Hall documentation

Elliot Wadge

June 2021

Introduction

This document is meant to give the reader an insight into how the Hall measurement system program works. The program consists of a handful of files making heavy use of the PyQt library to handle the visuals while the communication to the instruments is handled by the PyVisa library. The rest is handled by base python and NumPy for some of the fitting. The GitHub for the program is here https://github.com/Elliot-Wadge/Hall-GUI and it is meant to be viewed alongside this document. The code relies on the use of object oriented programming and the reader is encouraged to familiarize themselves with classes and inheritance before attempting to understand the program as a whole. Understanding pieces of the code without this knowledge is still doable.

Contents

1 Usage

The Hall tab is used for taking Van der Pauw measurements. It takes a specified number of voltage measurements at multiple current levels and repeats this 8 times across the different switch combinations. From this data the program fits the 8 different voltage vs current lines using np.polyfit() of the first degree and uses the resulting slopes to calculate the results displayed on the right side of the UI. The R-squared values of the fitted lines are displayed below the graph from left to right in ascending order corresponding to the switch number (i.e. the left most is switch 1 and the right most is switch 8). The repeat button just presses the go button again when the current measurement changes, this means any changes to the inputs will be executed in the repeat. The backup button uses pythons sub-process library to run the batch file corresponding to backing up the data to the network drive, unfortunately as of now this creates a dependency on the path of the batch file, so if it is moved you will have to change the specified path in the code. The IV tab takes voltage measurements across the specified switch and then fits that line and displays the slope as resistivity.

2 Code Overview

The UI is coded entirely in python using Python 3.9 and makes use of the PyQt library to make the visuals. The PyVisa library is used to communicate with the instruments and the program is broken down into different files that are named after the tasks they accomplish. Here is a list of the files and their function.

- GUI-main.py: main file, run to launch the program, builds and connects everything
- FileWriting.py: writes our results to a file
- fitting.py: fits the results
- custom_widgets.py: builds large sections of the UI into single widgets which are then placed on the main UI
- workers/IV.py: the IV test routine
- workers/Hall.py: the van der pauw routine
- graphing.py: the behaviour of the graphs and how they are drawn
- miscellaneous.py: random functions that didn't really fit anywhere
- IV-test.py: proof of concept to test the system without UI (not necessary)

something that may be new to the reader and cause confusion are the type hints used throughout the code such as "available_name(self, filename: str) - str:" the ':str' has no affect on the code it is simply to tell the reader that this function should recieve filename as a string and the 'str' after the arrow tells the reader that this function will return a string. After you understand what this is it can save you a ton of time understanding what is going on and I encourage you to type hint code that you add as well. If the functions aren't returning anything I sometimes leave the return type blank instead of the proper 'None' type.

3 Code Specifics

3.1 GUI-Main

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/GUI-main.py. The main file is mostly concerned with setting up the visuals and connecting the signals to the right slots (i.e. connecting buttons to there corresponding function etc.). It also sets up the threads for the two measurements to be used. To understand what's going on here you mostly need a tutorial in PyQt, which is outside the scope of this document. I will link to some resources in the resource section that will hopefully get you started.

3.2 File Writing

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/FileWriting.py. The FileWriter class is responsible for writing our results in the proper format to a file and is loaded into the main file as self.writer. It's setter methods from roughly line 20 to line 33 are directly connected to the .textChanged() method of the corresponding input boxes, this can be seen the GUI-main file in the "connectSignals" method. This means that when the text in say sampleID is changed the filewriter class automatically updates its stored value of sampleID so as long as you change the name before the writing is done the updated name will be used. The "available_name" method takes a potential name for a file as a string and checks if that name is taken. If it is, it adds a number to the name and returns the new name as a string. The "write_to_file" method takes the results of the measurement as a dictionary (this is sent by the fitter) and then writes those results to a text file in the correct format.

3.3 Fitter

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/fitting.py. The Fitter class is responsible for taking data from either of the two measurements and returning the results of the analysis in the proper format. The "fit" method takes the 8 lines from the Van der Pauw measurement and fits them individually with np.polyfit and returns the slopes as a list. It also emits a signal of the R-squared value which is connected to the displays below the graph as well as the rSqrd value stored in the writer. "IVfit" takes a single line from the IV measurement, fits it, and then emits that slope to be shown in the UI. "calculateResults" is called immediately after the measuring part of the Van der Pauw routine is done. It takes a dictionary that contains relevant sample information and the data and follows a routine heavily borrowed from the existing labVIEW code to get the results which are then emitted to be displayed and written to a file. In line 80 there is a while loop to solve the equation for f on pg.10 of the Hall Effect Measurement Handbook by Jeffrey Lindemuth, this is a neat little trick from LabVIEW that is faster than the root finding methods in the scipy library and it only uses base python which is nice.

3.4 custom_widgets

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/custom_widgets.py. This file is concerned with building pieces of the UI that will be pieced together in the GUI-main file. If you wish to add more buttons and such you should probably add them to the widgets in this file, they can also be added directly to the main window in GUI-main but doing all that stuff in the main file takes up way too much space. Unfortunately doing it this way adds an additional layer when trying to access the lowest level widgets.

3.5 workers/IV.py

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/workers/IV.py. When you press the go button in the IV tab the IVWorker is set up and then its takeIVMeasurement method is called. The setting up of the worker involves passing the voltmeter and other instruments to the method, and setting the inputs on these instruments. The number of points to take is hard coded to 11 in the setInputs method, this is easy enough to change if desired. When the measurement is taken the program starts by ensuring that the devices are at zero and they have been cleared, then it follows the procedure. The details of the ascii commands can be found in the word document in "Z:/labview-projects/Manuel-Hall Measurement VI". The dataPoint emission's are sent to the graph to be plotted and the lineSgnl emissions are sent to the fitter, finished tells the system that the task is done at which point a number of things are done. All these signals are connected using the "connectSignals" method which is called in the IVGo method of GUI-main.

3.6 workers/Hall.py

link to github: https://github.com/Elliot-Wadge/Hall-GUI/blob/master/workers/Hall.py. When you press go in the Hall tab, this worker is set up and its takeHallMeasurement methods are called. The documentation in the code is the best way to understand this function but I will cover some things not talked about in the code. dataPoint is connected to the graphs to update them in real time, lines is connected to the fitter calculateResults, fieldState and switchSgnl are connected to the textwidgets in the status bar at the bottom of the Window. These are all connected in the hallGo method of GUI-main. Again the commands sent to the devices in this function can be found in "Z:/labview-projects/Manuel-Hall Measurement VI".

4 Dependencies

the following packages are required to run the program

- PyQt5: pip install PyQt5
- numpy: pip install numpy
- PyVisa: pip install PyVisa
- QChart: pip install PyQtChart

5 Learning Resources

- introduction to classes https://www.w3schools.com/python/python_classes.asp
- more in depth classes https://realpython.com/python3-object-oriented-programming/
- introduction to pyqt (first three videos are a good start) https://www.youtube.com/watch?v=Vde5SH8e1OQ&list=PLzMcB
lB8MZfHPLTEHO9zJDDLpYj
- book on pyqt https://www.amazon.ca/Mastering-GUI-Programming-Python-cross-platform/dp/178961290X/ref=asc_df_17
20&linkCode=df0&hvadid=80608037717144&hvnetw=o&hvqmt=e&hvbmt=be&hvdev=c&hvlocint=&hvlocphy=&hvtargi
4584207582640366&psc=1
- Threading https://realpython.com/python-pyqt-qthread/
- PyVisa documentation https://pyvisa.readthedocs.io/en/latest/
- the github for this project https://github.com/Elliot-Wadge/Hall-GUI