

PLSpectrometer GUI Documentation (Supplementary)

Alistair Bevan

September 7, 2023

Contents

1	Introduction	2
2	GitHub	3
2.1	Installation	3
2.2	Updating the Code	3
2.3	Visual Studio Code	3
3	Modifications and Procedures	4
3.1	Summary of Changes	5
3.2	Basic Overview of the GUI	5
3.3	Recalibrate	5
3.4	Optimize	6
3.5	Move	7
3.6	Scan	7
3.7	Temperature Sensor	8
3.8	Data and File Saving	8
3.9	Search Tab	9
3.10	QtDesigner File	10
4	Future	11
4.1	PLE Automation	11
4.2	Concerns with Optimize	11
4.3	Input Restrictions	11
5	Extra Notes	11
5.1	Abort	11
5.2	Temperature Sensor Design and Theory	12

1 Introduction

This supplementary document was made to accompany the GUI used to operate a photoluminescence (PL) spectrometer and Elliot Wadge's documentation for it. The GUI, coded in Python by Elliot, was itself based on a preexisting GUI made in LABVIEW by Matt Frick. The current program consists of many separate files. The GUI consists of a set of controls, as well as a series of tabs for plotting intensity (in counts/s) versus wavelength, plotting intensity versus energy, and searching and plotting previous data. The search tab utilizes a modified version of a program written by Colton Lohn.

While Elliot's documentation serves as a introduction to programming GUI in Python using the PyQt5 library and details the code, this additional document is meant to outline the modifications I made to the GUI starting in May 2023. The most notable changes were modifying the calibration controls, adding a temperature sensor, updating the file directories, and saving the data to the OneDrive. The search tab and optimize code were also modified.

I will try to walk through the changes and reasoning behind them, but I'll leave most of the details and specifics about the code to Elliot's document. His document also has a section containing very useful resources, which I would highly recommend to anyone who is going to be working with the GUI code. (They were certainly helpful for me. My only coding experience before this project was the simple python fitting and plotting I learned in the second and third year physics labs.) I apologise in advance for any errors in this document or issues with the modifications I made in the code. If you have any questions about the code or this document, please feel free to contact me at awb5@sfu.ca.

2 GitHub

Elliot's GUI code is available on GitHub at <https://github.com/Elliot-Wadge/Python-GUI>, though the code I worked on was forked by David Lister and can be found at <https://github.com/DavidLister/PLSpectrometer.git>. This section outlines the procedure for cloning the git repository to the new lab computer account, and syncing the local repository (on the lab computer) with the remote repository (on GitHub). I am by no means a pro at using GitHub, so these are just one set of instructions to get to the desired results; they may not be the most efficient or best methods.

2.1 Installation

To install the git repository on your device, follow these instructions.

1. If git is not installed, get it from <https://git-scm.com/downloads>. (To check if it's installed, you can type `git` in a Command Prompt window.)
2. Go to the location on your computer you would like the repository to be in, right-click and select "Git Bash Here". It may look slightly different and you may need to click "Show more options" to see it.
3. Type `git clone -b branch-name https://github.com/DavidLister/PLSpectrometer.git` to clone the repository. Replace "branch-name" with the branch you would like to be on. For instance, Temperature is the most up-to-date branch.
4. To run the program, ensure PyQt5, PyQtChart, numpy, plotly, and nidaqmx are also installed. (For example, you can type `pip install numpy` in the Command Prompt and it should install numpy.)

2.2 Updating the Code

The following instructions can be used to bring the code up to date with what is on GitHub. Note that it's best to avoid making code changes in your local repository without being signed into a GitHub account that is able to modify the code. This should ensure smooth pushing and pulling of changes.

1. Open the local repository folder (named "PLSpectrometer"), right-click, and select "Git Bash Here."
2. Check that you are on the correct branch by typing `git branch`. To switch branches, type `git checkout branch-name`. Replace "branch-name" with the branch you would like to be on.
3. To update your local repository with the latest changes, type `git pull`.

2.3 Visual Studio Code

For working on the code, I used Visual Studio Code (VSCode) which is a popular and versatile code editor developed by Microsoft and can be downloaded from <https://code.visualstudio.com/>. The only extensions needed to work with the GUI are "Python" and "GitHub Pull Requests and Issues", which are relatively easy to find in the Extensions tab. To run the GUI, open the GUI.py

file and then hit "Run Python File" in VSCode. (There should be a button for this, with an arrow icon, at the top right hand corner of the window.)

Of course, to work on the code, you can use whichever code editor you like. I mostly included this section because I wasn't sure which code editor to use when I started since I had only coded in Jupyter notebooks beforehand; I ended up finding VSCode pretty straightforward to use.

However, some issues seem to arise when the GUI is not run through VSCode. For instance, when I tried opening the GUI with Python, the position of the single and double spectrometer reset to their default values (100.0 nm and 366.0 nm) and the audio for the optimize function did not work.

3 Modifications and Procedures

As well as outlining the changes made to the GUI, it would be beneficial to also walk through some of the program procedures, as some of these have changed with the updates. I'll focus more on the usage of the program here, since the programming aspect is covered well in Elliot's documentation.

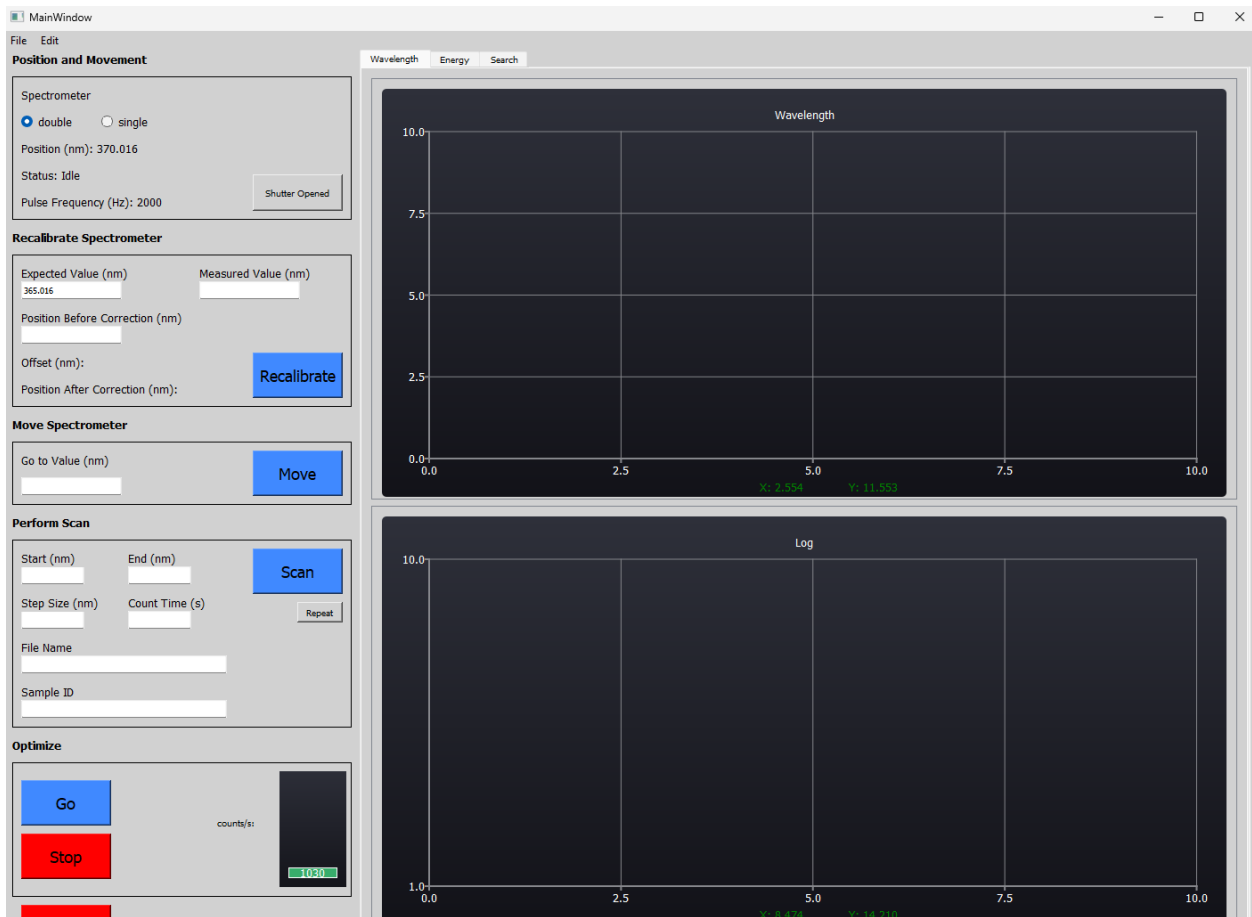


Figure 1: Layout of the GUI.

3.1 Summary of Changes

To quickly summarize all the updates made, the largest change to the procedure would arguably be to the recalibration controls. The edits to the optimize function were mainly correcting the directory and adding a moving average. Move and scan remain essentially the same as they were before, with the exception of the inputs being rounded to 0.001 nm. Additionally, a temperature sensor feature was added, which measures and records the temperature of the apparatus at each data point. Files are now saved to the OneDrive but it is not difficult to change this in the code so that the files are instead saved on the local device. The search tab was updated so that the file names are hyperlinked for plotting and the function searches through data files in the OneDrive. The QtDesigner file used for the layout of the GUI was also updated, with `spectrometer_new.ui` and `qt_designer_new.py` being the most up to date versions.

3.2 Basic Overview of the GUI

Upon opening the GUI you should see the controls on the left hand side. To the right of these are the plots used to visualize the data as it is collected in real time. There are three tabs; the first and second can be used to switch between wavelength and energy on the x-axis, while the third one can be used to search for and plot data.

To switch to the spectrometer you would like to control (double or single), simply click on the double or single radio button at the top left hand side of the window. The double is selected by default. The "Position" displayed in the top left will reflect the current position of the selected spectrometer and will be updated to reflect the spectrometer selected. This will also update when the spectrometer is done a move, and not during a move; since a scan basically consists of a series of moves, it will essentially update in real time during a scan, given the moves are relatively small.

The spectrometer will always approach a scan or move from the same direction to avoid error introduced by backlash. (Backlash is caused by the spacing between the gear teeth.) Depending on the initial position, the spectrometer may first move to a position 20 nm smaller than the value we would like to move to or start the scan from and then begin the approach. When it is within 0.003 nm of the desired position it will slow down.

For most functions, buttons will be disabled until the function is stopped or ends. The "Abort" function serves to stop a scan or optimize, but won't do anything to stop a move.

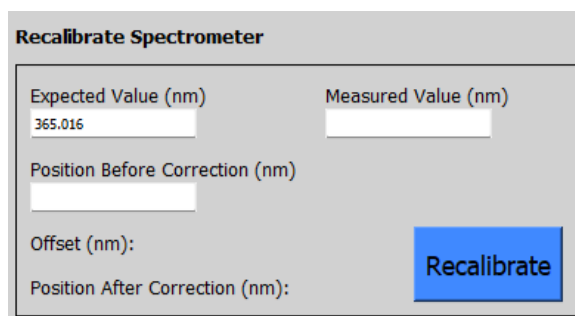
There is also a button for opening and closing the shutter on the double spectrometer. This is useful if the slit widths are large and the room lights need to be turned on.

As it is right now, I think the GUI is best opened using VSCode on the lab computer, for reasons discussed in subsection 2.3 and subsection 3.4.

3.3 Recalibrate

Recalibrating the spectrometer is generally done by observing some feature of the data, such as a peak, and knowing what we expect its value to be. The controls work the same for the double and single spectrometer; both can be calibrated individually, without affecting the other. Note that if the spectrometer selected is changed (e.g. double to single), the "Recalibrate Spectrometer" box will remain unchanged (i.e. the current values will remain displayed).

To recalibrate, enter the wavelength value you observe from the data in the "Measured Value" box, while the wavelength value you expect (e.g. a literature value, such as 365.016 nm for Hg) can be entered in the "Expected Value" box. By default, a value of 365.016 nm is already entered in the box. The current position of the spectrometer also needs to be entered in the "Position Before Correction" box. Press "Recalibrate" and the GUI will calculate the offset and update the stored



The image shows a GUI window titled "Recalibrate Spectrometer". It contains several input fields and a button. The "Expected Value (nm)" field has the value "365.016". The "Measured Value (nm)" field is empty. The "Position Before Correction (nm)" field is empty. The "Offset (nm):" field is empty. The "Position After Correction (nm):" field is empty. A blue button labeled "Recalibrate" is located on the right side of the window.

Figure 2: Recalibrate section of the GUI.

position of the spectrometer accordingly. The position displayed in the "Position and Movement" box will also be updated.

Originally, the recalibration was done by hand and the position of the spectrometer was specified directly. This was a little tedious and risked confusion with adding or subtracting the offset. However, there are still cases where it may be necessary to specify a certain position (e.g. you would like to reset the position to that displayed by the spectrometer itself). To do this, simply input the same value in the "Expected Value" and "Position Before Correction" boxes and the value you would like to set the spectrometer to in the "Position Before Correction". This will result in an offset of zero, and the "Position After Correction" will be the same as the value inputted in the "Position Before Correction" box.

3.4 Optimize

The optimize function serves to help with the alignment of samples, mirrors and lenses and would be used before performing a scan. Only the double spectrometer supports optimize, as the single does not have a photomultiplier tube (PMT). Thus, whichever spectrometer is selected will have no effect on the optimize function; it will always use the PMT on the double spectrometer.

Essentially, the spectrometer is moved to a specific wavelength and the counts at this wavelength are measured in 0.15 s intervals. The count rate is displayed in a bar chart at the bottom left of the GUI; there is also an audio component consisting of beeps at different frequencies, which is helpful when you cannot see the computer screen while adjusting the setup. Note that the intensity in the bar chart is in counts/s, so the values read here should agree with the data that ends up being collected at the particular wavelength. The value displayed is actually a moving average, which was done to reduce the impact of noise.

To begin optimizing, hit the "Start" button; pressing either the "Stop" or "Abort" button will end optimize. The graph will update in real time, while the frequency of the beeps rises with the number of counts that the PMT is detecting (400 - 800 Hz). Because the range of counts can be so large, the frequency scale resets for every factor of 10 increase or decrease in counts/s. This results in a drastic drop in frequency if it rescaled by moving up a factor of 10 or a drastic increase in frequency if it is rescaled down by a factor of 10. In the range where this drop doesn't happen, a rising frequency (higher pitch) signals in increase in intensity.

There is one issue that remains unresolved with the optimize function, though currently it is not impacting the functionality. Occasionally, an error message is displayed in the terminal ("QObject::startTimer: Timers cannot be started from another thread") and a beep is missed. This rarely occurs in VSCode but when the GUI is opened directly with Python none of the beeps seem to be audible. Just for reference, VSCode on the lab computer is currently using the 3.10.4

version of Python and the 5.15.9 version of PyQt5.

3.5 Move

The move function was hardly changed but I'll still mention it here. It works by specifying a value in the "Go To Value" box and hitting the "Move" button; the spectrometer will then be sent to this position by sending pulses to the stepper motor. This can be done for both the double and single spectrometer. The only change was that the value entered will be automatically rounded to three decimal places (i.e. 0.001 nm) as not doing this can result in an offset error. This arises from the fact that the smallest increment of movement is actually 0.00025 nm, which is equivalent to one pulse, so trying to move to a position which is not a multiple of 0.00025 will introduce rounding errors. This was a much bigger deal for a scan which essentially consists of a series of move functions. (The addition of small offsets for each move in a scan would result in a large error.)

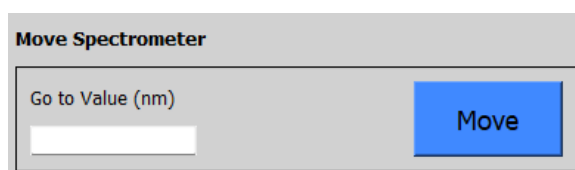


Figure 3: Controls for moving the spectrometer to a specified position.

Rounding to 0.001 nm seemed to be the most reasonable solution and some instances of rounding to 0.001 nm already existed in the code. Most scans are done at 0.005 nm step sizes anyway, as achieving greater precision than this is not very realistic with the current setup.

3.6 Scan

Besides rounding the inputs to 0.001 nm and adding a temperature component, the scan function should be mostly the same as before.

Before running a scan, the spectrometer should be properly calibrated and optimized. (Or perhaps not if the scan is for the calibration.) Enter the start and end wavelength, step size, count time and sample ID into the appropriate boxes and hit the "Scan" button to begin the scan. The sample ID is optional but useful for including details about the run (e.g. temperature of sample, slit width of the double and single spectrometer, voltage and current of the LED); it is also helpful when using the search tab.

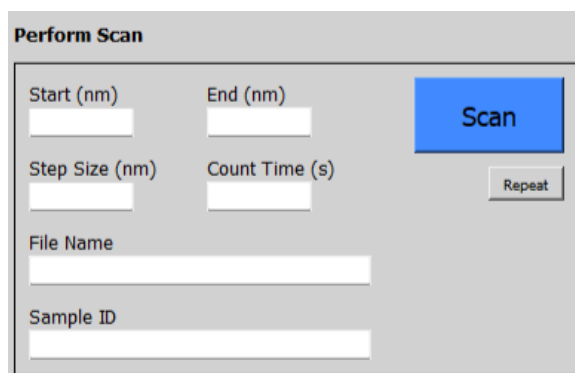


Figure 4: Controls for scanning.

The scan is essentially just a series moves each followed by data collection. It will move the step size, collect data for the count time, and then repeat this until it reaches the end point. The position (nm), intensity (counts/s) and temperature (degrees Celsius) are all recorded in a data file.

The repeat button can be pressed and will repeat the scan until it is deselected. Note that the repeat button is basically just repressing the scan button. As a result, changes to the inputs will effect the next scan.

3.7 Temperature Sensor

A temperature sensor consisting of a thermistor, resistor and NI USB-6009 DAQ device was added to monitor the temperature of the double spectrometer during its runs. (There is another system for monitoring the temperatures within the cryostat, which is separate from this GUI.) Mainly this was so we could see if there was any correlation between shifting of the calibration spectrum and temperature and so the temperature during overnight scans could be recorded. (At the time this document is being written, we are still looking into this.)

The scan function was modified slightly so that the temperature at each wavelength would be recorded in the data file. There is a separate file, `temperature.py`, which defines a `TemperatureSensor` class where most of the measurement and initialization details are coded, while the formulas for converting from the DAQ voltage reading to temperature are defined in `miscellaneous.py`. The voltage used for each temperature is actually the average of 10 measurements at 0.01 s intervals.

If the sensor is not detected, the program should still work, with the temperature being recorded as "N/A". However, there will be no warning displayed in the GUI if this is the case. (One would have to be monitoring the readout in the terminal.)

In the `temperature_sensor` folder which can be found in the testing folder, I included the files I used when I was making the temperature sensor. Notably, this consists of a separate GUI that displays the temperature, maximum temperature range and resistor voltage reading. The collection interval can be set to values between 1 and 10 seconds. This GUI may come in handy if troubleshooting the temperature sensor.

3.8 Data and File Saving

One major modification occurred due to the account on the lab computer being changed. This messed up the directory for saving files, since it was specified explicitly in the code.

Originally, I was able to modify this so that it would save to files contained in folders corresponding to the date the data was collected (e.g. "2023 08 30" for August 30, 2023) in `Documents\PL\Data` under the user. The code was generalized using `os.path.expanduser('~')` so it should work for any user. Additionally, if the folders do not exist already, they will be created.

However, when the lab began using the OneDrive, it seemed to make more sense to save the data to the OneDrive. Currently the data is being saved under `C:\Users\"user-name"\Simon Fraser University (1sfu)\Simon Watkins - MOCVD-LAB\data\ZnO\PL\Data\"YYYY MM DD"`. Note that even though samples may not necessarily be ZnO (e.g. some GaN samples have been analyzed), they are still in the ZnO folder. (I'm guessing this is because originally only ZnO samples were measured using the spectrometer and it was too much of a hassle to change it later on. For now, I've also decided it would be too much hassle to figure out.)

Saving to the OneDrive comes with the advantage of easy access to the data and secure storage, but at the cost of risking a scan being interrupted if the connection between the computer and OneDrive is lost. This is more of a concern for overnight scans, though as of now there have not

been any issues. (We have actually run the spectrometer constantly for over five days, doing repeat calibration scans to investigate the temperature changes, and there have been no issues with the data saving to the OneDrive.) The code for saving locally is commented out in the line just above the code for saving to the OneDrive, so switching the saving location would be straightforward. It is line 36 of scan.py.

3.9 Search Tab

The search tab can be used to locate and plot previous data sets. You just need to enter in details of the sample ID (e.g. the sample name) and a year to search for the data in. The directory used here also required updating, and is now set to search for files in the OneDrive.

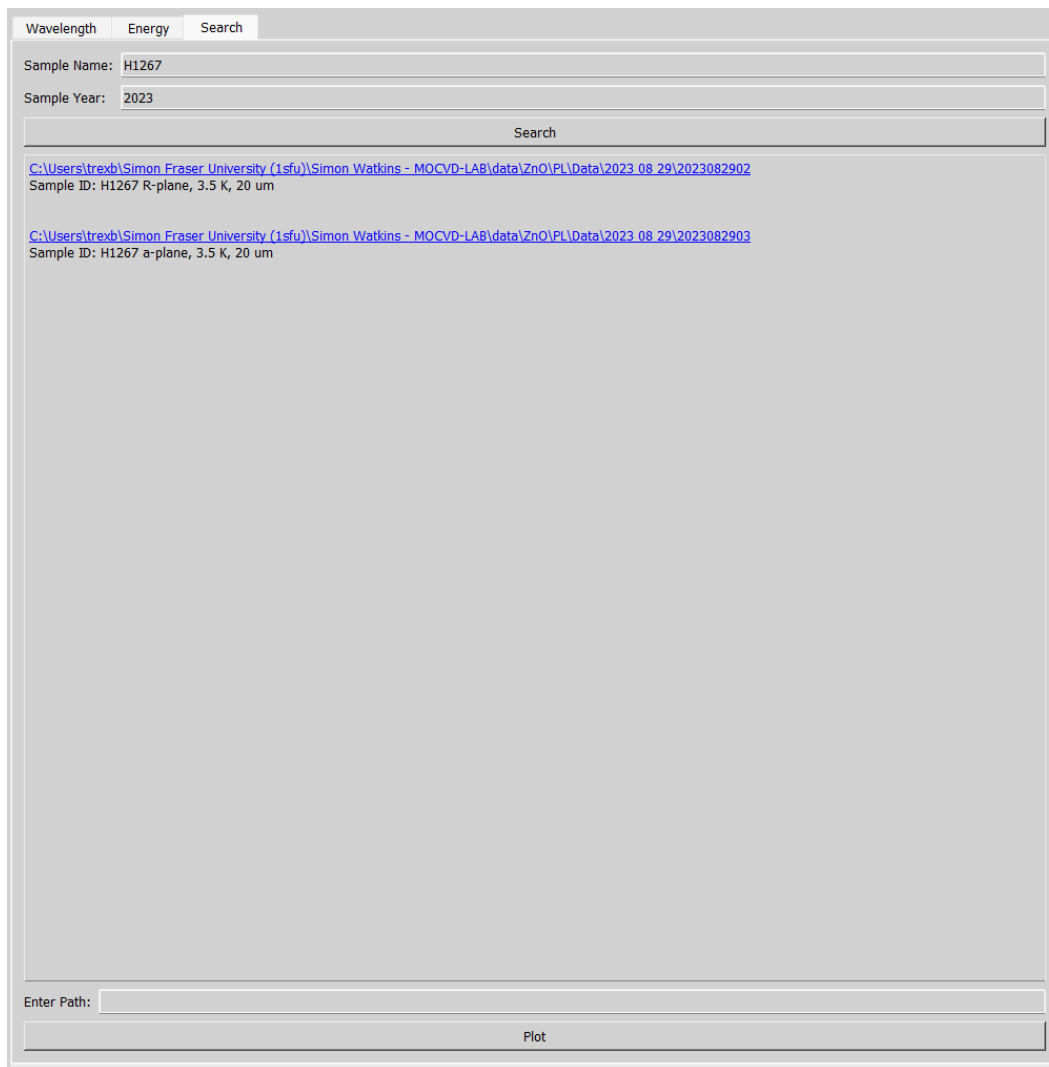


Figure 5: Search tab in the GUI.

Unfortunately, since the SearchUI program this is based on has to open files to read them, searching the OneDrive requires downloading all the files being searched through onto the computer. This can be problematic, as there are many PL data files in the OneDrive. All the files currently amount to just over 500 MB right now so at least storage should not be a major concern.

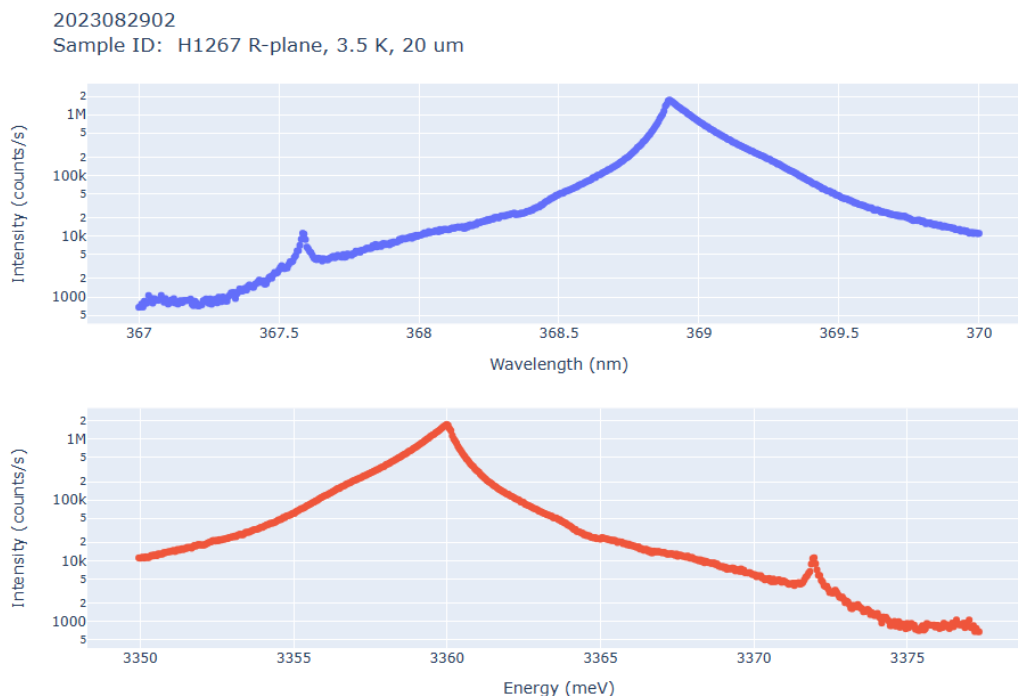


Figure 6: Example of results plotted using the search tab.

The time taken to download the files is the main issue.

This problem can be somewhat lessened by specifying a year to search for the data in, as only the files under this year would need to be downloaded. (I actually coded it so it shows a warning message if a year is not entered.) We can even narrow our search to the specific month or day of data collection (e.g. entering "2023 09 30" for the year will search only in the "2023 09 30" folder, while entering "2023 09" will search through folders beginning in "2023 09"). Another solution is just searching for something without specifying a year and downloading all the files to the local device, which would make future searches more efficient.

I also added the feature of hyperlinking the results of the search, so that you can simply click the file name and the data will be plotted in a browser such as Edge or Chrome. Previously, one would have to copy the name into the "Enter Path" box at the bottom of the screen and hit "Plot" which was a little more tedious. The hyperlink method was actually done in a separate SearchUI in the OneDrive, which I presume to be the one by Colton Lohn. I'm not sure why it wasn't included in the spectrometer version of the SearchUI but this update seems to work fine. I decided to leave the "Enter Path" box and "Plot" button in as well.

3.10 QtDesigner File

Updating the recalibration procedure required modifying the GUI layout and adding a few new input boxes and labels. Rather than modifying `qt_designer.py` (a Python file), it is much better to make changes to `qt_designer.ui` (the QtDesigner file). Then, opening a Command Prompt in the location of the QtDesigner file, you can convert the updated QtDesigner file into a python file by typing `pyuic5 input.ui -o output.py`. (You can use the `cd` command to navigate to the directory where your `.ui` file is located. For example, you can type `cd path\to\your\ui\file\directory`.)

Here, "input" is the name of the QtDesigner file and "output" is the name you would like to use for the Python file.

The updated files with the recalibrate calculator are `qtdesigner_new.py` and `spectrometer_new.ui`. Before this, the `qtdesigner.py` and `spectrometer.ui` files were used. The `spectrometer - Copy.ui` file appears to be an even older version which is missing the "Repeat" and "Stop" buttons. I think it was probably a backup that was created before those features were added in.

4 Future

4.1 PLE Automation

Probably the main and most ambitious plan for the spectrometer is to enable automation of the double and single spectrometer. Essentially the single would move to different wavelengths and the double would do a scan at each wavelength. This would be very useful for doing long, overnight scans.

4.2 Concerns with Optimize

As mentioned in subsection 3.4, there may need to be further changes made to the optimize function, specifically the audio component. Opening though VSCode currently seems to allow for the error message and audio cut outs to be very minimal but if this worsens it could significantly impact the functionality of optimize.

4.3 Input Restrictions

While there are restrictions on some of the inputs in the GUI, a lot of it is up to the user to do correctly. This may be problematic if the user is new and unfamiliar with the GUI or if the user makes a typo in an entry. At the very least, the program checks that floats are entered when they are required but there are minimal warnings or restrictions on these values. Something that does exist is a warning message when asked to do a move over 100 nm. I mentioned in subsection 3.9 that I added a warning when not specifying the year during a search. I don't think it's essential that more restrictions are added, but it may be a rabbit hole to go down one day.

5 Extra Notes

The following section contains some miscellaneous notes I thought it was worth adding in case they were ever useful. Some relate to questions I had or think I might have if starting to work with the spectrometer. Everything here is extra, so as not to clutter the other sections.

5.1 Abort

While using the spectrometer, I found myself (and others) cautious about using the abort function, mostly because I hadn't yet looked into the specifics about how it worked. Basically, pressing "Abort" will do nothing during a move but it will stop a scan and optimize.

If "Abort" is pressed during a move, the spectrometer will just keep moving until it reaches the position it was told to move to. During a scan, which is made up of many small moves, it will end the scan once the move and measurement it is currently on is done. If the interval is small this will be almost instantaneous. The position of the spectrometer will be correctly updated.

Pressing "Abort" during an optimize will do the same thing as pressing "Stop"; it will end the optimize function.

If you hit "Abort" while "Repeat" is still selected (I learned this one from experience) it will stop the current scan and then start the next scan. And it will keep repeating. The only way to actually stop the scans from continuing is to deselect "Repeat", and then hit "Abort".

5.2 Temperature Sensor Design and Theory

I thought it might be nice to briefly cover some details to do with the design and theory of the temperature sensor. The sensor setup consists of a thermistor, resistor and NI USB-6009 DAQ device. A circuit diagram is shown in Fig. 7. The manufacturer product number of the thermistor was NTCLE100E3103HT1 and it was purchased at <https://www.digikey.ca/en/products/detail/vishay-beyschlag-dralloric-bc-components/NTCLE100E3103HT1/2230723>. I have included the thermistor datasheet with the temperature sensor testing files.

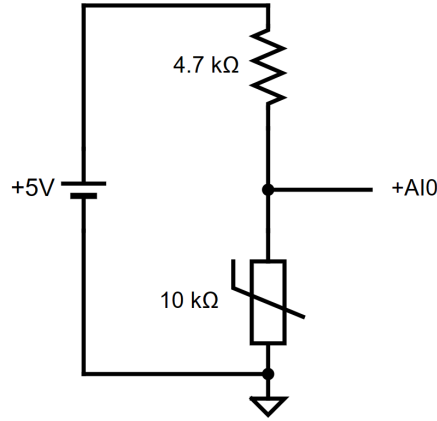


Figure 7: Circuit diagram of the temperature sensor.

As the temperature of the thermistor changes, its resistance will change as well. This can be modelled by the Steinhart-Hart equation which is commonly used to derive a precise temperature of a thermistor. The coefficients are usually published by thermistor manufacturers, and for the thermistor used here, an expanded form of the Steinhart-Hart equation was provided as well.

The process used to make measurements is shown in Fig. 8. Essentially, the voltage across a resistor of known resistance was measured, and using Ohm's law, the resistance of the thermistor could be deduced. This could then be converted to the temperature of the thermistor using the Steinhart-Hart equation.

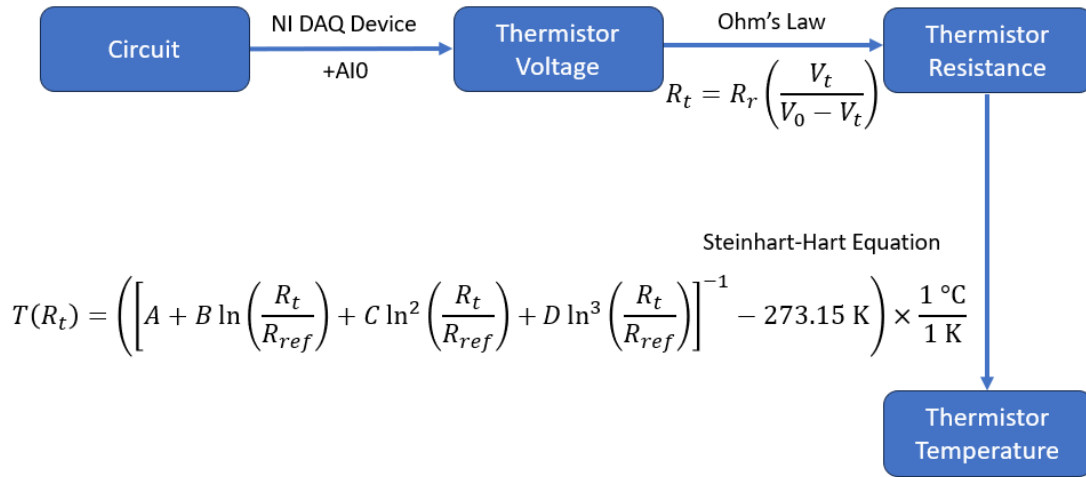


Figure 8: Summary of the process for temperature measurement. Note that the coefficients in the Steinhard-Hart equation are in K^{-1} , but I have modified it so that the temperature calculated ends up being in degrees Celsius.