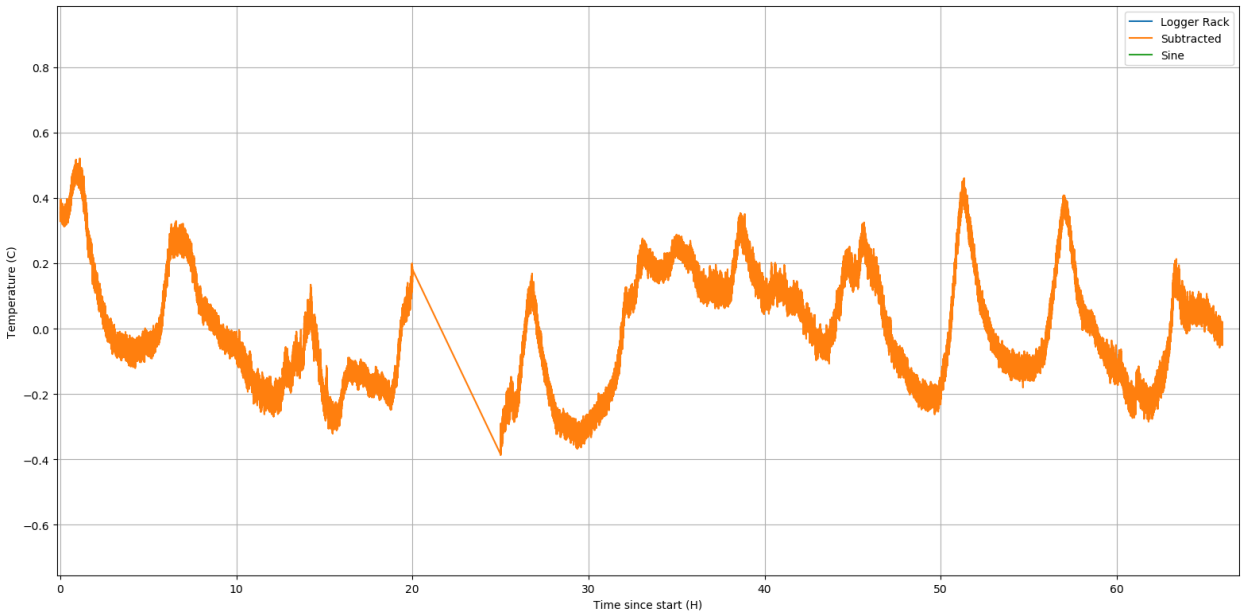

Figure 13. Logger rack data after subtracted fit

After removing day/night fluctuations, we see clear spikes in the temperature. We concluded that the spikes in the data seen above are from the air conditioning cycles in the AD-hall.

Fluctuations in Experimental Data

After one overnight measurement on July 29-30, one experimental dataset had fluctuated significantly. To see if this was due to temperature fluctuations, I plotted temperature against the experimental data. To scale the data and temperature to each other, I subtracted the average of each dataset and normalized it by dividing each point by the min-max range of the dataset. Below is a plot of the magnet top temperature and experimental data, scaled.

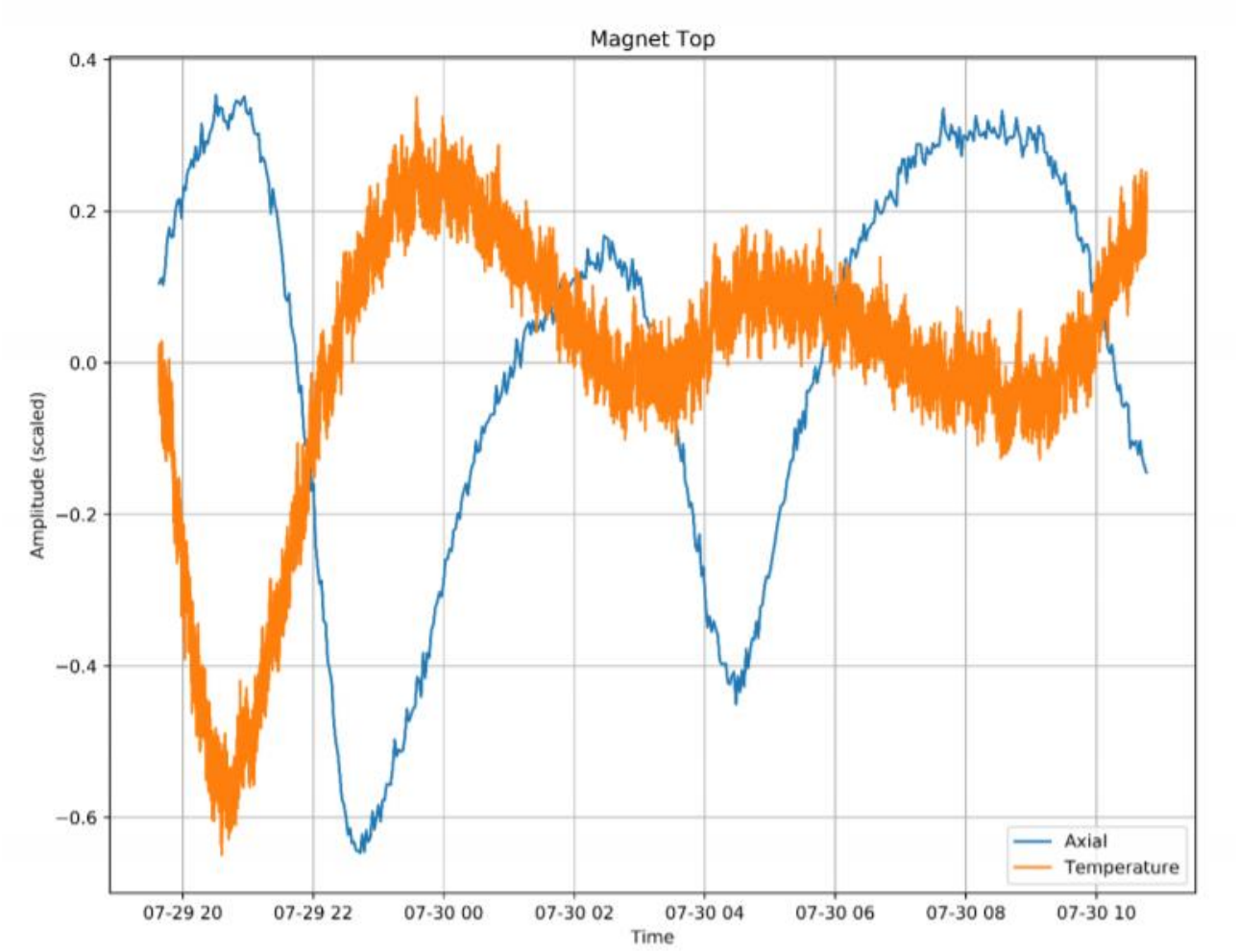

Figure 14. Fluctuations in experimental data and temperature of magnet top, both scaled.

There is clearly some type of inverse relation between the experimental data and the temperature. With this information, we might take future action to insulate parts of the experiment or correct for temperature fluctuations.

Relays for Cryogenic Temperature Probes

The experiment has temperature probes in the cryogenic system. When these are turned on, they cause noise in the experimental data. Previously, the temperature probe cables have manually been disconnected during measurements to avoid this noise. As a substitute for this tedious maneuver, I designed a circuit board with relays that can be inserted between the data logger and the cryogenic system to allow for automatic connection and disconnection of the temperature probes. I did not have time to manufacture this circuit, but my colleagues will print and install it after my time here is done. The schematic and board design of the relay circuit are shown below.

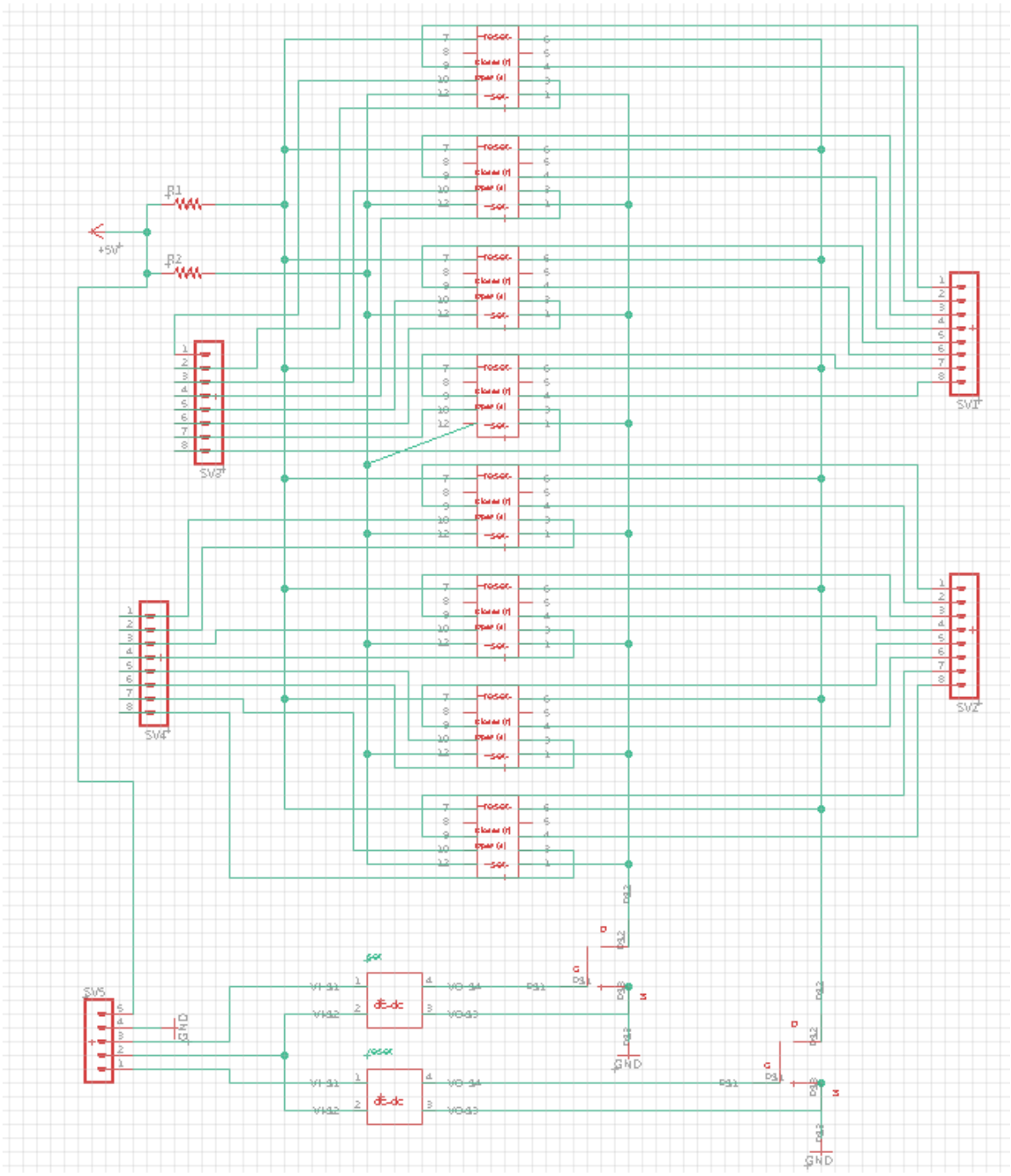

Figure 15. Schematic of relay circuit

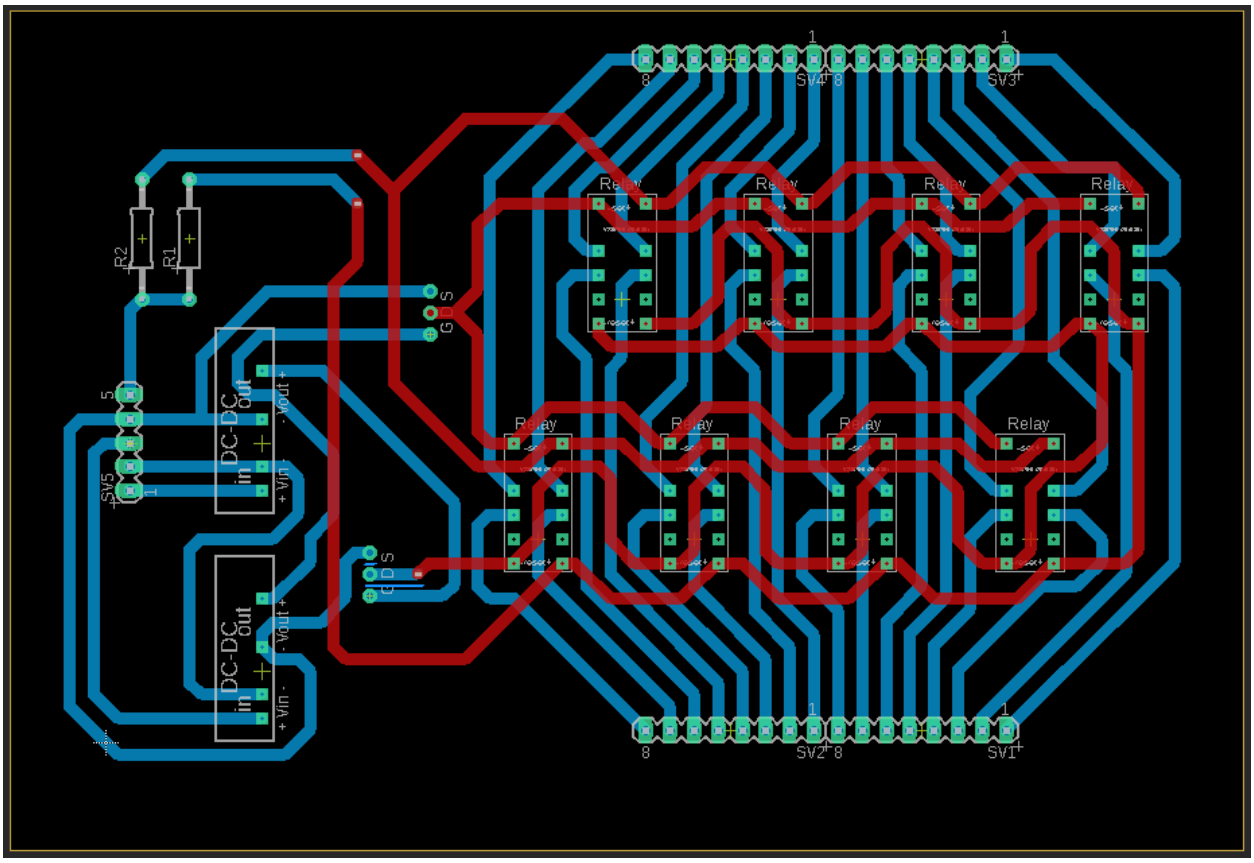

Figure 16. Board design of relay circuit