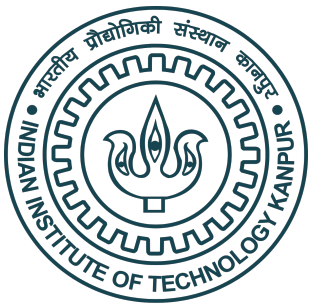

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

END-TERM REPORT

EE496: UNDERGRADUATE PROJECT I

Magneto-electric Measurements and Driver Synthesis for Spintronic Equipments using Python

Submitted by:

Nikhil Jain
Roll No: 220709
Email: nikhilj22@iitk.ac.in
Department of Electrical Engineering

Academic Year: **2024–25 (Even Semester)**

Supervised by:

Dr. Arnab Bose
Department of Electrical Engineering

Contents

1 Abstract

This project aims to automate the control and synchronization of multiple electronic instruments to perform precise magnetoelectric measurements. By integrating a programmable current supply (Kikusui PBZ series), a precision source-measure unit (Keysight), and a lock-in amplifier, we measure the electrical response—such as voltage or resistance—of a test device subjected to a controllable magnetic field generated by Helmholtz coils. The setup supports flexible measurement modes, including IV characterization and voltage detection under varying magnetic fields, either independently or simultaneously. A custom-built software interface ensures synchronized instrument control, configurable triggering, and streamlined data acquisition. This system provides a robust platform for studying magneto-transport phenomena in novel electronic or spintronic materials.

2 Introduction

The development of spintronic devices, which leverage the spin of electrons for information processing, requires precise measurement of magneto-electric properties to characterize material behavior under varying magnetic fields. Manual control of laboratory instruments like power supplies, source-measure units (SMUs), and lock-in amplifiers is time-consuming, error-prone, and limits experimental scalability. Automating these measurements is critical for advancing research in spintronics, enabling high-throughput data collection, reproducibility, and the ability to explore complex magneto-transport phenomena in novel materials. This project addresses the problem by developing Python-based driver scripts and a measurement system to synchronize and control a Kikusui PBZ60 power supply, a Keysight B2900 SMU, and a Stanford Research Systems SR830 lock-in amplifier. The system facilitates automated current-voltage measurements with real-time visualization, supporting experiments in electronics and physics research labs.

3 Standard Operating Procedure for Instrument Driver Scripts for SR830, PBZ60, and B2900 devices

3.1 Objective / Purpose

This SOP describes the setup and use of Python driver scripts to control three laboratory instruments:

- **Stanford Research Systems SR830 Lock-in Amplifier**

- **KIKUSUI PBZ60-3.3A Power Supply**

– **Keysight B2901 Source Measure Unit (SMU)**

These scripts allow automated instrument control for use in experiments and data acquisition pipelines.

3.2 Scope

This SOP is applicable in lab environments requiring automated control of the above instruments via a PC using GPIB or USB interfaces. It is intended for use in electronics, physics, and instrumentation research labs where device communication is necessary for data collection and automation.

3.3 Prerequisites / Requirements

Download the driver files and all the main codes for the instruments from:
Click here

Software:

- Python 3.9 (strictly for Keysight driver compatibility)
- Virtual environment (optional but recommended)
- Python packages:
  - pyvisa
  - pyvisa-py
  - Keysight's .whl driver for B2900
  - PBZ's driver file (depending on x64 or x86) from here.

Hardware:

- PC or laptop with GPIB-USB adapter
- SR830 Lock-in Amplifier
- KIKUSUI PBZ60-3.3A Power Supply
- Keysight B2900 SMU

3.4 Procedure / Steps

3.4.1 Install Dependencies

```
python -m venv instruments_env
source instruments_env/bin/activate   # Windows: instruments_env\
    Scripts\activate
pip install pyvisa pyvisa-py
pip install "C:/path/to/keysight_ktb2900-1.0.1-cp39-cp39-win_amd64.whl"
```

3.4.2 File Organization

```
project_root/
 sr830.py              # SR830 driver
 pbz60.py              # PBZ60 driver
 b2900.py              # B2900 driver
 main.ipynb               # User's script [based on demands of user]
```

3.4.3 How to write code to run instruments in main.ipynb ?

These are example snippets.

SR830 Lock-in Amplifier

```
from sr830 import SR830
sr = SR830(name="lockin", address="GPIB0::8::INSTR")
voltage = sr.get_voltage()
```

PBZ60 Power Supply

```
import pbz60
pbz = pbz60.PBZController(connection_type="USB", resource="GPIB0::1::
    INSTR")
pbz.set_voltage(12.0)
pbz.output_on()
```

B2900 SMU

```
from b2900 import B2900Controller
b2900 = B2900Controller(address="GPIB0::23::INSTR")
b2900.set_voltage(1.5)
current = b2900.measure_current()
```

3.4.4 Run and Operate Instruments

1. Turn on all instruments (KIKUSUI, Keysight, SR830 Lock-in Amplifier).
2. Start the computer and open the working directory or required software folder.
3. Open VS code and create a new file : lets name it main.ipynb (jupyter notebook), for writing your code (if in future for anything) . keep the environment qcodes for running.
4. Any other environment is all good , until you install dependencies (6.1).
5. Run the control script (main.ipynb).

3.5 Troubleshooting / Errors

Issue	Solution
`VisaIOError` or instrument not responding	Ensure instrument is on and connected. Check GPIB address. Try tightening the GPIB cable screws for being confirm that their is no issues regarding the connections.
`ModuleNotFoundError` for `b2900`	Reinstall the Keysight .whl package using correct Python version.
Communication timeout	Add small delays between commands or use VISA query methods.
Driver not working on non-Windows OS	Keysight driver may not be cross-platform; use only on Windows.

Table 1: Troubleshooting Issues and Solutions

To check if the connections are set with the instruments :
Run this code :

```
import pyvisa
rm = pyvisa.ResourceManager()
resources = rm.list_resources()
for res in resources:
    print(res)
```

The output should contain :

Resource	Instrument
GPIB0::1::INSTR	for pbz60
GPIB0::8::INSTR	for sr 830 lock in amplifier
GPIB0::23::INSTR	for b2900

Table 2: Expected Instrument Connections

```
import pyvisa

rm = pyvisa.ResourceManager()
print(rm.list_resources())
[2]  ✓ 0.2s
('ASRL1::INSTR', 'GPIB0::1::INSTR', 'GPIB0::8::INSTR', 'GPIB0::23::INSTR', 'USB0::0x2A8D::0x9101::MY60440182::0::INSTR')
```

Figure 1: example output

If any of this is not shown `VisaIOError` is likely to show.

3.6 Expected Results / Output

- **SR830**: Returns measured voltage or frequency values.
- **PBZ60**: Successfully sets and maintains output voltage.
- **B2900**: Returns measured current or voltage from connected circuit.

Correct functioning means each instrument responds to queries and commands without errors, and data can be logged or used for experiment control.

3.7 User-Configurable Parameters

- **SR830**

Configurable Variable	**Function**	**Allowed Values**
INPUT_COUPLING	Sets the input coupling mode	'A', 'B', 'AC', 'DC'
REFERENCE_SOURCE	Selects the source of the reference signal	'internal', 'external'
EXT_TRIGGERS	Configures the external trigger mode	'sine'
AMPLITUDE	Sets the voltage amplitude of the output signal	–
FREQUENCY	Sets the reference frequency for signal detection	–
PHASE	Adjusts the phase of the reference signal	0-360 deg
TIME CONSTANT	Defines the integration time for signal averaging	{1e–5, 3e–5, 0.0001, 0.0003, 0.001, ..., 30000}
SENSITIVITY	Sets the full-scale input voltage sensitivity	{2e–9, 5e–9, 1e–8, ..., 1}

- **PBZ60 (Kikusui)**

Parameter	**Description**
`voltage`	Output voltage level (e.g., up to ±60 V)
`current`	Current limit setting (e.g., up to ±3.3 A)
`mode`	Operating mode: "CV" (voltage) / "CC" (current)
`output_state`	Turn output ON/OFF
`protection`	Overvoltage or overcurrent protection (if supported)

Table 3: PBZ60 User-Configurable Parameters

- **B2901 (Keysight)**
  SCPI (Standard Commands for Programmable Instruments) and Pyvisa library are

the primary modes of instructions to run this instrument, unlike others that run on driver commands and not need any library setup. [though a venv need to be setup downloading the driver files (available in the drive link)]

Except from the major commands there are a list of other commands in the B2900B-BL-Series-Precision-Source-Measure-Unit-Programming-Guide.pdf in the drive link.

Enabling the Source Output

Source output is enabled by the :OUTP ON command.

Example

```
ioObj.WriteString(":OUTP ON")
```

Setting the Source Output Mode

Source output mode is set by the :SOUR:FUNC:MODE command.

Example

```
ioObj.WriteString(":SOUR:FUNC:MODE CURR") 'Current output

ioObj.WriteString(":SOUR:FUNC:MODE VOLT") 'Voltage output
```

Applying the DC Voltage/Current

DC current/voltage is immediately applied by the :SOUR:<CURR|VOLT> command during the source output is enabled.

If you want to control the DC current/voltage output timing using a trigger, use the :SOUR:<CURR|VOLT>:TRIG command. See Figure 1-2.

Example

```
ioObj.WriteString(":SOUR:FUNC:MODE CURR")
ioObj.WriteString(":SOUR:CURR 1E-3")     'Outputs 1 mA immediately

ioObj.WriteString(":SOUR:FUNC:TRIG:CONT 1")
ioObj.WriteString(":SOUR:FUNC:MODE VOLT")
ioObj.WriteString(":SOUR:VOLT:MODE FIX")
ioObj.WriteString(":SOUR:VOLT:TRIG 1")   'Outputs 1 V by a trigger
```

Stopping the Source Output

Source output is stopped and disabled by the :OUTP OFF command.

Example

```
ioObj.WriteString(":OUTP OFF")
```

Figure 2: snippet of the pdf

Parameter	Description
`voltage`	Set output voltage
`current`	Set output current (in current source mode)
`mode`	Source mode: "VOLT" or "CURR"
`compliance`	Current or voltage limit to prevent damage
`range`	Manual/Auto range for voltage or current measurement
`measure_mode`	What to measure: "VOLT", "CURR", etc.
`trigger`	Trigger settings for synchronous measurement
`output_state`	Enable or disable channel output
`sweep_params`	Start, stop, step size, dwell time for IV sweeps

Table 4: B2900 User-Configurable Parameters

For all these settings there are commands in the driver file that we can find and then just set it as per our wishes.

3.8 Loop and Live Data plotting

I referred to this playlist for basics of loops and livedata plotting and saving the data to the csv files :
Qcodes playlist
This would be used in getting data from multiple devices together and live plotting them together. This is necessary because if the data is wrong we can check the credibility just by having the few first data points to check.

3.9 Running Multiple devices Together

3.9.1 What is Being Measured?

A loop is set from one device's current that starts from one value and ends to another value , meanwhile voltage is measured from some other device and the result is live plotted.
The system is being swept through a series of current inputs (`xyz.set_current()`), and the response from the system is plotted.
The plot shows *System Response vs Current* with error bars reflecting measurement uncertainty.

3.9.2 Key points to know

Parameter	Description
`number_of_points`	Defines how many distinct current values will be tested between `start_Current` and `End_current`. Usage: The code uses `np.linspace(start_Current, End_current, number_of_points)` to generate equally spaced current values for the sweep.
`number_of_repeats`	How many times is the whole experiment repeated. Usage: Repeating measurements helps get better and reach more reliable conclusions.
`sampling_points`	Number of data samples collected per measurement repetition. Usage: Used to gather enough signal samples from the device for a single value point which is then averaged out to remove any noise and increase signal fidelity. Standard deviation is also calculated which is shown as error bars while plotting and `sampling_points >1`.
`time_of_sleep`	Time delay (in seconds) between setting a current and taking a measurement. Usage: Ensures the system stabilizes before capturing data (important in systems with settling time or lag).
`trace_mode`	A boolean or mode flag that if true measure the code from front to back and then from back to front. Usage: It may be necessary to get additional data without abruptly switching the current to the `start_ h`.

Table 5: Key Parameters for Running Multiple Devices

Key points

- Multiple samples and repetitions enhance accuracy.
- Real-time plotting and export functions are included for analysis.

4 Measurement System for PBZ60 Power Supply and SR830 Lock-in Amplifier

Driver Code : `pbz_sr.py`
Test : `pbz_sr_test.ipynb`

4.1 Introduction

This report provides a detailed explanation of a Python-based measurement system designed to interface with a PBZ60 power supply and an SR830 lock-in amplifier. The system facilitates automated current-voltage measurements with real-time data visualization using live plotting. The code, implemented in the `pbz_sr.py` file, allows users to control the power supply, collect data from the lock-in amplifier, and visualize the results dynamically. The system supports multiple measurement repeats, data saving in CSV and text formats, and plot exporting as images. This report describes the code's purpose, the libraries used, the measurement loop, the functions implemented, and the user interface of the live plotter.

4.2 Purpose of the Code

The code implements a `Plotter` class that automates the process of sweeping a current range using the PBZ60 power supply while simultaneously measuring X and Y channel outputs from the SR830 lock-in amplifier. The system is designed for experiments requiring precise current control and sensitive voltage measurements, such as characterizing electronic devices or materials.
In synopsis: Looping current from `start_current` to `end_current` in Pbz60 and plotting the complex voltages measured (x and y) of SR830 on the y axis.

Key features include:

- **Current Sweep**: Linearly varies the current from a start to an end value over a specified number of points.
- **Repeated Measurements**: Performs multiple measurement cycles, optionally reversing the current sweep direction (trace mode).
- **Data Collection**: Collects multiple samples per current point to compute mean and standard deviation for X and Y channels.
- **Real-Time Visualization**: Displays live plots of X and Y measurements versus current, including error bars for standard deviations.
- **Data Storage**: Saves measurement data to CSV and text files and exports plots as PNG images.
- **User Interaction**: Provides a graphical user interface (GUI) with buttons for starting, stopping, saving, and loading data.

4.3 Libraries Used

The code leverages several Python libraries to handle numerical computations, GUI creation, live plotting, and file operations. Below is a detailed description of each library and its role:

Library	Role
`NumPy (numpy)`	Used for numerical operations, particularly for generating an array of current values using `np.linspace` and converting lists to arrays for plotting.
`Time (time)`	Provides the `time.sleep` function to introduce delays between measurements, ensuring stable readings from the SR830.
`Statistics (statistics)`	Computes the mean and standard deviation of measurement samples using `statistics.mean` and `statistics.stdev`.
`PyQtGraph (pyqtgraph)`	A plotting library used for creating interactive, real-time plots. It provides: `GraphicsLayoutWidget` for organizing multiple plots, `PlotItem` for plotting X and Y versus current with symbols and error bars, `ErrorBarItem` for visualizing standard deviations, `InfiniteLine` for adding movable vertical lines to the plots, `ImageExporter` for saving plots as PNG files.
`PyQt5 (pyqtgraph.Qt)`	Provides the GUI framework through `QtWidgets` (e.g., `QMainWindow`, `QPushButton`, `QLabel`) and `QtCore` (e.g., `QTimer` for scheduling measurements).
`CSV (csv)`	Facilitates writing measurement data to CSV files using `csv.writer`.
`PBZ60 and SR830 Drivers (pbz60,sr830)`	Custom drivers for interfacing with the PBZ60 power supply (`set_current` method) and SR830 lock-in amplifier (`snap` method for X and Y readings).

Table 6: Libraries Used in `pbz_sr.py`

```
import numpy as np
import time
import statistics
import pyqtgraph as pg
from pyqtgraph.Qt import QtWidgets, QtCore
import pyqtgraph.exporters
import sys
import csv
import os
from datetime import datetime
from pbz60 import PBZController
from b2900 import B2900Controller
```

Figure 3:

4.4 Code Structure and Key Functions

4.4.1 Initialization (__init__)

The `__init__` method initializes the `Plotter` instance with parameters defining the measurement setup:

- `pbz`: PBZ60 power supply object.
- `sr`: SR830 lock-in amplifier object.
- `start_Current`, `End_current`: Range of current values for the sweep.
- `number_of_points`: Number of current points in the sweep.
- `number_of_repeats`: Number of measurement cycles.
- `sampling_points`: Number of samples per current point.
- `time_of_sleep`: Delay between samples (in seconds).
- `trace_mode`: Boolean indicating whether to reverse the current sweep direction after each repeat.
- `note_string`: A string for annotating saved data files.

It initializes:

- A list of current values using `np.linspace`.
- State variables (`running`, `index`, `reverse`, `current_repeat`, `all_data`).
- Data storage lists (`current_values`, `x_means`, `x_stds`, `y_means`, `y_stds`).
- The PyQt5 application and GUI via `setup_ui`.

4.4.2 GUI Setup (setup_ui)

The `setup_ui` method creates the GUI using PyQt5 and PyQtGraph. The interface includes:

- **Window**: A `QMainWindow` titled "Live Measurement Plot."
- **Top Info Section**:
  - A label showing the current repeat (Repeat: 0/N).
  - A label indicating the trace mode status (Toggle is True/False).
- **Plot Area**:
  - Two plots (`plot_x`, `plot_y`) for X and Y versus current.
  - Each plot includes a curve (`x_curve`, `y_curve`), error bars (`x_error`, `y_error`), and a movable vertical line (`x_vline`, `y_vline`).
- **Control Buttons**:
  - `Start`: Initiates the measurement process.
  - `Stop`: Pauses or resumes measurements.
  - `Save`: Saves data and plots manually.
  - `Load Data`: Loads previously saved data for plotting.
- **Status Labels**:
  - `alert_label`: Displays messages (e.g., "Data Saved!").
  - `stats_label`: Shows real-time statistics (current, X, Y, and their standard deviations).

The method connects button clicks to corresponding methods and sets up an escape key handler to quit the application.

4.4.3 Measurement Loop and Current Control (measure_next_point)

The `measure_next_point` method is the core of the measurement process, implementing a recursive loop to sweep through current values and collect data. It is scheduled using `QtCore.QTimer.singleShot` to avoid blocking the GUI. The loop operates as follows:

- **Check Running State**:
  - If `self.running` is `False`, the method exits.
- **Check Current Index**:
  - If `self.index` exceeds the number of current points, it increments `self.current_repeat`.
  - If `self.current_repeat` reaches `self.number_of_repeats`, the measurement stops, data is saved, and the application quits.
  - Otherwise, it resets the index, clears data lists, and optionally reverses the current list (if `trace_mode` is `True`).

- **Set Current**:
  - Sets the current on the PBZ60 using `self.pbz.set_current(current)`.
- **Collect Samples**:
  - Loops `self.sampling_points` times, pausing for `self.time_of_sleep` seconds between samples.
  - Calls `self.sr.snap('x', 'y')` to read X and Y values from the SR830.
  - Stores samples in `x_data` and `y_data` lists.
- **Compute Statistics**:
  - Calculates mean and standard deviation for X and Y using `self.mean_and_std`.
- **Store Data**:
  - Appends the current, X mean, X std, Y mean, and Y std to respective lists.
  - Stores the full data tuple (repeat, current, X mean, X std, Y mean, Y std) in `self.all_data`.
- **Update Plot and Stats**:
  - Calls `self.update_plot` to refresh the plots.
  - Updates `self.stats_label` with the current, X, and Y statistics.
- **Schedule Next Measurement**:
  - Increments `self.index` and schedules the next call to `measure_next_point` after 10 ms.

4.4.4 Plot Updating (update_plot)

The `update_plot` method refreshes the X and Y plots:

- Converts data lists to NumPy arrays for efficiency.
- Updates `x_curve` and `y_curve` with current versus mean values.
- Updates `x_error` and `y_error` with standard deviations as error bars.

4.4.5 Start Measurement (start_measurement)

The `start_measurement` method initializes a new measurement session:

- Sets `self.running` to `True`.
- Hides the save button.
- Resets state variables and data lists.
- Updates the repeat label and plot.
- Starts the measurement loop by calling `measure_next_point`.

4.4.6 Stop/Resume Measurement (stop_measurement)

The `stop_measurement` method toggles the measurement state:

- Flips `self.running` to pause or resume.
- Shows the save button when paused.
- If paused and data exists, displays final statistics (mean and max of X and Y).
- Updates the stop button text to "Resume" or "Stop."
- If resumed, restarts the measurement loop.

4.4.7 Save Data and Plots (save)

The `save` method saves measurement data and plots:

- Generates a filename based on the current date (`measurement_YYYYMMDD`).
- Saves data to a CSV file with headers (Repeat,Current,X,X_std,Y,Y_std).
- Saves data to a text file with the `note_string` as a header and tab-separated values.
- Creates a `plots` directory and exports X and Y plots as PNG files with timestamps.
- If not automatic (`auto=False`), displays a "Data & Plots Saved!" message.

Image of the x vs current saved.

4.4.8 Load Data (load_data)

The `load_data` method loads previously saved data:

- Opens a file dialog to select a .txt or .csv file.
- Clears existing data lists.
- Reads the file, skipping the header, and populates data lists with current, X mean, X std, Y mean, and Y std.
- Updates the plots and displays a "Data Loaded" message.

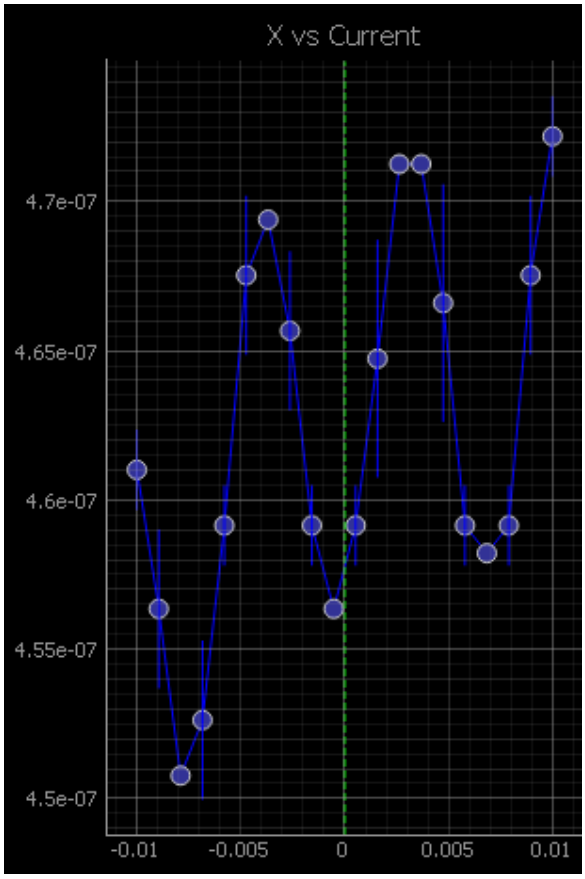

Figure 4: X vs Current Plot (saved file)

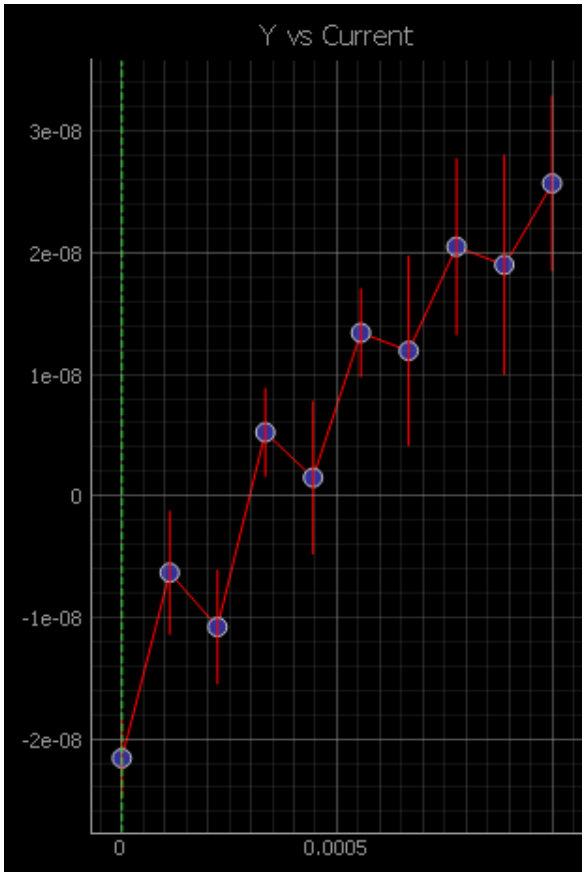

Figure 5: Y vs Current Plot (saved file)

4.4.9 Mean and Standard Deviation (mean_and_std)

The `mean_and_std` method computes the mean and standard deviation of a data list using the `statistics` module.

4.5 Live Plotter Interface Description

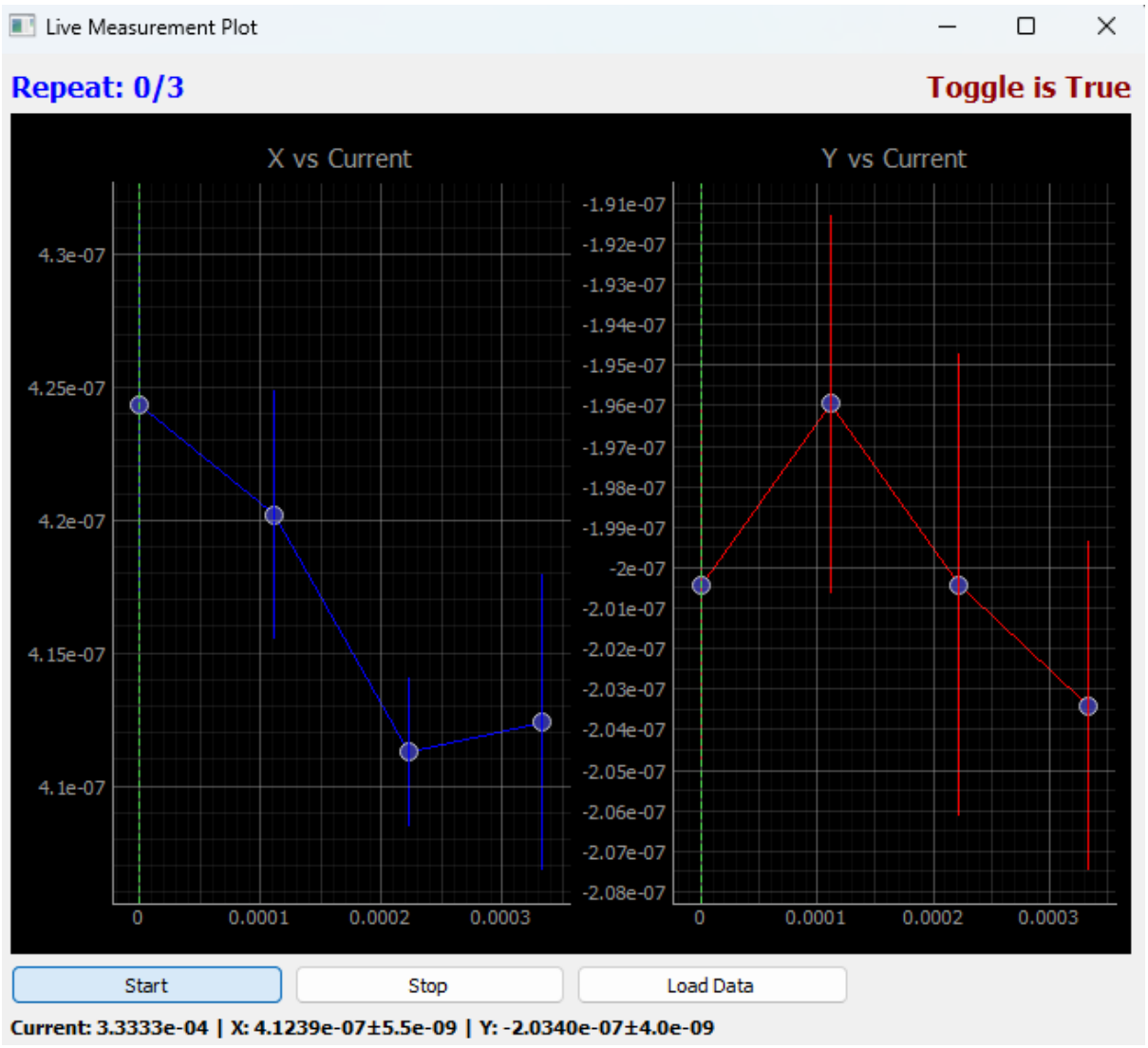

Figure 6: Running position GUI, note the current reading is displayed in the bottom. The error bars denote the standard deviation of the sample points which were measured and averaged out to get a single data point in the plot. On the top bar- there is toggle - true : means the experiment will run back and forth, also repeat 0/3 means its the first set of the 3 trials run on the machine

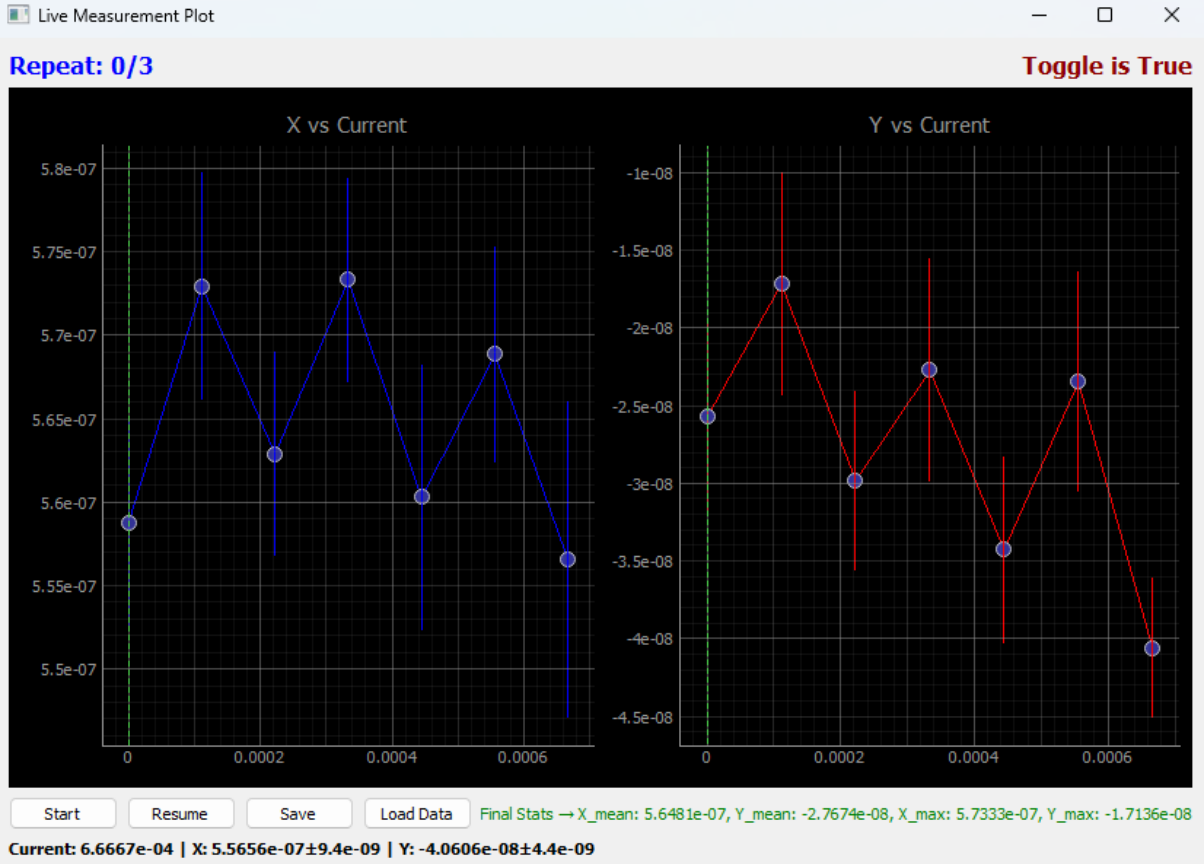

Figure 7: Stop position GUI, note the details of the dataset is provided at bottom, green font color

4.5.1 Window Layout

Title: "Live Measurement Plot."
Top Info Section:

- **Repeat Label**: Displays the current repeat (e.g., "Repeat: 0/5") in bold blue text.
- **Toggle Status Label**: Shows the trace mode status (e.g., "Toggle is True") in bold dark red text.

Plot Area:

- **X Plot**: A plot of X mean versus current (blue line with circle markers).
  - Includes blue error bars for X standard deviation.
  - Features a movable green dashed vertical line (`x_vline`) for reference.
  - Displays a grid for readability.
- **Y Plot**: A plot of Y mean versus current (red line with circle markers).
  - Includes red error bars for Y standard deviation.
  - Features a movable green dashed vertical line (`y_vline`).
  - Displays a grid.

Control Section:

- `Start Button`: Initiates the measurement process.
- `Stop Button`: Pauses or resumes measurements (text toggles to "Resume" when paused).
- `Save Button`: Saves data and plots (hidden during measurements).
- `Load Data Button`: Opens a dialog to load saved data for plotting.
- `Alert Label`: Displays status messages (e.g., "Data Saved!") in green.
- `Stats Label`: Shows real-time statistics (e.g., "Current: 1.0000e-03 — X: 2.3456e-06±1.2e-07 — Y: 3.4567e-06±1.5e-07") in bold text.

4.5.2 Visual Appearance

The plots update in real-time as measurements are taken, with points added sequentially. Error bars provide a visual indication of measurement variability.
The movable vertical lines allow users to mark specific current values for reference.
The interface is clean, with bold labels and color-coded elements (blue for X, red for Y, green for alerts) for clarity.

4.5.3 User Interaction

- **Starting a Measurement**: Clicking "Start" clears previous data, resets the plots, and begins the current sweep.
- **Pausing/Resuming**: Clicking "Stop" pauses the measurement, showing the save button and final stats. Clicking "Resume" continues from the current point.
- **Saving**: Clicking "Save" generates CSV, text, and PNG files with timestamps.
- **Loading Data**: Clicking "Load Data" opens a file dialog, and selected data is plotted immediately.
- **Exiting**: Pressing the Escape key closes the application.

4.6 Conclusion

The `pbz_sr.py` code provides a robust and user-friendly system for automated measurements using the PBZ60 power supply and SR830 lock-in amplifier. It integrates precise current control, sensitive voltage measurements, and real-time data visualization through a PyQtGraph-based GUI. The measurement loop efficiently handles multiple repeats and samples, while the interface allows users to start, stop, save, and load data with ease. The use of libraries like NumPy, PyQtGraph, and PyQt5 ensures efficient computation, plotting, and interaction. This system is well-suited for laboratory experiments requiring high-precision measurements and dynamic data analysis.

5 Measurement System for PBZ60 Power Supply and B2900 Source Measure Unit

Driver Code : `pbz_b2900.py`
Test : `pbz_b2900_test.ipynb`

5.1 Introduction

This report describes a Python-based measurement system (`pbz_b2900.py`) designed to interface with a PBZ60 power supply and a B2900 source measure unit (SMU). The code is similar to the previous system (`pbz_sr.py`), which interfaced with a PBZ60 power supply and an SR830 lock-in amplifier, but introduces specific changes to accommodate the B2900 SMU. The system automates current-voltage measurements with real-time data visualization, supporting multiple measurement loops, data saving in CSV and text formats, and plot exporting as PNG images. This report focuses on the changes made in `pbz_b2900.py` relative to the previous code, detailing the updated purpose, libraries, measurement loop, functions, and user interface.

5.2 Purpose of the Changed Code

The updated code implements a `MeasurementApp` class that automates sweeping a current range using the PBZ60 power supply while applying constant currents via the B2900 SMU and measuring its voltage output.
In synopsis: Looping current from `start_current` to `end_current` in Pbz60 and plotting the voltages measured of B2901 while at constant current mode (CC) on the y axis. The `keysight_current_value` is a variable that can be set by choice.

Key changes include:

- **Instrument Change**: Replaces the SR830 lock-in amplifier with the B2900 SMU, focusing on voltage measurements instead of X and Y channel outputs.
- **Current Control**: Introduces a list of constant B2900 (Keysight) current values, cycled through during measurements.
- **Sweep Direction**: Replaces trace mode with explicit forward and backward sweeps within each loop.
- **Data and Plotting**: Stores and plots voltage versus PBZ current, with separate curves for forward and backward sweeps on a single plot.

5.3 Changed Libraries

The libraries remain largely the same, with the following change:

- **B2900 Driver (b2900)**: Replaces the SR830 driver, providing methods like `measure_voltage`, `apply_current`, and `set_source_mode` for interfacing with the B2900 SMU.

5.4 Changed Code Structure and Functions

5.4.1 Initialization (__init__)

Class Name: Renamed from `Plotter` to `MeasurementApp`.
New Parameters:

- `pbz_resource`, `b2900_resource`: Specify instrument connection details.
- `keysight_current_values`: A list of constant B2900 current values.
- `expt_name`: A string for naming output files.

Removed Parameter: `trace_mode`, replaced by a `forward` boolean flag for sweep direction.
Data Storage: Stores PBZ current, B2900 current, and B2900 voltage mean/std, replacing X/Y mean/std.

5.4.2 Instrument Setup (connect_instruments)

Configures the B2900 as a current source with voltage compliance (default 10V), instead of SR830 X/Y measurements.
Sets B2900 to current source mode (`set_source_mode("CURR")`) and applies initial zero current.

5.4.3 GUI Setup (setup_ui)

Window Title: Changed to "PBZ60 & B2900 Measurement."
Top Info: Displays loop number and direction (Forward/Backward) instead of repeat count and trace mode.
Plotting:

- Consolidates into a single plot (voltage vs. PBZ current) instead of two (X and Y vs. current).
- Uses two curves: forward (blue, circles) and backward (red, crosses), with a legend.
- Removes error bars and movable vertical lines.

Controls: Retains Start, Stop, Save, and Load Data buttons, but updates stats label to show PBZ current, B2900 current, voltage, and direction.

5.4.4 Measurement Loop (measure_next_point)

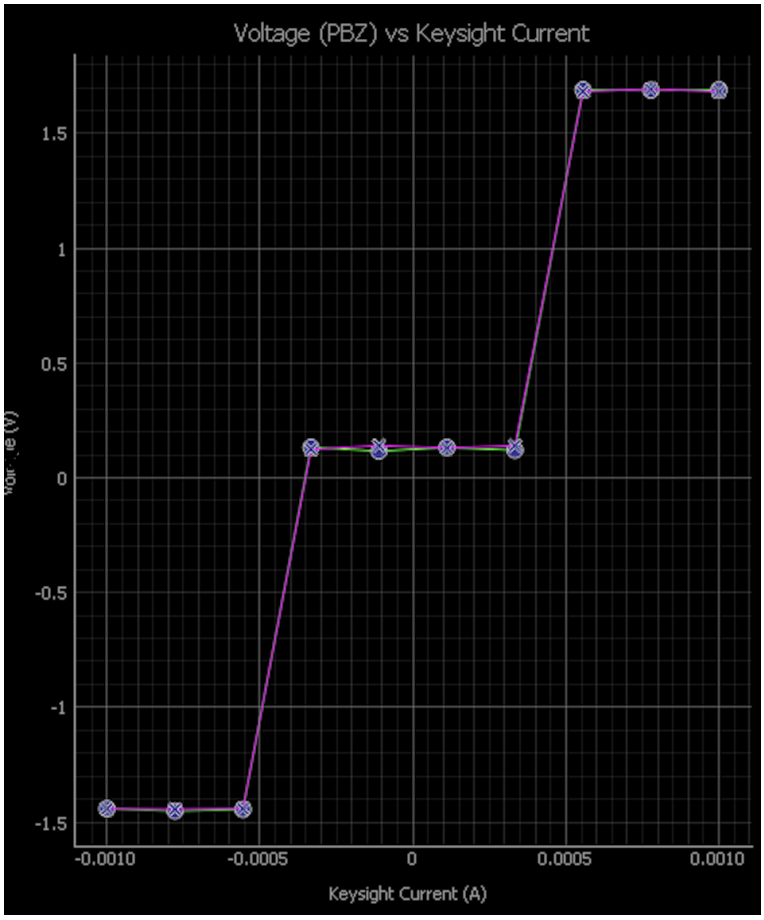

Figure 8: plot, note trace on (forward and backward plotting)

Sweep Logic:

- Implements forward and backward sweeps within each loop, toggling the `forward` flag.
- Stops after `number_of_loops` (not repeats).

B2900 Control:

- Sets B2900 current from `keysight_current_values`, cycling through the list.
- Measures B2900 voltage only, with a fixed 0.5-second settling time before sampling.

Data Storage:

- Stores loop number, direction, PBZ current, B2900 current, and B2900 voltage mean/std.
- Maintains `current_loop_data` for the current loop's plot.

Plot Update:

- Clears plots at the start of each loop.
- Saves the current loop's plot before clearing.

Stats Label: Displays PBZ current, B2900 current, voltage mean/std, and direction.

5.4.5 Other Changed Functions

- `update_plot`:
  - Separates data by direction (forward/backward) for the current loop.
  - Updates a single plot with forward and backward curves.
- `start_measurement`:
  - Initializes with loop count (starting at 1) instead of repeats.
  - Clears `current_loop_data` and sets direction to Forward.
- `save`:
  - Saves data with headers for loop, direction, PBZ current, B2900 current, and voltage mean/std.
  - Includes `expt_name` in filenames and saves plots per loop.
- `load_data`:
  - Loads data with updated headers, displaying the latest loop's data.
- `clear`: New function to reset `current_loop_data` and instruments.
- `cleanup`: New function to safely close instrument connections, setting currents to zero and disabling outputs.
- `measure_voltage`: New function to query B2900 voltage, replacing SR830's `snap`.

5.5 Changed Live Plotter Interface

Window: Titled "PBZ60 & B2900 Measurement," resized to 1000x700.
Top Info:

- Shows loop number and direction (e.g., "Loop: 1/5 — Direction: Forward") in bold blue.

Plot:

- Single plot of voltage vs. PBZ current.
- Forward curve (blue, circles) and backward curve (red, crosses) with a legend.
- Grid enabled, labeled axes (Voltage in V, PBZ Current in A).
- No error bars or movable lines.

Controls:

- `Start`: Begins measurement, clearing previous data.
- `Stop`: Pauses/resumes, showing Save button when paused.
- `Save`: Saves data and current loop's plot.

- `Load Data`: Loads saved data, showing the latest loop.
- `Alert Label`: Shows status (e.g., "Measurement in progress...") in green.
- `Stats Label`: Displays PBZ current, B2900 current, voltage mean/std, and direction in bold.

Appearance: Real-time updates with color-coded curves (blue for forward, red for backward) and a simplified layout.

5.6 Conclusion

The `pbz_b2900.py` code adapts the previous measurement system to interface with a B2900 SMU instead of an SR830 lock-in amplifier. Key changes include using the B2900 for voltage measurements, introducing constant B2900 current values, implementing forward/backward sweeps per loop, consolidating plotting into a single voltage vs. current graph, and adding functions for instrument cleanup and per-loop plot saving. The system retains the core functionality of automated measurements and real-time visualization, tailored for experiments requiring precise current sourcing and voltage measurement with the PBZ60 and B2900.

6 Measurement System for PBZ60 Power Supply, B2900 Source Measure Unit, and SR830 Lock-in Amplifier

6.1 Introduction

This section describes a Python-based measurement system designed to interface with three laboratory instruments: the Kikusui PBZ60 power supply, the Keysight B2900 Source Measure Unit (SMU), and the Stanford Research Systems SR830 Lock-in Amplifier. The system automates nested current sweeps, where the B2900 SMU loops through a range of current values, and for each B2900 current, the PBZ60 power supply sweeps through its own current range. Simultaneously, the SR830 measures the X and Y components of the complex voltage. The system supports real-time data visualization, multiple measurement loops, and data saving/exporting.

6.2 Purpose of the Code

The code implements a `ThreeInstrumentMeasurement` class that automates nested current sweeps and voltage measurements:

- The B2900 SMU sweeps current from a specified start to end value.
- For each B2900 current, the PBZ60 power supply sweeps current across its own range.
- The SR830 measures X and Y components of the complex voltage for each combination of currents.
- Results are plotted as X and Y voltages versus PBZ60 current, with separate curves for each B2900 current value.

Key features:

- **Nested Current Sweeps**: B2900 current as the outer loop, PBZ60 current as the inner loop.
- **Forward and Backward Sweeps**: Each PBZ60 sweep includes both directions per B2900 current step.
- **Data Collection**: Multiple samples per point for mean and standard deviation of X and Y voltages.
- **Real-Time Visualization**: Live plots of X and Y versus PBZ60 current, with distinct curves per B2900 current and direction.

6.3 Code Structure and Key Functions

6.3.1 Initialization (`__init__`)

The `ThreeInstrumentMeasurement` class initializes with:

- Connection details for PBZ60, B2900, and SR830.
- B2900 current range (`keysight_start_current`, `keysight_end_current`).

- PBZ60 current range (`start_current`, `end_current`).
- Number of points for B2900 and PBZ60 sweeps.
- Number of measurement loops, samples per point, and delay between samples.
- Experiment name for output files.

It sets up:

- B2900 and PBZ60 current arrays using `np.linspace`.
- State variables (running status, indices, sweep direction, loop counter).
- Data storage for currents, X/Y means, and standard deviations.

6.3.2 Instrument Setup (`connect_instruments`)

Configures:

- **PBZ60**: Sets initial zero current.
- **B2900**: Current source mode, zero initial current, 10V voltage compliance.
- **SR830**: Prepares for X and Y voltage measurements.

6.3.3 GUI Setup (`setup_ui`)

The GUI includes:

- **Window**: Titled for three-instrument measurement, sized 1200x800.
- **Top Info**:
  - Loop number and B2900 current in bold blue (e.g., "Loop: 1/5 - Keysight Current: 1.000e-03 A").
  - PBZ60 sweep direction in bold dark red (e.g., "Direction: Forward").
- **Plot Area**:
  - X and Y voltage plots versus PBZ60 current.
  - Multiple curves per B2900 current value (blue circles for forward, red crosses for backward) with a legend.
  - Error bars for standard deviations.
- **Controls**:
  - Start, Stop (toggles to Resume), Save (hidden during measurements), and Load Data buttons.
  - Alert label for status (green) and stats label for real-time data (PBZ60/B2900 currents, X/Y mean/std, direction).

6.3.4 Measurement Loop (`measure_next_point`)

Implements nested sweeps:

- **B2900 Loop**: If PBZ60 index exceeds points, resets it, toggles direction, and increments B2900 index.
- **Loop Termination**: Stops after specified loops, saves data, and exits.
- **Current Setting**: Sets B2900 current, then PBZ60 current (reversed for backward sweep).
- **Sampling**: Collects multiple X and Y voltage samples with delays, computing mean and standard deviation.
- **Data Storage**: Stores loop, currents, direction, and X/Y statistics.
- **Updates**: Refreshes plots and stats label, schedules next measurement.

6.3.5 Plot Updating (`update_plot`)

Groups data by B2900 current and direction, plotting:

- X and Y voltages versus PBZ60 current.
- Color-coded curves with markers and error bars.

6.3.6 Cleanup (`cleanup`)

Closes connections, setting currents to zero and disabling outputs.

7 Results

The measurement systems yielded the following expected results:

- **SR830**: Returns measured voltage or frequency values.
- **PBZ60**: Successfully sets and maintains output voltage.
- **B2900**: Returns measured current or voltage from connected circuits.

Correct functioning is indicated by instruments responding to commands without errors, enabling data logging for experiment control. Real-time plots show X/Y vs. current (PBZ60-SR830) and voltage vs. current (PBZ60-B2900), with error bars reflecting measurement uncertainty. Data is saved in CSV/text formats, and plots are exported as PNGs.

8 Discussion

The measurement systems successfully automate magneto-electric measurements, addressing the challenges of manual instrument control. The PBZ60-SR830 system excels in sensitive voltage measurements under varying currents, suitable for characterizing spintronic materials. The PBZ60-B2900 system, with its focus on voltage measurements at constant currents, complements this by enabling IV characterization. Real-time visualization and error bars enhance data credibility, allowing immediate detection of anomalies. The GUI improves user efficiency, though troubleshooting issues like `VisaIOError` highlight the importance of robust connections and driver compatibility. Limitations include the B2900's Windows-only driver and the need for precise GPIB address configuration. Future improvements could integrate additional instruments or support cross-platform drivers.

9 Conclusion

The project developed robust Python-based systems for automated magneto-electric measurements using the PBZ60 power supply with either the SR830 lock-in amplifier or B2900 SMU. Key learnings include the importance of synchronized instrument control, real-time data visualization, and user-friendly interfaces for laboratory experiments. The systems enable precise current-voltage measurements, supporting research in spintronic materials. The project demonstrates the power of Python for instrument automation, offering a scalable framework for future enhancements in electronics and physics research.

Acknowledgement

I acknowledge the guidance of Dr. Arnab Bose, Department of Electrical Engineering, IIT Kanpur, for supervising this project. Additional thanks to the Qcodes community for their playlist on loops and live data plotting, which informed the implementation of data acquisition and visualization.

I would also like to thank Nayani Om Swarup (230686) for providing the initial updates when I took over the project, and for helping with parts of the code implementation.

References

[1] Driver files and codes for SR830, PBZ60, and B2900, available via provided download links.

[2] Qcodes playlist for loops and live data plotting.

[3] B2900BBL-Series-Precision-Source-Measure-Unit-Programming-Guide.pdf for B2900 command details.