mvme (j_mvme) Development and Testing with Cf-252, Na-22, and Co-60

Jack Lawrence

Contents

1 Introduction

The world of data visualization and acquisition has seen significant advancements over the years. Especially in the realm of physics and radiation detection, the ability to monitor, analyze, and represent data efficiently is paramount. The following documentation servers as both an amendment to existing *mvme* documentation, detailing the improvements I made to the software, as well as a detailed report of the work I did at Lawrence Livermore National Laboratory (LLNL) in the [redacted].

At the core of this exploration are two key features: the 1D and 2D histograms. Histograms are how data is displayed in *mvme* and with the changes I made to these tools, researchers will be better able to monitor, collect, and analyse their experimental data. The main amendments I made involve the addition of the multi-window histogram feature that allows multiple hisograms (ie: detectors) to be monitored simultaneously. These views, essential for real-time analysis of large input sets, have been augmented to enhance their flexibility and applicability. Notably, the 2D view demonstrates significant strides in data representation, considering the prior limitations of *mvme* in this dimension.

However, software is only as good as its real-world application. To test the system's mettle, a series of experiments were conducted. Using varying detector setups, from silicon detectors to liquid scintillators and photomultiplier tubes (PMTs), and from high purity germanium (HPGe) detectors, I challenged the software's ability to receive, allocate, and visualize data. Tests were conducted using Californium-252, Sodium-22, and Cobalt-60 in collaboration with NIM electronics and detectors. The Cf-252 and Co-60 samples were primarily used to test the 1D histogram features and the Na-22 was used to test the 2D histogram features via coincidence measurements.

Finally, the results of the tests are discussed and known bugs are reported. From ensuring software robustness to ensuring detector efficiency, this exploration captures the meticulous efforts that go into enhancing the user experience in the realm of radiation detection and data representation.

2 DAQ Software Installation

The current DAQ setup involves the use of two different software packages. The first and most primary software is *mvme* (also called *j_mvme*) - this is the software that does all of the data collection and sorting. However, a second software is required as a companion to ensure that the modules are up-to-date: *mvp*. This is a simple software tool that should be installed on at least one DAQ computer; it allows the MVLC and MDPP modules to be updated with novel firmware.

2.1 Installing *j_mvme*

Installation steps for *mvme*/*j_mvme* on a Linux machine running Debian 10 are detailed below. Installation steps for *mvme* on other machines can be found at `https://github.com/flueke/mvme`. Note that *j_mvme* is currently only available at LLNL and has not been distributed publicly at the time this report was written.

2.2 Step-by-Step Guide

Start by running the following command to install the proper dependencies.

```
$ apt-get install build-essential cmake libboost-all-dev qt5-
   default qtbase5-dev-tools libquazip5-dev libqwt-qt5-dev
   zlib1g-dev libusb-dev libqt5websockets5-dev ninja-build
   libgraphviz-dev
```

Next, fork a copy of *mvme* form the github listed above or acquire a copy of *j_mvme* source code from LLNL; this document uses *j_mvme* nomenclature from here on. Once the *j_mvme* source code is downloaded onto your Linux machine, proceed with the following installation steps:

```
$ cd j_mvme
$ mkdir build
$ cd build
$ cmake -GNinja -DCMAKE_BUILD_TYPE=Release -
   DCMAKE_INSTALL_PREFIX=~/local/j_mvme ..
$ ninja
```

If the program properly builds, execute the traditional *mvme* run command

```
$ ./mvme
```

which will open the *j_mvme* program.

2.3 Installing *mvp*

Installation steps for *mvp* on a Linux machine running Debian 10 are detailed below. Installation steps for *mvp* on other machines are not currently available, though `mesytec.com` provides a version for Windows and one should attempt a similar procedure if using an alternative machine is necessary.

Historically, installing *mvp* has been difficult because certain libraries used to build the program are out of date. By default, the machine will try to build *mvp* with its own, more up-to-date libraries - the mismatched versions will likely cause an `undefined symbol` error. Luckily, the program itself comes with the libraries it needs to build, one just has to configure the computer to use the correct libraries source. The following steps will ensure that *mvp* is properly installed and functioning.

Begin by downloading the *mvp* software from `www.mesytec.com/downloads/firmware%20updates/mvp/`. I recommend downloading `mvp-1.x.x-Linux-x64.tar.bz2`, but using another download option is just as fine. The goal is to extract the program folder from the `.tar.bz2`, so whichever way provides you with access to that folder, do it!

After you've got the file downloaded, open a new terminal and navigate to the directory that holds the download. Now we're going to extract the folder in the following manner:

```
$ tar -xvjf mvp-1.x.x-Linux-x64.tar.bz2
```

This will extract the file contents and put them in a folder with the same name as the `.tar.bz2` file, in the same directory. Below is a breakdown of the commands used:

`tar`: The command itself.
`-x`: Option to extract files.
`-v`: Option for verbose mode (displays a list of extracted files).
`-j`: Option specifying the use of bzip2 compression.
`-f`: Option to specify the filename of the archive to work with.

Alternatively, one can extract the program folder and send it to a different directory with the following command:

```
$ tar -xvjf mvp-1.x.x-Linux-x64.tar.bz2 -C /path/to/
    destination
```

which copies the extracted contents to a specified destination.

Once the program folder is accessible, all that remains is for the program to be executed. In order to ensure that the program uses its own libraries, a command must be run before the program can be executed. Below, I run this "calibration" command in the same line as the program execution command:

```
$ LD_LIBRARY_PATH=/path/to/mvp-1.x.x-Linux-x64_directory/lib:
    $LD_LIBRARY_PATH /path/to/mvp-1.x.x-Linux-x64_directory/
    bin/mvp
```

This alone will open *mvp* with the correct libraries. Although, this specification of library usage will be reset every time a new terminal session is opened. For this reason, it is most efficient to make this permanent in the form of an `alias`:

```
$ sudo nano ~/.bashrc
```

This command opens the Bash shell script that initializes each shell session - putting commands here will make them permanent. Adding the `alias` example below to this file will allow you to open *mvp* in any terminal session.

```
$ alias mvp_update="LD_LIBRARY_PATH=/path/to/mvp-1.x.x-Linux-
    x64_directory/lib:$LD_LIBRARY_PATH /path/to/mvp-1.x.x-
    Linux-x64_directory/bin/mvp"
```

The name of the alias is up to the user, here i use `mvp_update` to call the program. Make sure to save `'.bashrc'` file updates and restart the kernel to ensure the command is put into permanent effect.

2.3.1 Port Permissions

After *mvp* is opened and the firmware update file is loaded, you may encounter a `Permission Denied` error - this is a reference to the USB port permissions which may need to be changed so that the firmware can pass to the module. To update the USB port permissions execute the following command:

```
$ sudo usermod -a -G dialout $USER
```

This allows non-root users to access the `dialout` group, which includes the USB ports (called ttyUSB#).

3 File Hierarchy

One of the biggest hangups with developing *mvme* is understanding the contents of the source files, where certain features are created, and how the files communicate with one another. The following section serves as an overview of relevant file locations, prominent files, and insight into the file network. I'll begin by including a list of files I manipulated while developing *mvme*; all files are located in the `src` directory.

```
analysis/analysis.cc
analysis/analysis.h
analysis/analysis_ui.cc
analysis/analysis_ui_p.cc
analysis/analysis_ui_p.h
analysis/a2/a2.cc
analysis/a2/a2.h
analysis/a2_adapter.cc
histo1d.cc
histo1d.h
histo1d_widget.cc
histo1d_widget.h
histo2d.cc
histo2d.h
histo2d_widget.cc
histo2d_widget.h
mvme.cpp
analysis/ui_eventwidget.cc
analysis/ui_eventwidget.h
```

The above list should provide some insight as to which files contain the most relevant capabilities when developing mvme. As you can see, the majority of the files are located directly in the `src` directory or nested in the `analysis` directory. This should make intuitive sense considering that most adjustments made thusfar relate to data analysis and revolve around the *Analysis* window. Most of these files create or contribute to the creation of the *Analysis* window, most notably `analysis/ui_eventwidget.cc`. This file is the base of the *Analysis* GUI: it constructs the processing levels and allocates the histogram sink locations where data is sent to be displayed. This file is also home to most of the buttons and options available in the *Analysis* window; changes made to histogram creation and display will likely be made here.

The `analysis/analysis.cc` file is also extremely relevant as far as software development goes. While the previously mentioned file deals with the cosmetics of the *Analysis* window and the objects that the user interacts with, this file is the hidden or "behind-the-scenes" analog. In this file lies the functions performed in order to properly display data. For example: there is a `beginRun()` function for each of the sinks (1DHisto Sink, 2DHisto Sink, RateMonitor Sink, etc.) that is responsible for correctly connecting data inputs to the display (created by `analysis/ui_eventwidget.cc`), as well as constantly updating this datastream. This is all done in `analysis/analysis.cc`. The creation of the sinks themselves is a layer further down in `analysis/a2/a2.cc`, though this file is not as useful for most feature adjustments.

Other notable files include `mvme.cpp`, where all windows of the program are initialized, `histo1d_widget.cc` and `histo2d_widget.cc` which handle the specifics of each histogram widget, and of course all of the associated header files (`[filename].h`) which include most variable and class definitions.

4 Histograms

The following sections detail one of the most important aspects of *mvme* - the histograms. Histograms allow us to view the data we are collecting to draw conclusions about nuclear processes, gain insight into the subject of an experiment, and troubleshoot the DAQ system.

4.1 1D Histograms

The 1D histogram is currently the most flexible data viewing feature available in *mvme.* These histograms exist in the *Data Display* levels, located in the southern half of the *mvme Analysis* window, and can display both raw and processed data. What is referred to as "processed" data is any data that has passed through at least one calibration level. Calibration levels allow the user to apply mathematical operators to incoming data, or filter which counts are displayed on histos based on user-input criteria. For example: the input data can encounter a mathematical inequality, that gates the data based on a threshold (eg: 200keV) to filter low-energy background counts. The calibration features were not a focus of this software development mission, and more information is available in the official *mvme* documentation.

The raw and processed data lives in the northern half of the *Analysis* window - all data sets can be displayed as a histogram. To create a new 1D histogram, left click in the blank space of your target level and select **New ▷ 1D Histogram**, which will prompt you to choose the histogram settings. Note that the histogram **Input** can accept singular signals (eg: Channel 0) or collections of signals (eg: Channels 0-32); each option creates a singular histogram location but rears a collapsible tree with branches to select each individual histogram.

Each 1D histogram window looks identical, aside from the axes labels and the displayed data. The toolbar at the top of the window allows the user to select between a linear and a logarithmic Y-Axis, fit a Gaussian to the data, and estimate the exponential decay rate between two selected points. The user is also able to crop the data window and adjust the resolution of the data (2-16 bit). All histogram data can be saved, exported, cleared, and displayed in a `'.txt'` format easily with the toolbar, and histogram display information (ie: under/overflow, bin width, etc.) can be toggled. Each window also contains a scroll box that allows the user to click through the histograms available from the particular **Input**.

The data itself is displayed in the center portion of the window with the Y-Axis always displaying counts, and the X-Axis displaying a particular metric (ie: amplitude, time, energy, etc.). A small window with statistical information floats in the top right corner of the data display window.

The user interface of the *Analysis* window controls the creation of histograms as well as how their data is allocated; most of the source code for this is located in `mvme/src/analysis/ui_eventwidget.cc`.

Multi-Window View was a feature that I added to *mvme* and it works with both 1D (1) and 2D (2) histograms. It allows the user to view all histograms from a given **Input** in the same window, whereas before, only one histogram could be displayed at a time. This option can be selected by right-clicking a 1D or 2D histogram tree and selecting the corresponding Multi-Window viewer. The code that adds the Multi-Window View for 1D and 2D histograms can be found under "Multi-Window Implementation (1D)" in the same file `ui_eventwidget.cc` file described above. This feature is extremely useful and necessary for large experiments with many detectors, such as those that utilize the novel [redacted] array.

4.2 2D Histograms

The implementation of the 1D and 2D Multi-Window viewers was slightly different, considering that the flexibility of 2D histograms in *mvme* was extremely limited compared to that of 1D histograms. For the 1D case, I added a function into the file that allows for the histogram windows to populate within the same, larger window on an organized grid, while maintain a livestream of the input signals from each detector. By default, *mvme* only creates arrays of size 1, 2, 16 and 32, though this new function accommodates arrays of any size. Early testing of this feature allowed for mismatched array sizes to create a histogram tree, but these trees were not sustainable and crashed the program. Thus, the input arrays must be the same size, otherwise a warning will be triggered and the histogram tree will not be populated - this avoids the crash and prevents the creation of such unintelligible trees.

Translating this feature from 1D histograms to 2D histograms was a involved task. Before I began developing *mvme*, there was no such thing as a "2D Histogram Branch"

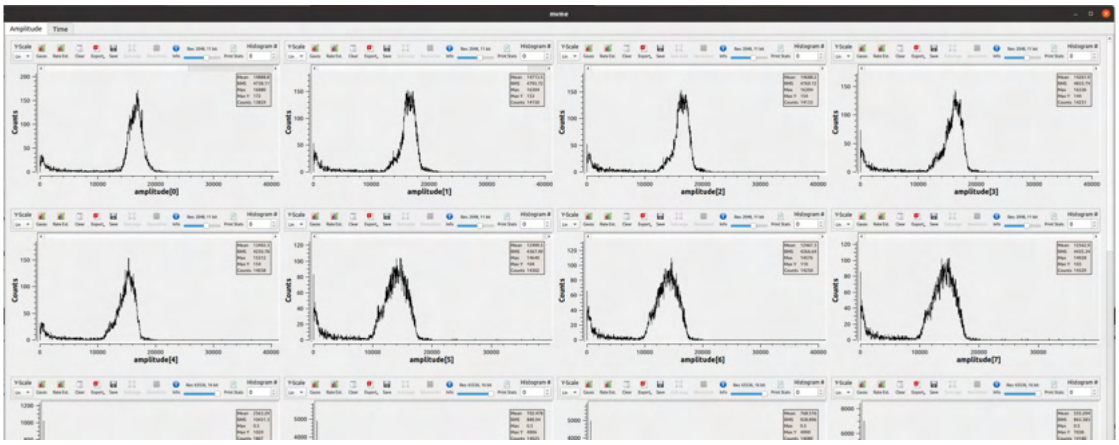


Figure 1: This is an example of the 1D Multi-Window View. Pictured are 8 different inputs from a 16-input source. The graphical content is arbitrary and was generated via radioactive source emulation (CAEN DT5800 Desktop Digital Detector Emulator). Note that this is an image from early testing and the numbers in the scrollboxes do not correspond with the displayed inputs - this was a bug that was later fixed.

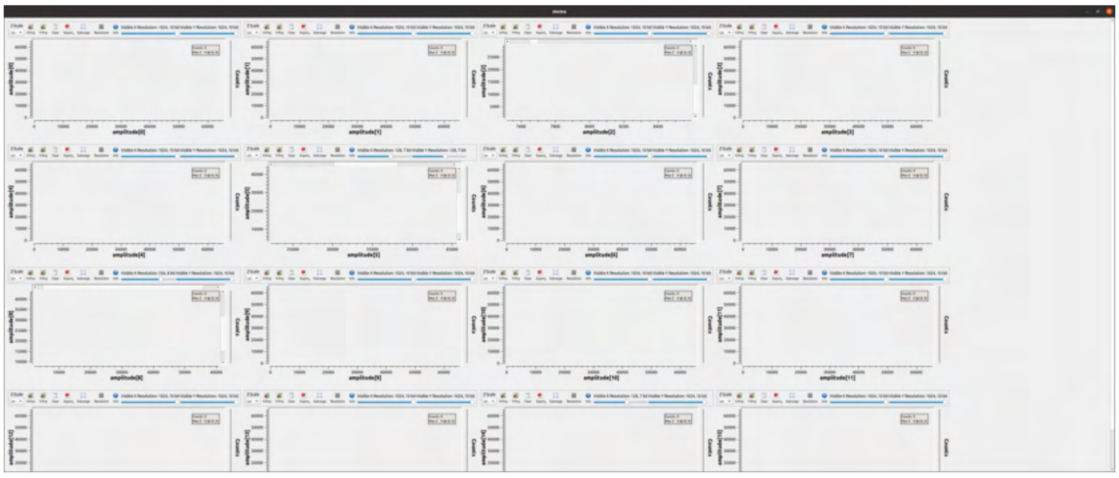


Figure 2: This is an example of the 2D Multi-Window View. Pictured are 16 different inputs from a 16-input source. The graphical content is blank, though each window operates independently as they do in the single-window view; note that the binning and axes ranges have been arbitrarily altered to emphasize this and elaborate on the feature's flexibility.

- in other words, each 2D histogram had to be created individually, whereas multiple 1D histograms could be created at once by selecting a multi-channel input. This was mainly because there was no way to select a multi-channel input when creating a new 2D histogram, so I first had to add this ability to the program. I ammended the memory allocation space so that when a new 2D histogram instance was populated, the feature could handle creating up to 32, 2D histograms at once. From there, I made it so that a new 2D histogram instance could accept both singular and array inputs.

To clarify, each 2D histogram instance takes two input source (rather than one like with 1D histograms). *mvme* then creates a 2D histogram, with each metric (ie: amplitude, time, energy, etc.) on one axis; the coincident counts are displayed in the graph space, colored from blue (low) to red (high) depending on the number of counts at a particular location. The toolbar is similar to that of 1D histogram widget, but with *X-Proj* and *Y-Proj* buttons that, when clicked, display the 1D histogram of each input. When the input sources are arrays, a 2D histogram is created from each corresponding array element. For example, say I have two array inputs of size 16 and called **Input_A** and **Input_B**. A 2D histogram instance created with these two inputs will create a tree of 16, 2D histograms, where the first branch is **Input_A_[0]** vs **Input_B_[0]**, the second is **Input_A_[1]** vs **Input_B_[1]**, and so on. Currently, these inputs cannot be adjusted for misalignment (ie: changed to **Input_A_[0]** vs **Input_B_[1]**), though the physical module inputs can be easily moved if necessary.

After I was able to successfully implement this adjustment to the software, I wrote a similar function to that which creates **1D Multi-Window View** to create the **2D Multi-Window View**. Like the histograms in the **1D Multi-Window View**, each 2D histogram in the multi-window view is independent of its siblings and each can be interacted with individually. This can be seen on the corresponding graphic (2), where I have randomly changed the resolution and axes ranges on some of the histograms to show they do not effect each other.

The addition of the **Multi-Window View** for each type of histogram, along with the improved flexibility of the 2D histogram capabilities in *mvme* improve the user experience and streamline the DAQ methodology. Researchers using my version of the software will be able to monitor dozens of detectors at once for analysis and troubleshooting. The ability of each widget to maintain independence in the **Multi-Window View** is extremely useful and gives the user a more customized experience.

5 Testing with Various Detectors

In order to test whether the updated version of *mvme* was properly receiving signals and allocating those signals to the correct locations, I setup a series of small experiments with the help of my labmates. The sections below describe the setup of three detectors arrangements: a silicon detector to measure the α-particle emission of a Cf-252 source with an activity of $3.44 \times 10^{-3}\mu$Ci , a gadolinium-loaded liquid scintillator and photomultiplier tube (PMT) for background discrimination, and two HPGe detectors for coincidence measurements using a Na-22 source with an activity of $1.04 \times 10^{-1}\mu$Ci. A HPGe clover detector and Co-60 source with an activity of 1.023μCi were also added for testing. It is important to state that the goal of this exploration was to test the reliability of software and electronics in preparation for nuclear expermentation, rather than performing detailed experiments with the isotopes. For this reason, the detector setups are functioning, yet rudimentary; signal measurements and spectral analysis was limited. A schematic for the first two detector setups is also shown below (3). Note that the clover detector and Co-60 source were added late in the project timeline, thus is not included in the original schematic below, though they follow the same setup procedure as with the single-crystal HPGes (just with different biases).

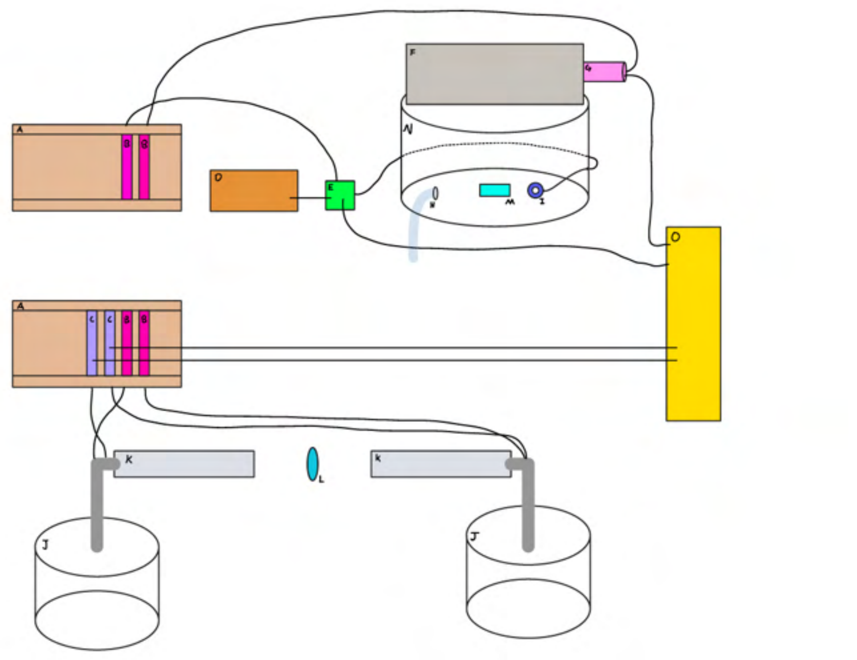
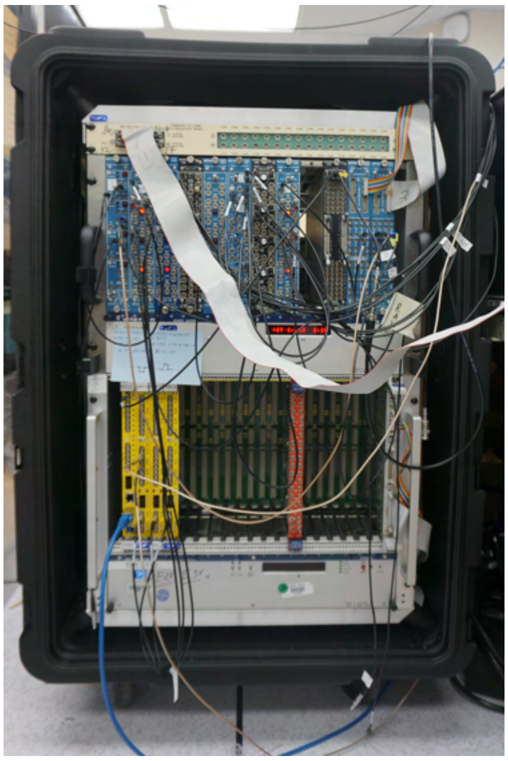


Figure 3: Above is the schematic for the primary detector setup, with the top half being devoted to the Cf-252 source and the bottom half devoted to the Na-22 source. High voltage connections bias the detectors and include a bias shutdown. NIM amplifier modules include a signal input and a preamp connection from HPGe detectors. Note that the preamps themselves are powered by +12V, and the voltage input is for the silicon detector only. Key: A – NIM Crate used to power high-voltage modules and amplifiers, B – high-voltage power supply, C – amplifier, D – +12V preamp power supply, E – preamp for silicon detector, F – liquid scintillator material, G – photomultiplier tube (PMT), H – port to vacuum pump, I – silicon detector, J – liquid nitrogen dewar, K – HPGe detector, L – Na-22 source, M – Cf-252 source, O – DAQ hardware interfaced with computer

5.1 Silicon Detector Setup

In order to study the resolution of *mvme*, along with its ability to accurately display energy spectra, I used an Ametek TB-018-150-200 Silicon Detector with 18 keV α resolution. The detector was placed in a steel chamber a few centimeters from the Cf-252 source and air was removed from the chamber via a scroll pump that maintained an internal pressure of $\sim 10^{-3}$ Torr. Maintaining near-vacuum pressure prevents air particles


(a) Front of the DAQ crate with a NIM rack (top) and a VME rack (bottom), both with cooling fans.

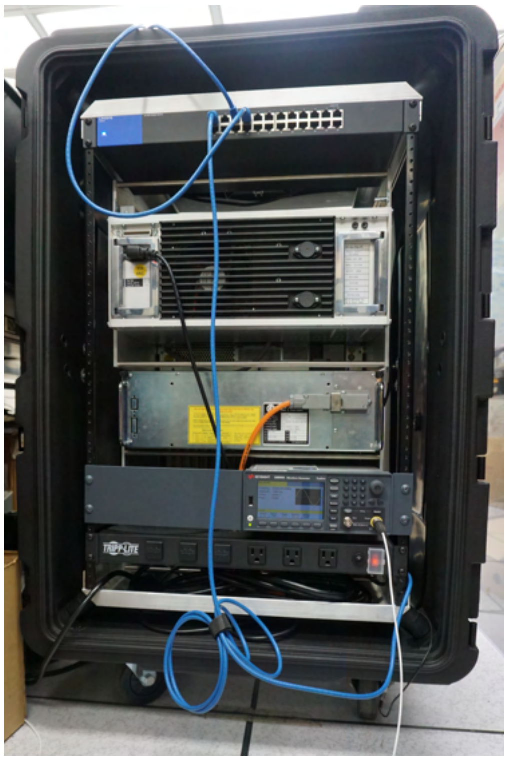

(b) Back of the DAQ crate with power strip, network bus, and waveform generator for clock signal.

Figure 4: Data acquisition (DAQ) crate. Not all modules present in the crate were used for testing.

from blocking α-particles emitted by the source. The chamber is also light-tight, which prevents background gammas from interfering with the detector and producing noise.

The silicon detector requires +30V provided from an external power source; the output signal is then attached to a preamplifier, powered by +12V externally, for impedance matching and initial amplification. From the preamp, the signal is fed through a ribbon cable to a Ribbon-to-LEMO module in the DAQ crate (4) and connected to one of the *mvme* modules (ie: the MDPP-16 SCP) in the VME rack with a LEMO cable. The detector setup was done alongside the liquid scintillator and PMT setup, because they detect radiation from the same source; both are show are show in Fig. 5

5.2 Liquid Scintillator and PMT Setup

And aluminium box filled with Gd-loaded liquid scintillating material was used in conjunction with the silicon detector to test the DAQ. in this experiment, the liquid scintillator is used for background discrimination. While it was most relevant in this scenario to see if the photomultiplier tube (PMT) signal came out clearly on the DAQ, discrim-

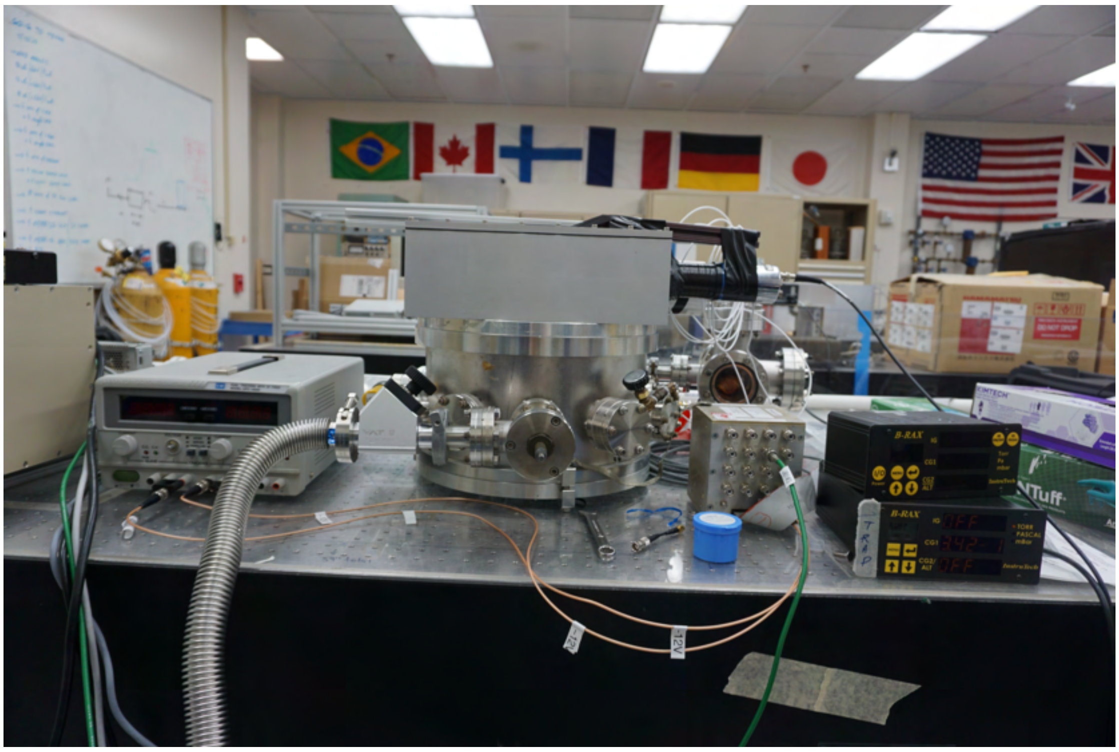


Figure 5: Pictured is the setup used to study Cf-252. It features a vacuum chamber holding the source and a silicon detector as well as a liquid scintillator placed on top of the chamber with an adjoining PMT.

ination can help decipher the type of the radiation from a source, whereas the silicon detector alone cannot; this is because the liquid scintillator produces light with energy corresponding to the the incident radiation that produces it.

The PMT is attached, light-tight, to a window on the liquid scintillator box with a special gel and black tape. As a further preventative against light pollution, the box, PMT, and chamber with Cf-252 source are covered in a black shroud. The signal from the PMT does not require external preamplification and is routed directly to the DAQ crate with a BNC-to-LEMO cable.

The signal from the PMT was first observed on an oscilloscope to ensure that the electronics were working properly. Unfortunately, it was discovered that the PMT was broken and did not output a signal, and could therefore not be used to test the DAQ equipment. The most likely cause for this was attributed to an over-voltage provided by the high voltage supply module, or improper biasing technique (ie: the bias was applied too quickly and fried the electronics). Conclusively, the PMT was mishandled prior to it being used in this experiment. Despite not being able to use the PMT signal, setting it up alongside the liquid scintillator begot valuable experience in handling lab equipment and connecting my knowledge of physics with actual experimental methods.

5.2 HPGe Detector Set

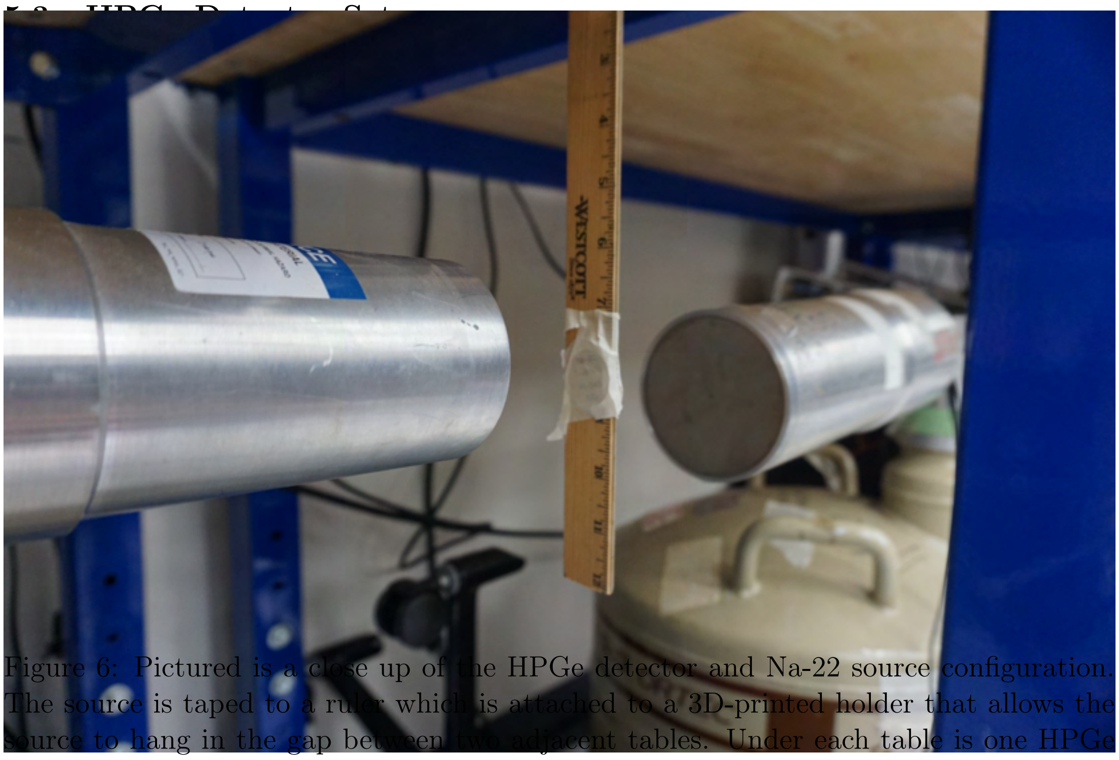


Figure 6: Pictured is a close up of the HPGe detector and Na-22 source configuration. The source is taped to a ruler which is attached to a 3D-printed holder that allows the source to hang in the gap between two adjacent tables. Under each table is one HPGe detector, 8.75 cm away from the source on either side. Note that the thickness of the ruler and source container is about 0.5 cm and produces negligible influence on this experiment.

The next test system I set up involved two HPGe detectors positioned 180°apart with approximately 18 cm between their beryllium windows. Each HPGe detector was attached to a liquid nitrogen dewar that was refilled every other day. The liquid nitrogen dewar was extremely important, as the germanium crystals are highly sensitive to thermal energy; the cold finger keeps the crystals close to the temperature of the liquid nitrogen (77 K) to prevent thermalized electrons from ionizing and creating electron-hole pairs that produce noise and blur experimental data.

$$\text{Na-22} \rightarrow \text{Ne-22} + \text{e}^{+} + \nu_e \tag{1}$$

Sodium-22 decays into Neon-22 via $\beta+$ emission, producing a positron (e^{+}) and electron neutrino (ν_e) 1. When the positron encounters an electron, they annihilate and produce two 511 keV gamma rays in opposite directions as their masses are converted into energy. When one gamma hits one detector, we can expect a coincident gamma to hit the other detector because they are produced at the same time in roughly the same location. Note that the acrylic case that holds the Na-22 source prevents the positrons from moving beyond the local source region, which inhibits off-axis annihilation. Without this attenuation, annihilation could occur closer to one detector than the other, which would cause the 511 keV gammas to arrive at the detectors at different times, blurring the resolution of the coincidence measurements.

Each detector has 5 input/output connections (4 BNC, 1 SHV) that serve various purposes. The first is the high-voltage input which receives power from a NIM module; the left detector requires -2000V and the right detector requires -3500V. The detectors also include a preamplifier cable that receives power from an amplifier NIM module. A third input is the *Test* input, which takes an external signal, from something like a waveform generator, in order to troubleshoot the preamp before an experiment is done. The remaining two connections on each detector are the signal outputs, which can be fed into NIM modules and/or the DAQ system directly. The only connections necessary for an experiment are the high-voltage input, the preamp input, and one signal output, though I used the *Test* input to perform diagnostic tests.

5.3.2 Testing with a Clover Detector

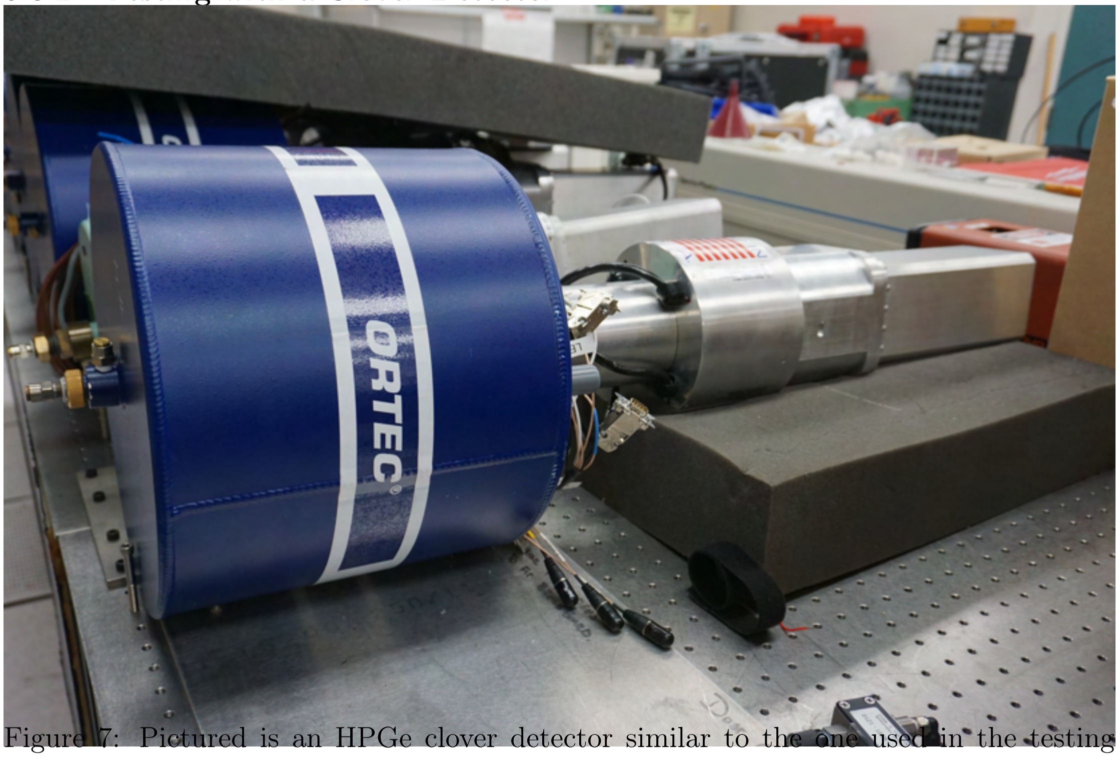


Figure 7: Pictured is an HPGe clover detector similar to the one used in the testing experiment with attached dewar. The four crystals are located in the front-most metal chamber of the detector arm.

Testing with the silicon detectors was not sufficient to make conclusions about the accuracy and reliability of the 1D histogram features, so additional tests were conducted using an HPGe clover detector 7. The clover detector includes 4 HPGe crystals and a compact dewar for liquid nitrogen. Each crystal in the detector is biased to a different voltage, specified by the manufacturer, depending on their level of purity. Each crystal was tested with the DAQ system, though only the results from one test are provided in the **Conclusions** section.

The clover detector, called Clover 1, was continuously cooled for over 48 hours before the tests were performed. The detector was provided with 4500V, 3500V, or 2000V depending on the leaf, and the preamps and output signals were routed to the amplifier module and MDPP-16 SCP module respectively. A sample of Co-60 with was placed on a plastic tube ∼22 cm from the clover detector and left to radiate for 40 minutes while data was collected with *mvme*, calibrated with the proper signal and energy settings.

6 Conclusions

The data acquisition system was tested for its output reliability, readability, and flexibility with a silicon detector and Cf-252 source, two HPGe detectors and a Na-22 source as well as an HPGe clover detector and Co-60 source. The integration of a liquid scintillator and PMT was scrapped after the PMT was tested and found to be broken. The Cf-252 and Co-60 setups were used primary to validate the 1D histogram capabilities and the Na-22 setup for the 2D histograms and associated features.

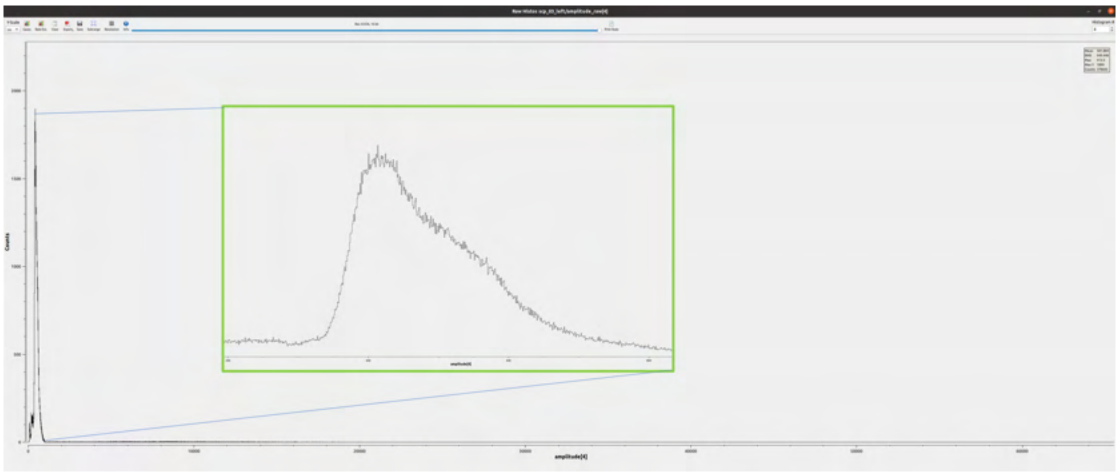


Figure 8: Pictured spectrum populated by detecting Cf-252 alpha decay for 16 hours. The axes were not calibrated with a second source and do not correctly bin energy. The energy distributios mirrors that which was found in [1] though is more irregularly shaped.

The alpha energy distribution of the Cf-252 source should be Gaussian according to Gulkanyan [1], though the the shape of this peak is very non-Gaussian. Initially this caused some concern about the reliability of the detector system because I wasn't really sure what I was seeing. I came to the conclusion that the irregularity of the peak was likely due to poor experimental design. The vacuum chamber used in the experiment had been left un-pumped for some time when this data was collected and because the pump was taken to be used elsewhere in the lab, I could not re-pump the chamber. Furthermore, the original high voltage supply I used to bias the detector was being used for another experiment, so I had to use an older, less-accurate voltage supply. Finally, the silicon detector is sensitive to gamma rays and lacked an adequate level of shielding from room-background, which could've caused an irregular distribution.

Regardless of the reason, it was important to test the DAQ system for reliability as it will soon be used for more precise experiments. I turned instead to a HPGe clover detector, which was being tested after returning from an off-site experiment. With the help of my lab group, the clover was vacuum pumped and annealed to heal radiation damage; it was then cooled with liquid nitrogen over several days before being tested. A Cobalt-60 source was placed 22 cm from the detector face and each detector leaf was biased. The leaves were tested and calibrated independently, and only one Co-60 spectrum is shown.

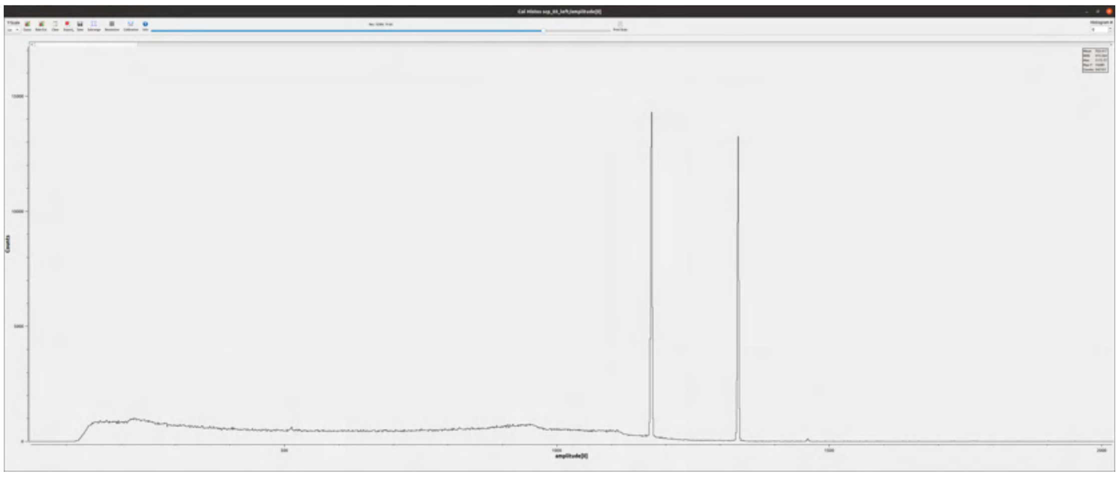
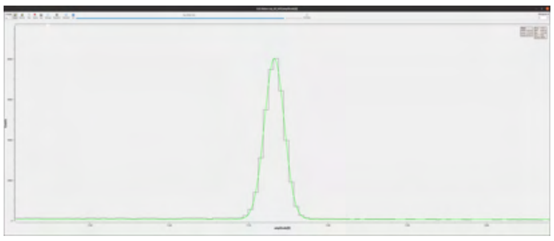


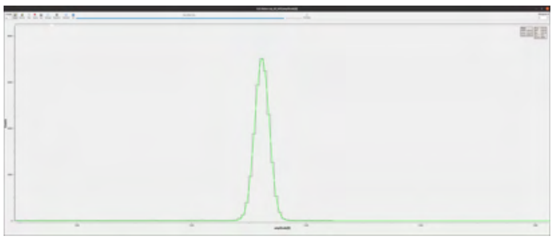
Figure 9: This figure shows the Cobalt-60 spectrum captured by *mvme* using an MDPP-16 SCP module. The 1173 and 1332 keV peaks can be clearly seen, with a more detailed peak analysis in the follwoing figures.

The Co-60 spectrum generated was detecting using Clover 1, Leaf 1, biased to +2000V with a high voltage supply; the DAQ system collected this spectrum in 40 minutes. The raw signal was routed directly to an MDPP-16 SCP module for delivery to *mvme*. The frontend settings were adjusted to account for the shape of the detector signal, which was first analysed on an oscilloscope. The resulting spectra was binned arbitrarily and thus needed to be calibrated; this was relatively simple as Co-60 is a well studied isotope with extremely distinct 1173 and 1332 keV peaks. After calibration, the peaks lined up perfectly with their known/assumed energies; the standard deviation σ was calculated next by divding each FWHM by $2\sqrt{2 \cdot ln(2)}$ (~ 2.35). The standard deviation was 1.25 (σ_{1173}) for the 1173 keV peak and 1.27 (σ_{1332}) for the 1332 keV peak; both of these values show that the standard deviation from the central energy value is ~ 1 keV, which denotes a well defined and resolved energy peak.

The Sodium-22 experiment was conducted to test the 2D histogram capabilities of *mvme*; the coincident 511 keV photons from positron-electron annihilation offer the per-



(a) This is the 1173 keV peak taken from the above Co-60 spectrum. Peak statistics are shown in the top right corner of the window, including a FWHM of 2.93 keV, and are generated via a Gaussian fit.

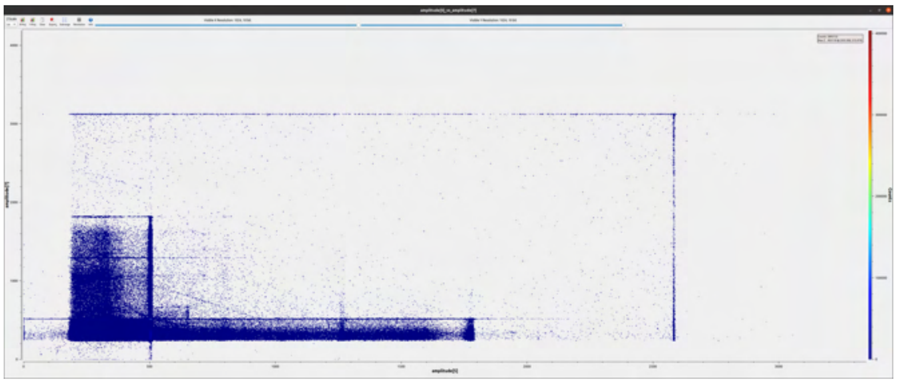


(b) This is the 1332 keV peak taken from the above Co-60 spectrum. Peak statistics are shown in the top right corner of the window, including a FWHM of 2.98 keV, and are generated via a Gaussian fit.

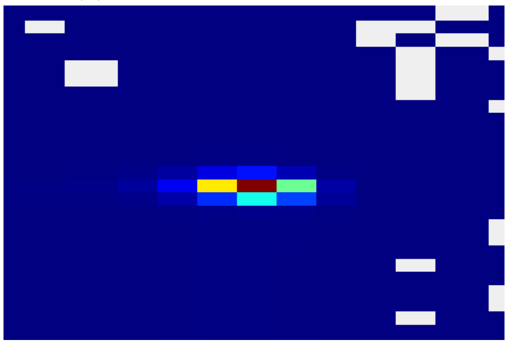
Figure 10: 1173 and 1332 keV peaks from the Cobalt-60 spectrum.

fect data to analyse with a 2D histogram. After setting up the experiment, whose details are mentioned in the previous section, I collected data for 63 hours. The following images show the data collected from the run, including a full-spectrum coincidence 2D histogram, and a zoomed-in image of the 511 keV coincidences. As expected, the 511 keV coincidences are displayed as an intersection on the 2D histogram, with the majority of the counts at the center of their overlap. There are also some trace coincidences found between the 511 keV energies and other energies - this is to say that the 511 keV photons were produced and recorded in abundance as expected. Because no additional source was used for calibration, the axes energies are not properly calibrated, though the known shape of the spectrum makes the 511 keV coincidences obvious.

The histogram was also displayed in the 2D histogram multi-window view and confirmed to display in the same way. The window that displayed the data was also fully independent of its neighbors and operated in the same way as a singular widget with all of the tools fully operational.



(a) Full spectrum coincidences.



(b) 511 keV coincidences closeup.

Figure 11: This figure shows the Na-22 coincidence measurement. The frequency of the production of coincident 511 keV photons amounts to the high concentration of counts at the coincident location on the histogram; the red-colored blocking makes this easier to see.

7 [redacted] Configuration

The [redacted] transport system was developed by my lab for quickly transporting irradiated materials from a radioactive source to a detector – this system is being used alongside the *mvme* data acquisition system for nuclear fission product yield experiments and cross-section measurements. The [redacted] is small plastic capsule that holds a certain material to be exposed to radiation whose products will be studied by HPGe clover detectors. The [redacted] begins its journey in front of the radiation source and a series of blowers, valves, and a tubing network allow the [redacted] to move to a detection location in less than a second. In order to interface this system to the DAQ, I was tasked with connecting *mvme* to the Arduino PLC that controls the [redacted].

The PLC contains a code that controls the valve/blower network which moves the [redacted] through the PVC tubing. Based on input from my lab group and the capabilities of *mvme*, I determined that the most efficient way to interface the [redacted] and DAQ software was to start the PLC with a signal generated by *mvme* at the same time the DAQ begins. I did this via the MVLC Trigger Logic I/O; the MVLC controls each of the DAQ modules, and has 13 frontend IO locations. The MVLC's trigger interface allows the user to choose the signal direction for each location (input or output) – the output signal is a NIM pulse with a width of 8 - 63356 ns.

The Arduino was initially set up to receive a DC signal of 10V in order to begin the program. However, the MVLC is only capable of generating a NIM pulse, which is a -1V signal. Instead, the PLC settings were arranged to receive an analog signal of +2 - 24V. This -1V NIM pulse was then fed through a NIM-to-TTL converter module, which output a +3V signal with the same signal width. Because the Arduino reads each input every 10000 ns, I set the output pulse width of the NIM signal to 48000 ns to ensure the program began - this signal is adjustable, but was consistently successful at pinging the PLC at this width.

The output signal is generated internally when the DAQ system starts and is routed through a 2 processing levels to the frontend output. After the *START* button is clicked, the signal is generated and instantaneously sent to the PLC to run the [redacted] control program at the same time data acquisition begins.

8 Troubleshooting and Known Issues

Though *mvme* has shown to be a reliable software with extensive functionality, there are a few issues I have encountered throughout my time developing the program. The first thing to note is that *mvme* is very sensitive to signal shape parameters. When a default module is added to the software, a *Frontend Settings* script is populated under the module settings (**Events ▷ [event_name] ▷ Modules ▷ [module_name] ▷ Frontend Settings**). Depending on the module type, there will be multiple shaping settings including *Rise Time*, *Decay Time*, *Gain*, etc. It is important to know the specifics of the signal you're sending to *mvme*, as well as how changes in the gain will impact your signal - it is *highly* recommended to study the signal with an oscilloscope before sending it into *mvme*. An improperly calibrated module can malfunction when it recieves an unrecognizable signal. There are two obvious symptoms of this kind of malfunction. First, the user will notice that *mvme* will not collect data from the module with the mismatched signal; secondly, all of the channel lights on the improperly calibrated module will light up red.

The way to fix this sort of error is simple, though not always trivial. What I mean is that, finding the culprit of the error is sometimes difficult as one or more of the signal settings may need to be changed. I've found that the gain is oftentimes part of the issue. Because the gain is essentially amplifying the signal voltage, the module jumpers may have difficulty with interpreting the enlarged signal if it is out of range; this is especially problematic when cosmic radiation hits a detector, as cosmic rays already have a large voltage peak-to-peak. For example, if you're routing a 1Vpp signal into a detector with 3V jumpers, the maximum gain you can have is 3. However, if background radiation exceeds 1Vpp, the module would likely malfunction because of the overvoltage caused by the gain.

If this issue comes up, the following steps are recommended: Begin by ensuring that *mvme* is not collecting data and power off the VME crate. Unscrew the module from the crate, and pull it out halfway so that its pins disconnect, but it doesn't fall out of its slot. Return to *mvme* and fix the setting(s) causing the problem; if the problematic settings cannot be determined, reset the settings to their default values and re-enter custom values one at a time. Push the module fully into its slot and screw it back in. Power on the crate and reconnect the MVLC in *mvme*; click the DAQ *START* button and notice that the red lights have gone off and the module can receive signals again. At any point during this procedure, feel free to restart *mvme* itself, or to open a fresh settings file; though this is not necessary, it may help the user to keep better track of changes made to the settings.

As far as bugs are concerned, there is only one bug that could be a potential issue for *mvme* users and it involves the 2D histogram feature. Now that the user can create arrays of 2D histograms, there is the potential for arrays of different size to be selected in the same array instance. Because of how *mvme* treats the array inputs, by lining up like channels (see *Histograms* section for more details), arrays of different sizes should not be selected together. Initially, when I was exploring this possibility, the mismatched arrays caused a crash when their 2D histogram tree was created. However, I added a check point that stops the 2D histogram array from being created when the input array sizes are incompatible, circumventing a crash. Although, when a new 2D histogram (array) is created and the *OK* button is clicked, an instance of the 2D histogram is added to the histogram sink, even if the software prevents the array from being added to this instance. And clicking on this instance will cause a crash.

I am still looking at this bug and trying to determine the best way to fix it, but be aware that this bug can cause the software to crash. The best way to avoid this is to be careful not to create a 2D histogram array out of inputs with different sizes. In the event that one of these instances is created from mismatched arrays, the instance can be deleted like any other histogram in order to prevent a crash.

References

[1] Hrant Gulkanyan and Amur Margaryan. Alpha-spectroscopy of cf-252 decays: A new approach to searching for the octoneutron, 2014.