ENPH 455/555 Final Report

Queen's Rotamak Data Acquisition System and Graphical User Interface

by

Jane Cohen

A thesis submitted to the
Department of Physics, Engineering Physics and Astronomy
in conformity with the requirements for
the degree of Bachelor of Applied Science

Queen's University
Kingston, Ontario, Canada
March 2024

Abstract

This thesis presents the design, implementation, and validation of a comprehensive data acquisition system and graphical user interface (GUI) for the Queen's University Rotamak, a nuclear fusion research device. The Rotamak, being at the forefront of exploring magnetic plasma confinement and nuclear fusion, requires advanced instrumentation to capture and analyze high-energy discharge data. To address this need, portable PicoScopes were utilized for their precise voltage signal measurements, coupled with a custom Python script for enhanced control over data collection.

An essential aspect of this work involved the development of a user-friendly GUI, designed to display collected data, and facilitate user interaction for setting and modifying PicoScope parameters without the need for direct script manipulation. This interface leverages PyQt5 for its extensive widget library and event-driven programming model, ensuring a seamless and intuitive user experience.

The iterative design process, from problem definition and hardware selection to software development and system integration is detailed in full. Emphasis is placed on the modular and adaptable nature of the system, allowing for future modifications as the Rotamak project evolves. Despite challenges such as ensuring synchronicity between PicoScopes and time limiting the amount of testing conducted, the system was successfully integrated into the laboratory, demonstrating its capability to support the Rotamak's research objectives.

Future work will focus on enhancing the system's reliability, including implementing error handling and additional features the the GUI, to ensure the system's longevity and utility in the dynamic research of nuclear fusion.

Acknowledgments

I wish to acknowledge Professor Jordan Morelli for his guidance throughout the development of this thesis.

I also extend my gratitude to Daniel Solano, Nathan Eddy and Jérémy Talbot-Pâquet, whose collaborative efforts and technical assistance have been essential in shaping the work presented in this paper.

Contents

List of Tables

List of Figures

Chapter 1

1.1 Introduction

Fusion has the potential to be a widely-used, safe, clean energy source. Its environmentally friendly nature (producing no greenhouse gasses or long-lasting waste) makes it a promising solution for sustainable energy needs. Fusion research also produces technological advancements in diverse fields, resulting in economic benefits, generation of jobs, and groundbreaking innovations. Professor Morelli and his students are building a nuclear fusion reactor device known as a Rotamak to further our understanding of magnetic plasma confinement and nuclear fusion. To study the results of this device, a data acquisition system has been designed for the high energy discharges of the Rotamak. This system uses Picoscopes (portable oscilloscopes) to collect and store the voltage signals produced by the Rotamak. For this portion of the project, a custom script has been developed to interface with the PicoScopes, giving the user control over any number of scopes. In addition, a graphical user interface that displays the results from each discharge has been designed to provide real-time information to users. The first interface window allows users to select a scope and channel to display the data for that respective device as well as run the data collection script. The second window allows users to adjust the parameters of each PicoScope channel and trigger that are used for the data collection system.

Chapter 2

2.1 Background

2.1.1 PicoScope

The data collection part of the system uses PicoScope-4227 devices. A PicoScope is a portable, USB-powered oscilloscope from Pico Technology that records highly accurate voltage measurements. The PicoScope-4227 is able to sample at a rate of 125 MS/s (mega-samples per second) with 12-bit resolution [1]. The device supports multiple triggering methods and sampling modes that allow the user to control when and how data collection occurs [2]. For this project, custom code has been developed to interface with the devices to increase control of the PicoScopes' functionality instead of using a proprietary desktop application.

Manually programming a PicoScope involves using the PicoScope Software Development Kit (SDK), which provides an Application Programming Interface (API). The SDK facilitates the control of data collection using the PicoScope 4000 API driver, ps4000.dll, which provides all the API functions used to interface with the scope. These API functions allows users to write custom software to automate all aspects of the scope's functionality.

The SDK supports various programming languages by providing users with wrapper classes. The Python wrapper class enables the use of ctypes in Python, allowing for direct calls to the C functions provided by the PicoScope API within Python code [3].

2.1.2 PyQt5 and QtDesigner

For the development of the graphical user interface (GUI), QtDesigner and PyQt5 were used. QtDesigner is a desktop application that allows users to create user interfaces (UIs) using a drag-and-drop method. The resulting UI designs can be exported as Python scripts, providing a foundation for further manual customization.

PyQt5 is a widely-used library that allows programmers to use Qt GUI software in Python. The core of PyQt5 is the use of "widgets", which are the building blocks of all PyQt applications. These widgets range from simple elements like buttons, labels, and frames to more complex components like group boxes and graphs. In addition, PyQt5 uses a signal and slot mechanism which allows for communication between different parts of the program. This mechanism facilitates event-driven functionality where widgets can emit signals in response to user actions or other events. These signals can then be connected to slots, which are functions designed to respond to the specific signal.

2.2 Problem Definition

2.2.1 Initial Project Objectives

The initial project proposed had requirements for the data collection system, the GUI and the system in full. These objectives are shown in Table 2.1.

Table 2.1: The initial project objectives. These include specifications for the data acquisition system, the graphical user interface and the full system.

PicoScope	GUI	Full system
Voltage readings have less than 1% uncertainty.	Displays collected data graphically.	Data file names contain the date, time and shot counter.
Multiple scopes can be used synchronously.	Accepts user input to control the PicoScopes.	Files are saved to folders sorted by day, month and year.
Synchronization between scopes is within 0.001 ms.	Results produced within 5 seconds of data collection.	Sufficient documentation exists for all parts of the system.

2.2.2 Revised Project Objectives

After the fall term, the above objectives were assessed and modified to reflect the current trajectory of the project. The same quantitative criteria were kept for the collection system and full system but it was determined that synchronization would not be tested based on the amount of time and resources available. For the GUI, all the requirements listed above were kept and additional specifications were set out as seen in Table 2.2.

2.2.3 Project Constraints

The lab's two PicoScope-4227s are the only equipment required for this project. Although this model is no longer available, any 4000s series can interface with the software that has been developed and satisfy the hardware requirements for this project. If the lab were to purchase more devices to increase the number of sampling channels, it would cost approximately $1,455 per PicoScope, or $730 per new channel [4]. While there are no environmental

Table 2.2: Revised project objectives for the graphical user interface.

Revised GUI objectives
User inputs are applied to the PicoScope sampling code.
User can specify the voltage range, sampling rate and time intervals of the data collection.
Application settings are saved across sessions.
Interface can display up to eight plots.
Results are produced within 5 seconds of data collection.
User selects which scope and channel is displayed on each plot.
User can enter a unique name for each channel and have it displayed as the title of the plot.
Settings for each channel and trigger can be reset to default values.

constraints for the project due to its digital format, the project is helping to promote the advancement of fusion research which addresses the objective of producing clean energy. In terms of ethical considerations, code from online sources introduces constraints related to plagiarism, intellectual property, and transparency throughout the project. While utilizing open-source libraries is common in software development, all uses of such code adhered to licensing terms and have been given appropriate credit to authors. In terms of safety, the lab has high voltage and current systems, which necessitate an awareness of potential hazards when work is being done. Additionally, care must be taken to ensure that high voltages are not coupled to the data collection system. Because the system was not tested with these components, these are hazards that must be considered in the future but did not impact the work done to this point.

The primary technical obstacle was that employing the oscilloscope's functionality through custom software introduced additional complexity to the process. The hardware specifications required for the data collection also introduced technological constraints. For the Rotamak, data sampling occurs during a single discharge that lasts approximately 40 ms and the desired rate for sampling is 10 MHz per channel. This high frequency sampling rate imposes restrictions on the hardware that can be used to take measurements. The last constraint is the requirement for a user-friendly, dynamic system that can be adapted and

improved. Because the lab is in its initial stage of development, the requirements of the system have not been fully defined and will evolve as research continues. Therefore, the system must be able to be adapted by others in the future.

2.3 Results

2.3.1 Fall Term Progress

The initial work completed was determining the requirements of the system and conducting research to decide on the software and techniques that would be most suitable for the project. Python was chosen for the project as it is the most well known language among the individuals working in the lab. By using this language, the project can be used and modified by future researchers.

Data acquisition system

The functional specifications of the PicoScopes were researched to ensure they satisfied the hardware requirements of the system. As mentioned previously, the PicoScope provides different modes for sampling. Block mode was selected as it stores data in the scope's internal buffer memory, resulting in the fastest data collection rate [5]. Data will be collected for 100 ms allowing for collection before, during and after the 40 ms discharge. The desired collection rate of 10 MHz is equivalent to 10 MS/s, which is below the scopes maximum collection rate of 125 MS/s. Collecting data from both channels at this rate for the 100 ms interval results in 2 MS of data per discharge which is well below the scopes buffer size of 32 MS. Therefore, the PicoScope can collect data at the required rate with a buffer large enough to store it. This sampling configuration also allows the PicoScope to satisfy the Nyquist criterion (requiring the sampling rate to be at least twice the maximum frequency of the signal) [6]. Therefore, the scope will be able to collect data for frequencies up to 5 MHz without aliasing occurring. If higher frequency signals must be detected, the sampling rate can be increased to the PicoScopes' maximum rate. Using this rate, a signal with a maximum frequency of 62.5 MHz can be sampled for 128 ms before the buffer is full.

Finally, efforts were made to develop the custom code to interface with the PicoScope. While the process for downloading the appropriate software and developing a working script was

extremely time consuming, the decision to develop code instead of using the proprietary application allowed the system to be customized to the requirements of the lab. Documentation for completing this set-up was recorded and a waveform captured by the PicoScope could be displayed on-screen using a simple python script.

Graphical User Interface

The only work completed during the fall term on the GUI was selecting the packages and software for the GUI. Choosing PyQt5 as the main library for the design satisfies the objectives of the GUI as it offers a wide range of customizable widgets and extensive documentation as mentioned previously. PyQt5 also integrates easily with other python libraries, enabling it to interface with other systems.

2.3.2 Winter Term Progress

Data Collection System

During the winter term, the PicoScope sampling code was vastly improved. The initial sampling code was based on the proprietary scripts which have the PicoScope parameters hard-coded. All fixed values were replaced with dynamic variables which was essential for the integration of the GUI and the sampling code as it allows the user's inputs on the UI to be assigned to the sampling parameters. In addition, the entire code was partitioned into methods that were added to a custom PicoScope class, called "Pscope". This shift from functional to object-oriented programming simplified the code to run data collection and allows for multiple scopes to be used at once. Instead of having to explicitly declare and store parameters for each scope, a "PScope" object is instantiated for each scope. These "PScope" objects have attributes for all the parameters needed to run the sampling code. The class diagram for the PScope class can be seen in Figure 2.1.

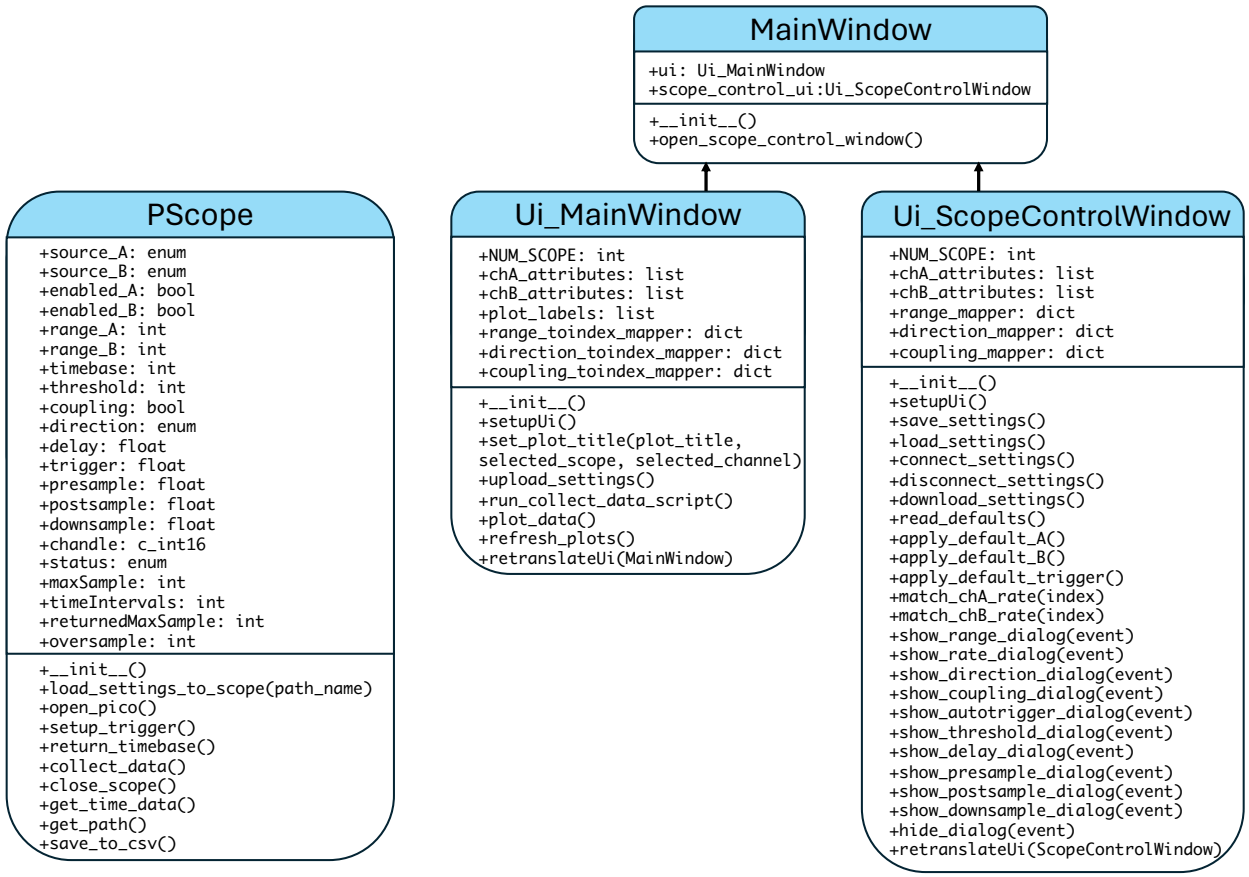

Figure 2.1: Class diagrams for the data collection system and graphical user interface. The class PScope has attributes for all the collection parameters of the data collection script. Getter and setter methods exist for all attributes in this class, but have been omitted for the sake of diagram clarity. Ui_MainWindow and Ui_ScopeControlWindow each define the structure and functions of a window in the interface. Both these classes are children of the MainWindow class.

Graphical User Interface

A significant amount of time during the winter semester was devoted to building the GUI. The QtDesigner desktop application was used to create the preliminary designs for each window. This process included deciding on all elements that would be present in the interfaces and setting the layouts. Configuring the layouts correctly ensures that the widgets in the window are appropriately dynamically scaled and re-arranged as the size of the window is changed by the user. Ensuring all components were present and the layouts were configured correctly was important because once the code has been modified, QtDesigner can no longer

be used. To ensure the GUI would satisfy all functional requirements, multiple iterations of the windows were made before exporting the UI files to python scripts.

The exported scripts have arbitrary names for all the layouts and widgets assigned by Qt-Designer such as pushButton_1, pushButton_2, etc. Because the interface has dozens of components, detailed names were assigned to each element and recorded with labelled diagrams. These diagrams are in Appendix E to assist future students who want to modify the GUI.

Next, the functionality of the GUI was implemented. This included adding signals and slots to all widgets that the user would interact with, adding pre-sets to droboxes, configuring plots and creating a system to manage all the settings within the interface. This part of the design included the most amount of iteration, with each new feature requiring multiple rounds of testing and modifications to the program before the desired functionality was achieved.

Integration

The last part of the project was the integration of the two systems together. A local directory was created to contains all the required software and files. The GUI was modified so all parameters entered by the user are saved to specific files in this directory. These files are read by the PicoScope sampling script and the parameters are set as the attributes of each scope object to control data collection. The PicoScope scripts saves the data it collects in another set of files, allowing the GUI to access and plot the data. The transfer of data is done automatically, without the user having to handle the files or directories. The diagram in Figure 2.2 highlights the communication and transfer of data between the GUI and PicoScope sampling script that are critical to the functioning of the overall system.

At the end of the project, the entire system was integrated into the lab. The system was then tested and the results are presented in Section 2.5.

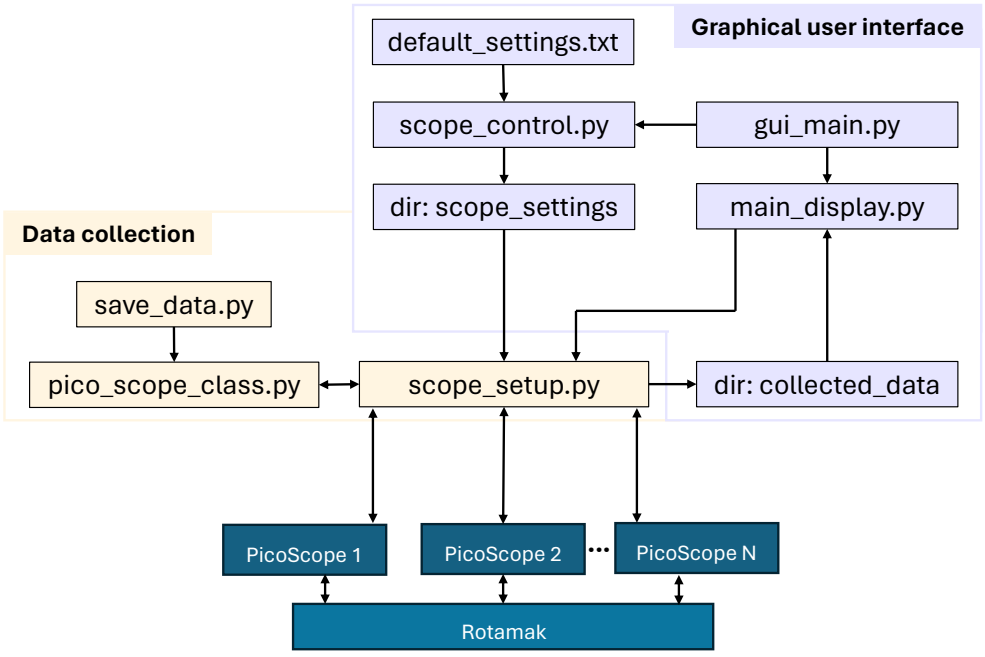

Figure 2.2: An overview of the data collection and graphical user interface for collecting data from the Rotamak. The Rotamak is connected to n PicoScopes, which are all connected to the overall system through the scope_setup.py script. The light purple boxes show the scripts and directories dedicated to the graphical user interface. The pale orange boxes show the scripts used to interface with the PicoScopes. The directories "scope_settings" and "collected_data" are included within the purple box as they are within the folder containing all graphical user interface source code. However, the data collection system also accesses both these directories.

2.4 Final Design

2.4.1 PicoScope class

As described in Section 2.3.2, a class was created for the PicoScope script to instantiate each device as a PScope object with its collection parameters stored as attributes. Custom methods in the class define these parameters and call various API functions to control the scope.

PScope class methods

The class provides a set of setter methods (e.g., `set_collect`, `set_voltage_range`) and getter methods (e.g., `get_collect`, `get_voltage_range`) to set and retrieve the various attributes of the scope objects based on the loaded settings.

The `load_settings_to_scope` method reads settings from a CSV file and applies these settings to configure the oscilloscope's channels and trigger. The `open_pico` and `setup_trigger` methods initializes the PicoScope device, sets up the channels (A and B) with their respective settings, configures the trigger settings and checks the status to ensure the device is ready for data collection.

Data collection is handled by the `collect_data` method, which sets up the device for block mode data capture, checks for data collection completion, and retrieves the collected data into buffers. This method also converts the analog signal to millivolts using the adc2mV function provided by the picosdk library.

The last part of the class defines two methods for saving the data collected by the PicoScope. The first, `get_path`, returns a path name for a folder on the desktop of the computer that specifies the year, month and date. The method checks if such a folder already exists, and creates one if it does not. The method `save_to_csv` saves the data to the path returned by `get_path` to a file for the specified scope, specifying the time, date and scope number in the file name. The method also splits the data by channel and saves it to files in the local

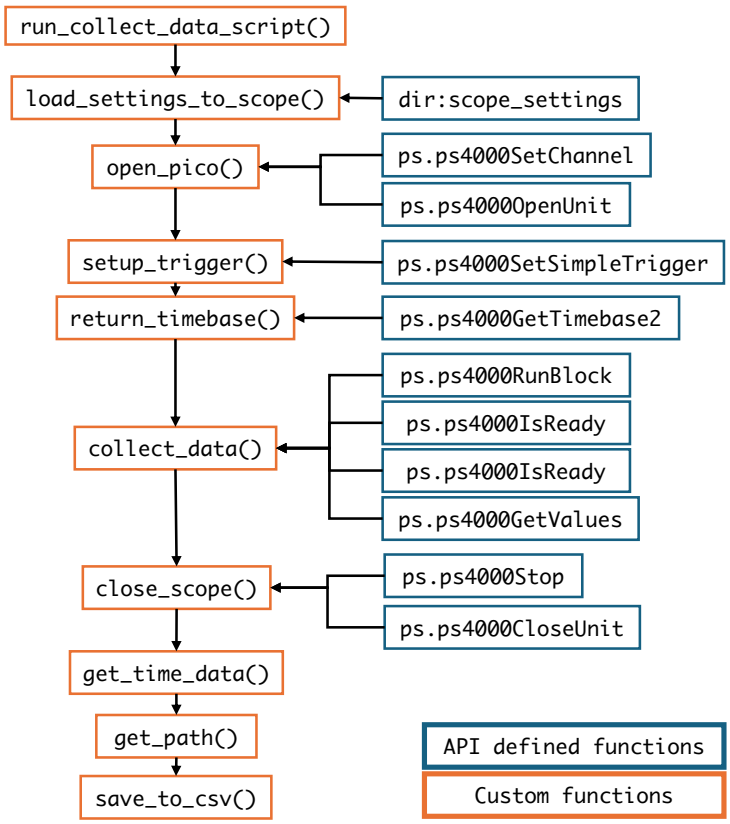

Figure 2.3: The general sequence of function calls for a script used to interface with a PicoScope. The custom functions are all members functions of the PScope class and are distinguished using orange boxes. The blue boxes are API defined functions. Each custom function utilizes one or more API functions.

directory. These are the files that are used by the GUI to display the collected data.

2.4.2 PicoScope scripts

A script for both one and two PicoScopes have been created to test the control of the scopes using the functions defined in the PScope class. The sequence of function calls for both scripts are the same and are detailed in Figure 2.3. Once more Picoscopes are purchased, the code can be expanded to follow the same process for multiple scopes by replicating this sequence. More details about increasing the control of the data collection through triggering, down sampling etc., are outlined in Appendix C.

2.4.3 Graphical User Interface

Managing multiple application windows

The code `gui_main.py` defines a superclass to set up a main application window with functionality to open a secondary window for scope control. The script imports the two modules `scope_control` and `main_display`, which contain the classes for the main window (`Ui_MainWindow`) and the scope control window (`Ui_ScopeControlWindow`) shown in Figure 2.1. Importing these classes allows the UI from the `Ui_MainWindow` class to be initialized, and a button configured that when triggered instantiates and displays the scope control window using the UI defined in the `Ui_ScopeControlWindow` class.

The decision to have one superclass that instantiates the two windows of the application when run was made for two main reasons: data storage and scalability. Having both windows be subclasses of the superclass allows them to share settings using the PyQt class "QSettings". This class allows attributes of an interface (such as combo box or checkbox selections) to be stored across sessions and between windows. Using this class avoids the need to use external files such as text files or json files to store application settings. The second reason having a superclass for both windows is beneficial is because it allows more windows to be added seamlessly. Once a user designs a new window, it can be imported as a module and have an instance of its class be instantiated using any functionality the programmer chooses. In fact, an extra blank button has been included in the current main window so that future users can use it to open another window. This will avoid them having to add a button manually and configure it within the existing layout.

Main display

The script `main_display.py` defines the class for the main display window of the GUI shown in Figure 2.4. Within the class, the `setupUi` method establishes the main window's structure. The majority of this method was automatically generated and encapsulates the design

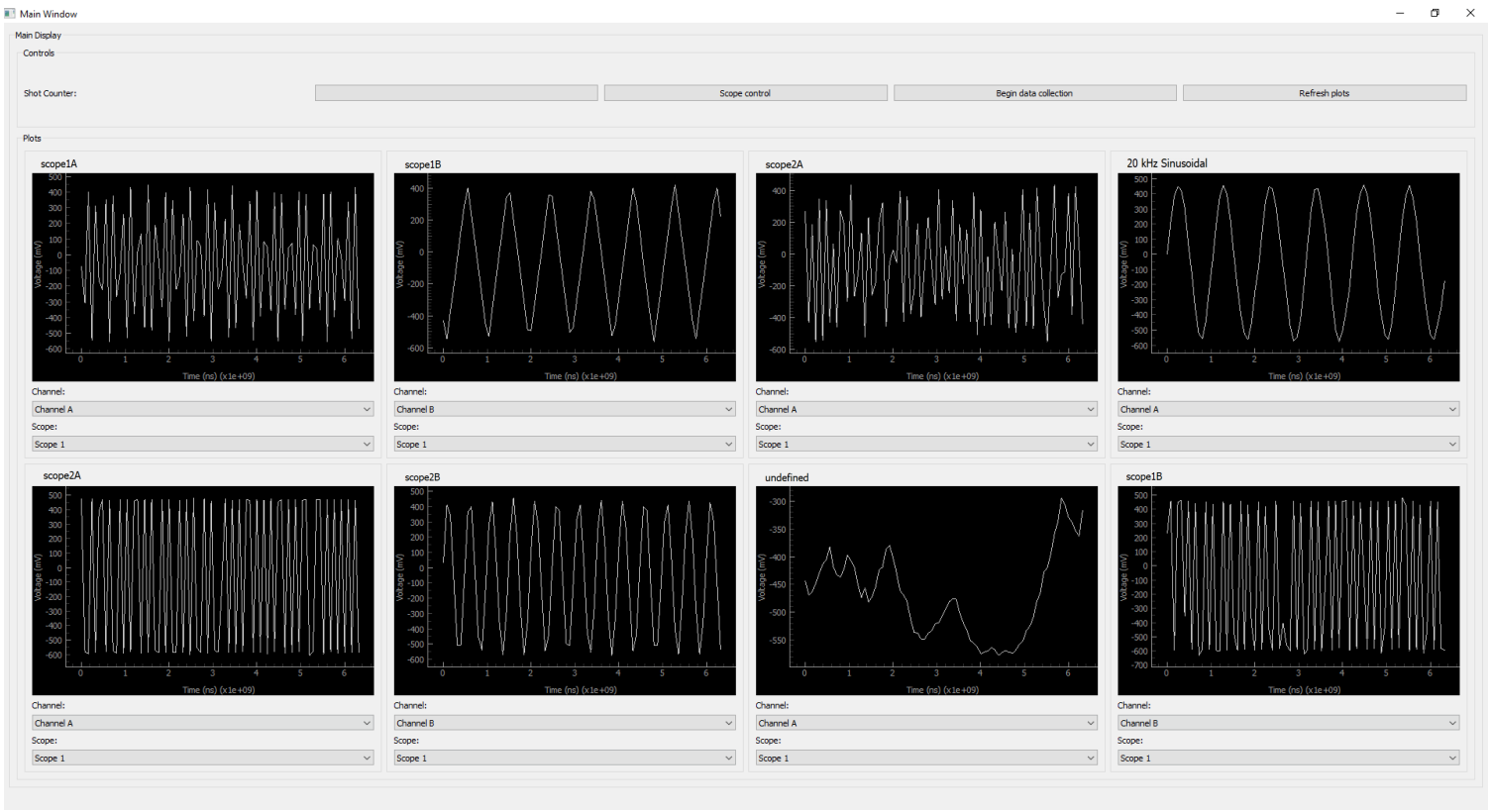

Figure 2.4: The main window of the graphical user interface for the Rotamak data collection system. The window consists of eight plots that display data for a specific scope and channel based on user selection using the "Channel" and "Scope" dropboxes below each plot. The "Scope control" button opens a second window. The "Begin data collection" button starts a data collection script that interfaces with a PicoScope. The "Refresh plots" button updates all the plots. The shot counter has not yet been fully implemented. An extra blank button is included to allow for easy additions to the window that require a button.

created using QtDesigner for this window. The plots were added manually using a plot widget from the pyqtgraph library to ensure they can dynamically display data and plot titles corresponding to the user's channel and scope selection.

The script also introduces an `upload_settings` method that reads in configuration settings from CSV files and stores these settings using the QSettings class. The `set_plot_title` function uses these settings to update the title of each plot based on the selected scope and channel.

The `plot_data` method is responsible for the core functionality of the main window display.

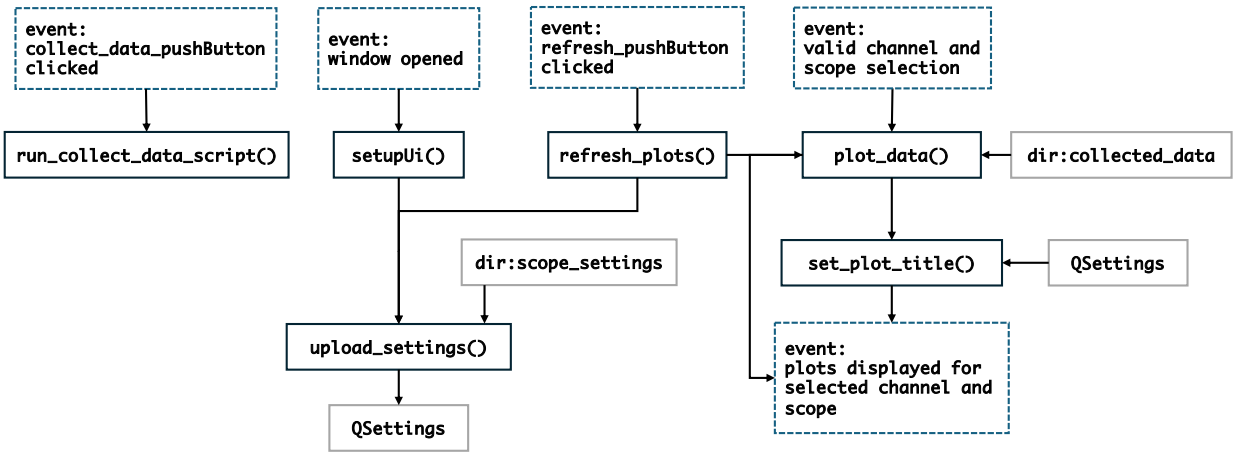

Figure 2.5: Flowchart for the methods included in the main_display script. These methods control the functionality of the application's main window which displays data graphically and runs a data collection script. The four events shown in the first row include button clicks and window opening and trigger various methods which trigger other methods, access files in directories and retrieve and update the application's settings in QSettings.

It reacts to user interactions with the scope and channel combo boxes. Upon a valid selection, this function reads in the corresponding data from a CSV file and plots it on the designated plot widget. In case of invalid selections or missing data files, it ensures that the plot is cleared and displays a message to the user.

Lastly, the PicoScope collection script is run when a user presses the "Begin data collection" button. This button triggers the `run_collect_data_script` method, which starts the collection script. Once the PicoScope script has been run and data has been collected, the plots must be manually refreshed using the "Refresh plots" button. This button signals the `refresh_plots` method which loops through all eight plots, individually calling `plot_data` for them.

The flow and functionality of the main window, specifically how various user interactions trigger different functions and processes within the software, is shown in Figure 2.5.

Scope control window

The `scope_control.py` script defines a class for the secondary window shown in Figure 2.6 that is used to adjust the PicoScope settings. Again, the `setupUi` method controls the layout

and components of the window, organizing it into a central widget, containing group boxes for selecting the scope, controlling channels, and setting trigger parameters. Each channel's group box has widgets allowing users to set parameters including the channel name, whether data should be collected from the channel, voltage range and sampling rate. The trigger control group box allows the user to set parameters such as direction of the trigger, threshold and number of pre-trigger and post-trigger samples. Two additional parameters "Extra 1" and "Extra 2" were created and left as part of the design to support the addition of features in the future. Adding widgets through the backend is time consuming, so this will facilitate quicker upgrades to the system if desired, and does not affect the current functionality.

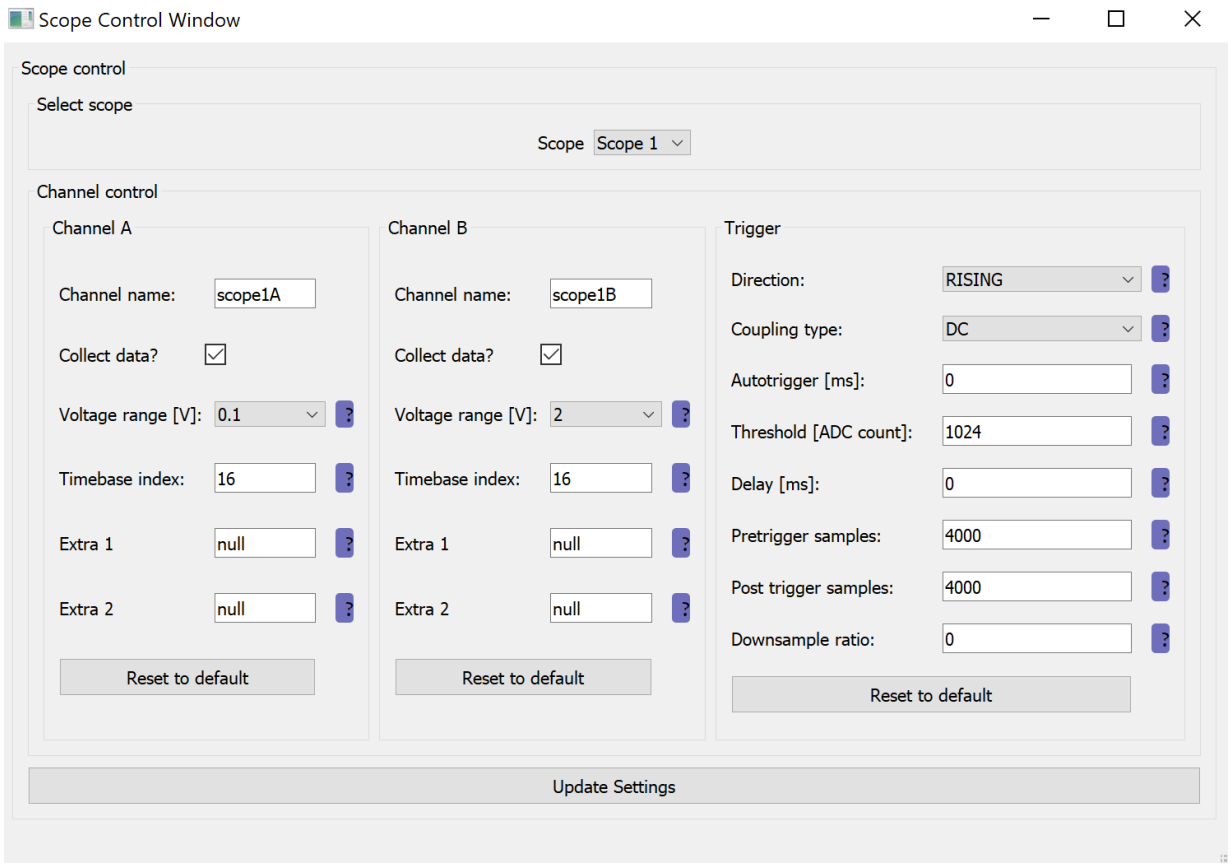

Figure 2.6: The secondary window for the graphical user interface used to update parameters that control the PicoScope data collection. The user has the ability to switch between scopes using the "Scope" dropdown. Three different control boxes contain parameters for channel A, channel B and the trigger. Each parameter is either a dropbox, checkbox or text edit line. The applications settings are automatically updated when a user changes a setting. To save the settings to an external file, the "Update Settings" button must be clicked.

Tooltips were implemented for various settings, providing users with quick guidance on what each setting controls. These tooltips are activated when the user hovers their mouse or trackpad pointer over certain UI elements, such as range or rate selection, using events like `show_range_dialog` as seen in Figure 2.7.

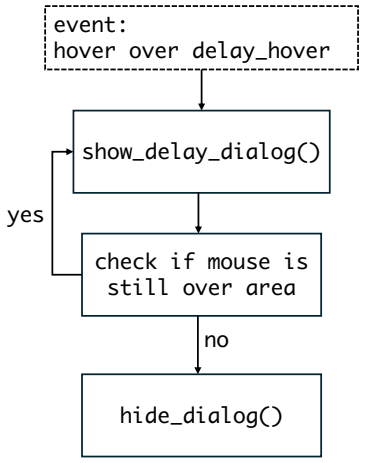

Figure 2.7: The flowchart for a tooltip that displays dialog to a user when their mouse hovers over a specific application component. In this case, when the user hovers their mouse over the `delay_hover` component, the `show_delay_dialog()` method is called, which displays the dialog until the mouse is removed from the area.

A significant part of the script's functionality is devoted to loading and saving settings. The `save_settings` method is invoked whenever a change is made to the UI, updating the stored settings accordingly. This allows the user to switch between scopes and close the application without losing the preferences they have entered. The `upload_settings` method retrieves these preferences for the currently selected scope and applies them to the UI. This method is called when the window is initially opened and each time the user switches the scope selection. To ensure settings for the previously selected scope are not saved when the scope selection changes, all the parameter widgets are disconnected from the `save_settings` method. Once the proper settings have been uploaded, the widgets are reconnected.
In addition to saving the data within the UI using QSettings, the script includes a feature to export settings to a CSV file for each scope. This allows settings to be stored and used by the PicoScope script. The `download_settings` method iterates over all scopes, compiles

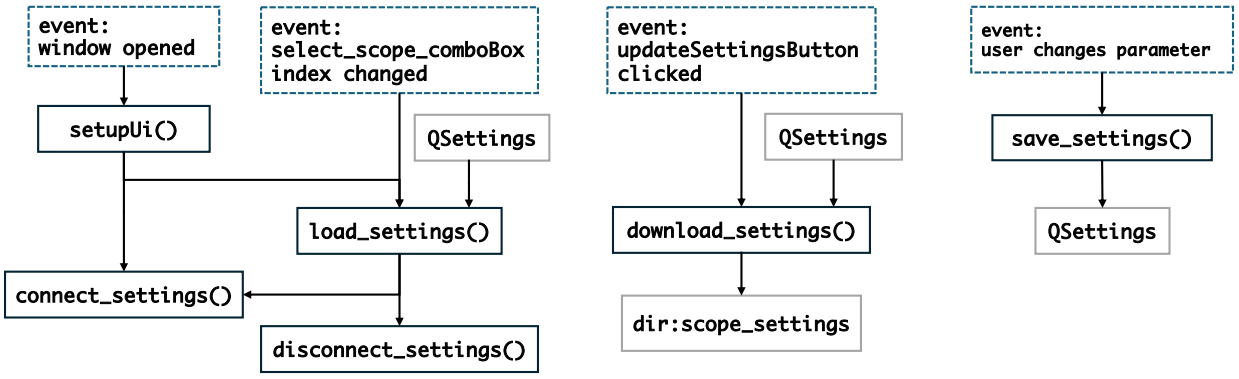

Figure 2.8: A flowchart for the methods responsible for handling the settings of the scope control window of the graphical user interface. The window being opened and the scope selection changing results in the settings being loaded to the application from QSettings using `load_settings`. Conversely, when a user changes a parameter, the new setting is saved to QSettings. When the "Update settings" button is clicked, the current application settings are saved to an external file in the `scope_settings` directory.

their settings into a structured format, and writes them to respective CSV files. This method is called only when the user presses the "Update Settings" button.

The event diagrams for the above methods responsible for handling settings in the scope control window class are shown in Figure 2.8.

Lastly, the script has buttons to reset settings to their defaults for both channels and the trigger configuration. Using methods `apply_default_A`, `apply_default_B`, and `apply_default_trigger`, values which are defined in an external text file are loaded to the UI when a user presses the "Reset to Default" buttons. These buttons also trigger the `save_settings method`, ensuring the default settings are automatically applied to the QSettings. How these default methods respond to user interactions is shown in Figure 2.9.

2.4.4 Data management

As mentioned above, the QSettings class is used to store application settings between sessions and allow multiple windows to access them. However, the PicoScope script does not have access to these settings. To allow the GUI to transfer parameters to the Picoscope class and for the GUI to plot the data collected by the PicoScope, two directories in the local workspace

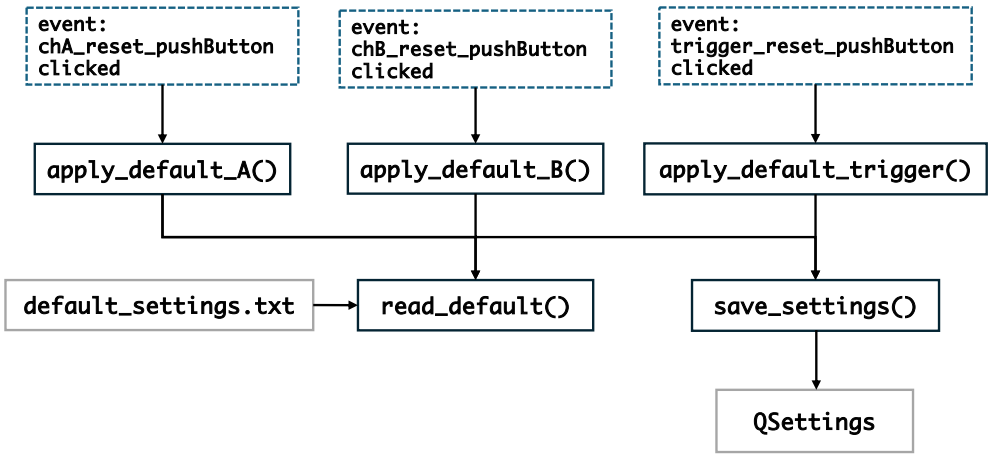

Figure 2.9: The flowchart for the methods responsible for applying default settings to the application. This sequence of function calls is triggered when any of the three "Reset to default" button are clicked. An external text file with pre-defined default settings is read and the settings are saved.

were made. The first is `scope_settings`. This directory contains a text file for each scope with all it's corresponding parameters. This file is updated when a user presses the "Update scope settings" button and is read when the PicoScope `load_settings_to_scope` function is called. The second directory, `collected_data` is used to store the data that the PicoScope collects. It is accessed by the GUI to plot the collection results. This flow of information, along with the communication between the various scripts involved in the system were shown in Figure 2.2.

The combo box widgets in PyQt store the current index that is selected, rather than the text or value of the option that has been selected. The PicoScope code also uses indexes for specific scope parameters. To limit the chance of error when using indexes, the combo boxes were all configured so that their indexes matched those used by the PicoScope. While these indexes are essential for the PicoScope and PyQt backend of the code, displaying them to users would not be informative. Therefore, custom mapping functions are used to convert the indexes to readable values displayed on the GUI and saved to files users may read or edit. An example of this process is shown in Figure 2.10, where a voltage value is mapped multiple times throughout the program.

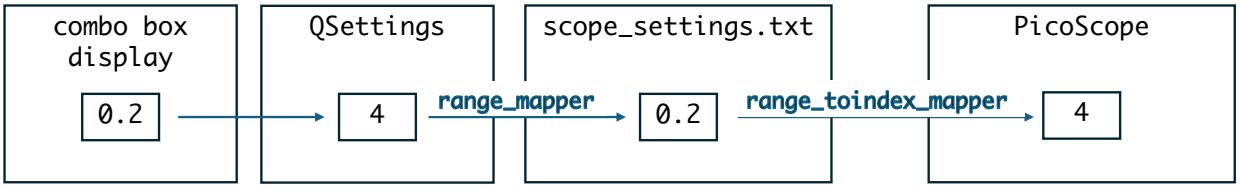

Figure 2.10: The conversion of a voltage range value using two mapping functions. When displayed by the combo box, the value is 0.2. The software automatically stores this as the index 4 in the application's settings. To display to users in a text file, `range_mapper` is used to convert the index back to it's decimal value. To use as an input parameter for the PicoScope, the value is mapped back to its index using `range_toindex_mapper`.

An example of the indexes and displayed values for voltage range are shown below in Table 2.3. The remaining parameters used and their corresponding indexes can be found in Appendix C.

Key	Enumerated member	Voltage range	Displayed value
0	PS4000_10MV	±10 mV	0.01
1	PS4000_20MV	±20 mV	0.02
2	PS4000_50MV	±50 mV	0.05
3	PS4000_100MV	±100 mV	0.1
4	PS4000_200MV	±200 mV	0.2
5	PS4000_500MV	±500 mV	0.5
6	PS4000_1V	±1 V	1

Table 2.3: The values used by various parts of the program to store voltage range. The comboboxes in the graphical user interface script and the PicoScope requires the index or 'key' corresponding to the voltage range. For user accessibility, the displayed value is the decimal representation of the index.

2.4.5 Using the system

To use the system, minimal set up from the user is required once all the software has been downloaded. For full instructions on how to download the software, please refer to Appendix D. With the software configured correctly, the user does not need to edit the code; all changes

and control of the system can be done through the GUI. The user must simply run the main GUI script, which will open the main window. From there, they can choose which plots they want displayed, run the PicoScope data collection script and access the scope control window to change the scope parameters.

Run time of the data collection is based on a range of different factors based on the trigger configuration of the scope. If the system is running with no or a single external trigger, the run time is found by multiplying the number of pre and post trigger samples by the specified collection rate. If a trigger is being used to start and end data collection, then the run time will depend on the number of pre and post trigger samples as well as the number of samples between the triggers.

2.5 Design Validation

2.5.1 Single PicoScope data collection

To test the synchronicity between channels of the PicoScopes, a single output from a function generator was connected to both channels of the scope. Figure 2.11 shows an example of the data collected. The data collection rate was set to 10 MHz for a duration of 40 ms to imitate the conditions of collection for a rotamak discharge. The satisfy the Nyquist sampling criterion, a sinusoidal signal with a frequency of 2MHz was used.

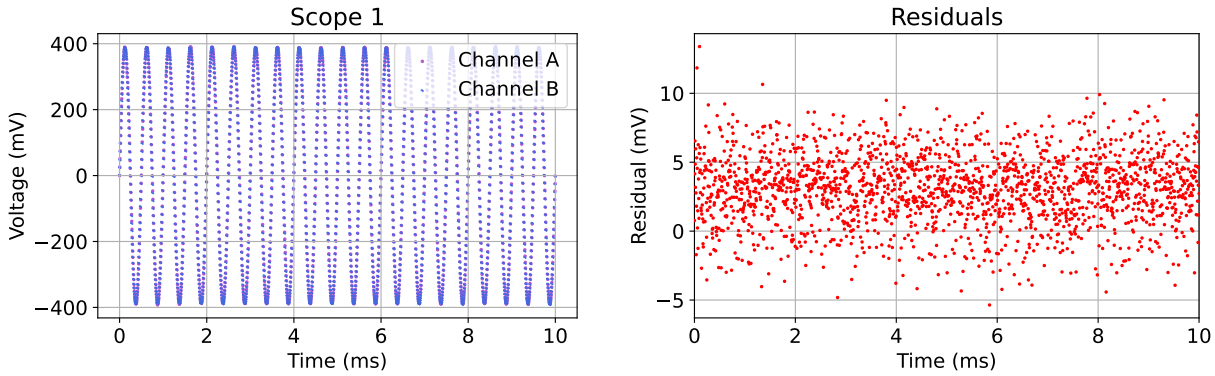

Figure 2.11: The data collected by channel A and B using a PicoScope-4227 oscilloscope. A collection rate of 10MHz was used to sample a 2MHz sinusoidal wave from a function generator. The data displayed has been down-sampled by a factor of 50 to allow for individual points to be seen. The full data collection was over 40 ms, but only the first 10 ms are displayed. The residuals plot shows the difference between the readings from the two channels.

The residuals shown in Figure 2.11 demonstrates the discrepancy between the two channels. The bias towards positive residuals indicates a systematic error in the collection system which could be attributed to flaws in the software or inconsistencies in the hardware.

In addition to testing the reliability of the channel readings, taking this sample data helped validate the functionality of the system. All the data was collected using only the GUI on the lab computer and was successfully displayed on the plots on the GUI.

2.5.2 Two PicoScope data collection

Data collection using two PicoScopes was also carried out to determine the ability of the system to automatically organize files and interface with multiple scopes. As seen in Figure 2.12, the files are sorted by date, time of collection and scope number.

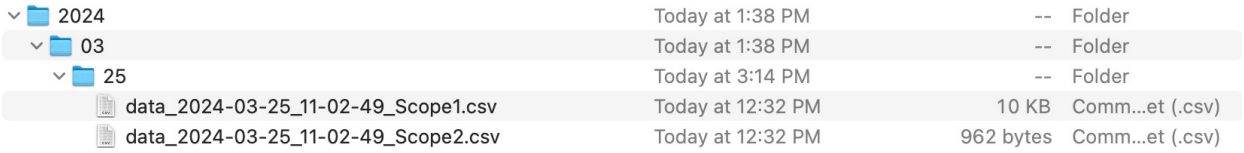

Figure 2.12: The file organization scheme for the PicoScope data collection system. Files are labelled using the date, time and scope number. The files are organized by year, month and day. All the sorting is done automatically, with folders being created if they do not already exist.

This data collection test also proved the ability of the system to take inputs from the user and apply them to the collection script. Voltage range, sampling rate and number of pre and post trigger samples were specified using the GUI. The data was not used for analysis because of the lack of an external trigger to synchronize the scopes. Regardless, this test demonstrates the ability of the system to interface with multiple scopes at once.

2.5.3 Graphical user interface

To ensure a robust and reliable interface, testing was conducted by experimenting with various settings and actively engaging with the interface in real-world scenarios. Many iterations of the user interface were completed, each one addressing flaws found through evaluation of the interface across different use cases. By adjusting settings and navigating through the interface's features, scenarios where the interface failed to perform as expected were addressed.

The graphical user interface satisfies all functional requirements for the system. It can successfully display eight plots instantaneously based on the users selection of scope and channel. In addition, it displays the plot title as the name that has been entered for that channel. Default values can be applied to all parameters and user inputs are automatically

saved. Settings are stored between sessions and reloaded to the application when a window is opened.

2.5.4 Overall system

The system was successfully integrated into the lab and works with all the hardware present. The application is user-friendly and all functionality is achieved using only the interface-making it accessible to all users. No programming, manually managing files or running scripts is required to use the system. In response to different user actions, print statements are used to update the user on the status of the system. To further facilitate effortless use of the system, instructions on how to download all the necessary software are detailed in Appendix D. In addition, all code contains comments to provide future programmers with sufficient information to edit and develop the code. All parts of the system were designed to support upgrades or additions to the system.

2.6 Limitations

The GUI and PicoScope data collection system, while functional, present a series of limitations that users should be aware of before the system is used in the lab. Firstly, the systematic error between the channel voltage readings must be addressed before the use of the PicoScope collection script is deemed reliable. In addition, the synchronicity of multiple PicoScopes has not been evaluated, which will be crucial for experiments requiring multiple scopes. Currently, the voltage and time data for each channel is saved to its own file and only channels from the same scope share a common set of time data. The ability to process data from multiple scopes using common time data would first require the synchronicity between multiple scopes to be established. These functional requirements were not achieved or tested due to time constraints.

Another limitation of the system is it's lack of error handling. Because the values entered in the GUI are used to control the PicoScope data collection, care must be taken to ensure that the PicoScope will be able to carry out the requested task. Failure to do so will not result in any harm to the GUI, but will result in no data collection. To ensure this does not occur, the user must be mindful of the amount of data the scope will collect. With the buffer size of 32 MS, the user must perform preliminary calculations to ensure the combination of sampling rate and number of samples does not exceed this size. In addition, the PicoScope script will fail if the voltage it reads is outside of the voltage range specified. Requiring the user to perform these preliminary calculations reduces the user-friendly nature of the system and decreases its robustness in handling all use cases. This lack of resilience affects the user experience and the reliability of the system as a whole.

The final major limitation of the system is the ability of the graphical user interface to display data as soon as it has been collected. This addition was not made due to a lack of time, but would be supported by the current state of the software.

Chapter 3

3.1 Conclusion

3.1.1 Summary of work and goals achieved

Overall, the work done has provided a foundation for a system that can be used and adapted in the lab. All of the objectives regarding the graphical user interface were achieved. It has the desired dual functionality of displaying data collected by the scopes and accepting user inputs to control the collection process. Because of the robust backend, only interactions with the GUI are required to use the system, making the system accessible for all users. Because the lab is in its early stages and timing limitations surrounding this project, designing the data collection system was met with less concrete success. Despite the failures to carry out extensive testing, the system was designed dynamically to allow changes and additions to be made in the future to tune the system to the lab's requirements. The provided documentation for the project will allow users to test the current software design and build upon it.

3.1.2 Future work

Data collection system

While it has been shown that the system supports multiple PicoScope devices, the synchronicity between them must be tested. To do so, an external trigger can be coupled to all scopes being tested. From there, voltage measurements can be taken and analyzed to determine the synchronicity of the devices. If the scopes can sample synchronously, the only

additional code that will need to be written is to dynamically determine how many scopes have been selected to collect data, and running the collections script for only those scopes. If it is found that the scopes cannot sample synchronously using the current data collection method, threading or the use of parallel programming may be necessary.

Graphical user interface

Multiple additions can be made to the GUI. Firstly, the script can be modified so that the plots automatically refresh when the PicoScope is done collecting data. This functionality can be used to test the response time of the plotting system. As well, a shot counter can be incremented each time a Rotamak discharge is run. There is already a widget for this in the main window, it just must be modified to display the shot counter value.
Another addition that can be made is the use of threading in the GUI. By incorporating threading, the GUI application can significantly enhance its performance and responsiveness. By delegating intensive tasks to separate threads, the main GUI thread can continue to process user inputs, update displays, and respond to events while the underlying script handles data collection and processing in the background. PyQt has a "QThreads" module that would help with this addition.

Error handling

A final project that can be undertaken for both systems is implementing error handling. This process would require creating additional code to calculate different parameters, check their validity and display warning messages if the input is not valid. The error handling mechanisms would be used to avoid invalid entries and create a more reliable data collection system.

Word count

5329 words

Bibliography

[1] Pico Technology, *PicoScope 4227 High-Resolution Oscilloscope with Arbitrary Waveform Generator*. Pico Technology, James House, Colmworth Business Park, St Neots, Cambridgeshire, PE19 8YP, UK, 2010. Specifications and performance of PicoScope 4227.

[2] Pico Technology Ltd., "Advanced digital triggers." Pico Technology Library, 2024. Accessed: 2024-01-18.

[3] picotech, "picosdk-python-wrappers." `https://github.com/picotech/picosdk-python-wrappers`, 2024.

[4] "Picoscope® 4000 series high resolution oscilloscopes." Pico Technology Ltd., 2024. Accessed: 2024.

[5] Pico Technology Ltd., *PicoScope 4000 Series Programmer's Guide*. Pico Technology Ltd., James House, Colmworth Business Park, St Neots, Cambridgeshire, PE19 8YP, UK, 8 ed., 2014. A guide for programming PicoScope 4000 Series Oscilloscopes.

[6] D. C. E. Robinson, E. Robinson, and D. Clark, "Sampling and the nyquist frequency," *The Leading Edge*, vol. 10, no. 3, pp. 51–53, 1991.

Appendix A

Statement of Work and Contributions

No work was completed for this project prior to September 1st. The work completed in the fall term included preliminary research, selecting the software for the project and writing a basic data collection script to interface with a PicoScope. In the winter term, the entire GUI was developed and the PicoScope script was improved significantly. Both systems were also integrated together and implemented in the lab where testing took place.

While no work was done collaboratively, support was provided by Daniel Solano, Nathan Eddy and Jérémy Talbot-Pâquet. Daniel provided overall guidance to help work toward large milestones as well as encouragement. Nathan helped with the use of QtDesigner, specifically how to configure the layouts. Jérémy assisted in de-bugging the initial PicoScope script to get the proprietary software to run on the lab computer.

Appendix B

Complete Work

All source code written by Jane Cohen can be found at `https://github.com/janecohen/ENPH455_Thesis/tree/main`.

Appendix C

PicoScope data collection

C.1 PicoScope Control

Detailed instruction for interfacing with a PicoScope can be found on the PicoScope Ltd. website [3]. For this project, all the API functions and parameters used have been outlined below. In addition, details on advanced control of the scopes is provided, although it was not implemented.

C.1.1 Advanced PicoScope Control

Sampling modes

Sampling modes determine how and when the oscilloscope collects data.

Block Mode: Captures a single block of data into the oscilloscope's memory. Suitable for short bursts of high-speed signals. The oscilloscope stops capturing once the memory is full.

Streaming Mode: Data is continuously streamed to the computer, ideal for long-duration captures at lower sampling rates. This mode allows for real-time data analysis and recording.

C.1.2 Triggering modes

Simple Edge Triggering: The most basic form, starting a capture when the signal crosses a defined voltage level. The direction of the crossing (rising or falling edge) can be specified.

Advanced Triggering: Offers more sophisticated conditions, such as pulse width, window, and dropout triggers, allowing for precise control over when data capture begins.

Logic Triggering: Combines multiple input conditions, enabling complex trigger setups.

C.1.3 Oversampling

Oversampling involves taking multiple readings at each sample point and averaging them to increase the signal resolution. This technique is particularly useful for improving signal quality when the sampling rate is lower than the oscilloscope's maximum. It's important to note that oversampling can increase the time to acquire data and is mainly applicable in block mode where the highest resolution is required.

C.1.4 Down sampling

Aggregation: Combining adjacent samples to reduce data size while retaining important signal characteristics. This method is often used in streaming mode to manage the large volume of data.

Decimation: Simply reducing the number of samples by only keeping every nth sample. This straightforward approach is useful for reducing data rates but can miss important signal details between kept samples.

C.2 PicoScope functions

Table C.1: Summary of PicoScope API defined functions.

Function	Purpose	Arguments
`ps40000OpenUnit`	Opens a scope device.	handle
`ps40000SetChannel`	Specifies which input channel and if it is enabled.	handle, source, enabled, coupling, range
`ps40000SetSimpleTrigger`	Sets up trigger.	handle, enabled, source, threshold, direction, delay, trigger
`ps40000GetTimebase2`	Configures timebase settings based on timebase index.	handle, timebase, noSamples, timeIntervalNanoseconds, oversample, maxSamples, segmentIndex
`ps40000RunBlock`	Run block capture to collect data.	handle, presample, postsample, timebase, oversample, time_indisposed, segment_index, lpReady, pParameter
`ps40000IsReady`	Receive data from RunBlock.	handle, ready
`ps40000SetDataBuffers`	Registers buffers for receiving data.	handle, source, bufferAMax, bufferAMin, maxSamples
`ps40000GetValues`	Retrieve data for all channels.	handle, start_index, cmaxSamples, downsample_ratio, downsample_ratio_mode, segment_index, overflow
`ps40000Stop`	Cause trigger event.	handle
`ps40000CloseUnit`	Shuts down device.	handle

C.3 PicoScope parameters

Timbase The conversion between index and sampling interval.

Table C.2: PicoScope sampling interval index conversions.

Timebase	**Sampling interval [ns]**
Low (n = [0,3])	$2^n/250,000,000$
High (n = [4, $2^{30}-1$])	$(n-2)/31,250,000$

Source The 'source' argument specifies which channel to trigger on.

Table C.3: Keys for the PicoScope channel selection.

Key	**Control**
0	Channel A
1	Channel B

Enabled The 'enabled' argument specifies if the channel is disabled or not.

Table C.4: Keys for the PicoScope channel control.

Key	**Control**
0	Do not collect data
Any non-zero integer	Collect data

Voltage range The 'range' argument specifies the voltage range the PicoScope will sample over.

Threshold The 'threshold' argument specifies the ADC count at which the trigger will fire.

Direction The 'direction' argument specifies the type of trigger.

Delay The 'delay' argument specifies the amount of time between the trigger and the first sample being taken. In milliseconds.

Autotrigger The 'autroTrigger_ms' argument specifies the amount of time without a trigger being detected before sampling begins. In milliseconds.

Coupling mode

Table C.5: PicoScope voltage range selection.

Key	Enumerated member	Voltage range
0	PS4000_10MV	±10 mV
1	PS4000_20MV	±20 mV
2	PS4000_50MV	±50 mV
3	PS4000_100MV	±100 mV
4	PS4000_200MV	±200 mV
5	PS4000_500MV	±500 mV
6	PS4000_1V	±1 V
7	PS4000_2V	±2 V
8	PS4000_5V	±5 V
9	PS4000_10V	±10 V
10	PS4000_20V	±20 V
11	PS4000_50V	±50 V
12	PS4000_100V	±100 V

Table C.6: Trigger event types for PicoScope.

Enumerated member	Direction
ABOVE	Above the upper threshold
ABOVE_LOWER	Above the lower threshold
BELOW	Below the upper threshold
BELOW_LOWER	Below the lower threshold
RISING	Rising edge, using upper threshold
RISING_LOWER	Rising edge, using lower threshold
FALLING	Falling edge, using upper threshold
FALLING_LOWER	Falling edge, using lower threshold
RISING_OR_FALLING	Either edge
INSIDE	Inside window
OUTSIDE	Outside window
ENTER	Entering the window
EXIT	Leaving the window
ENTER_OR_EXIT	Either entering or leaving the window
POSITIVE_RUNT	Entering and leaving from below
NEGATIVE_RUNT	Entering and leaving from above
NONE	None

Table C.7: Input frequency modes for PicoScope.

Key	Mode
DC	PicoScope accepts all input frequencies from zero to the maximum analog bandwidth.
AC	PicoScope accepts all input frequencies from a few hertz to the maximum analog bandwidth.

Appendix D

Software download

Before using the PicoScope script or GUI program, two sets of software must be downloaded. This process has already been carried out on the lab computer but is presented here for future reference.

D.1 PicoScope collection system

Visit `https://github.com/janecohen/ENPH455_Thesis/tree/main`. Download all the files and create a folder for them. This will be the main directory for the project. In the README.md file of the GitHub, there is a link to the PicoScope GitHub. Follow this link and follow the instruction for downloading the picoSDK software. Run the picoSDK exe by double clicking it and following instructions. This will download directory "Pico Technology". You then must move this folder to your project directory. Finally, run the command "pip install ." to install all the drivers. You are now ready to use the Pico API.

Open the project directory in VS code. The only changes that will need to be made are to the main directory paths in the PicoScope sampling scripts. Change all directory names to reflect the project directory location.

The "./PicoScope" directory contains all code for interfacing with PicoScope.

D.2 PyQt5

Simply run the following commands:

pip install PyQt5

pip install pyqt5-tools

pip install pyqtgraph

pip install pandas

The "./GUI" directory contains all code for the graphical user interface.

Appendix E

Interface diagrams

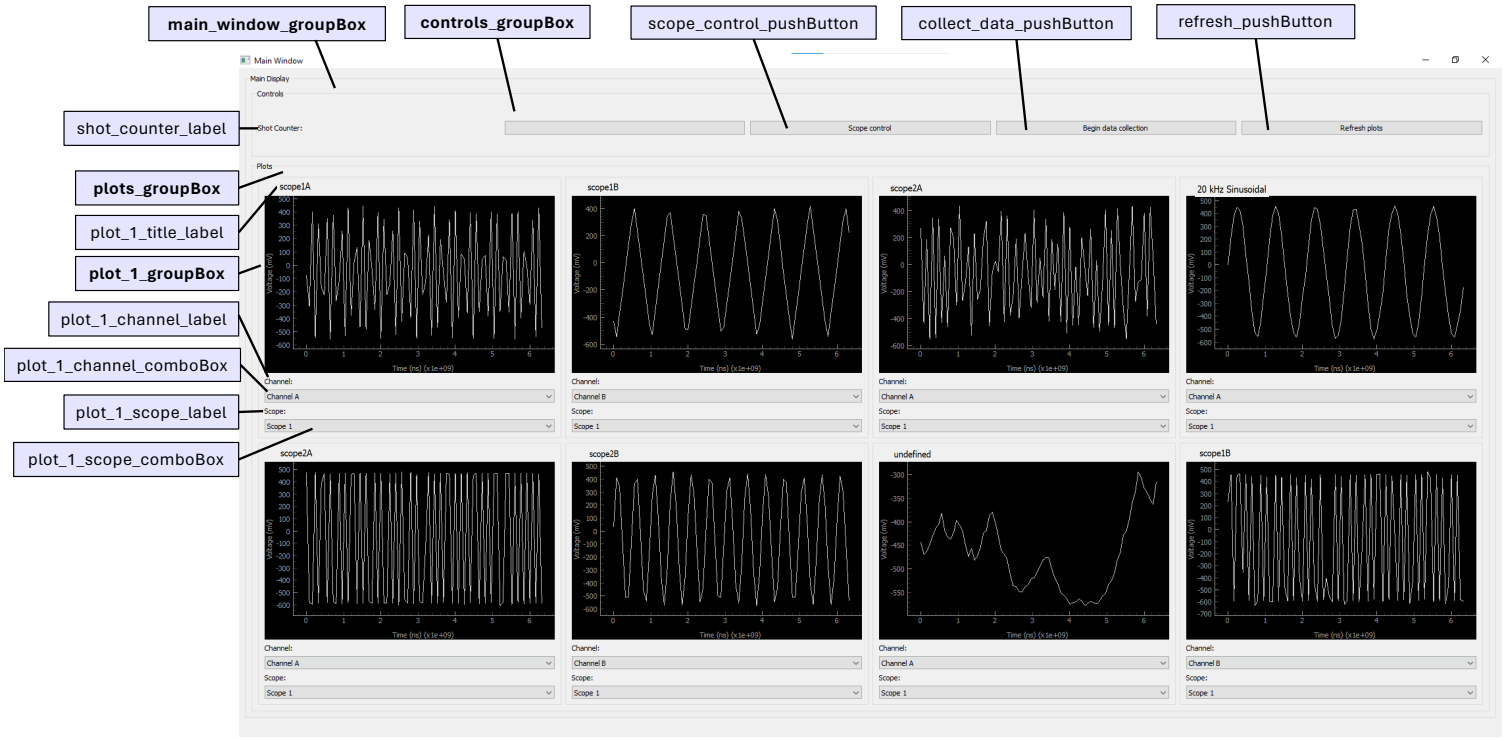

Figure E.1: The main window of the graphical user interface with all widgets labelled using their corresponding name in the source code. The window consists of eight plots that display data for a specific scope and channel based on user selection using the "Channel" and "Scope" dropboxes below each plot. The "Scope control" button open a second window. The "Begin data collection" button starts a data collection script that controls a PicoScope. The "Refresh plots" button updates all the plots. The shot counter has not yet been fully implemented. An extra blank button is included to allow for easy additions to the window that require a button such as opening a new window.

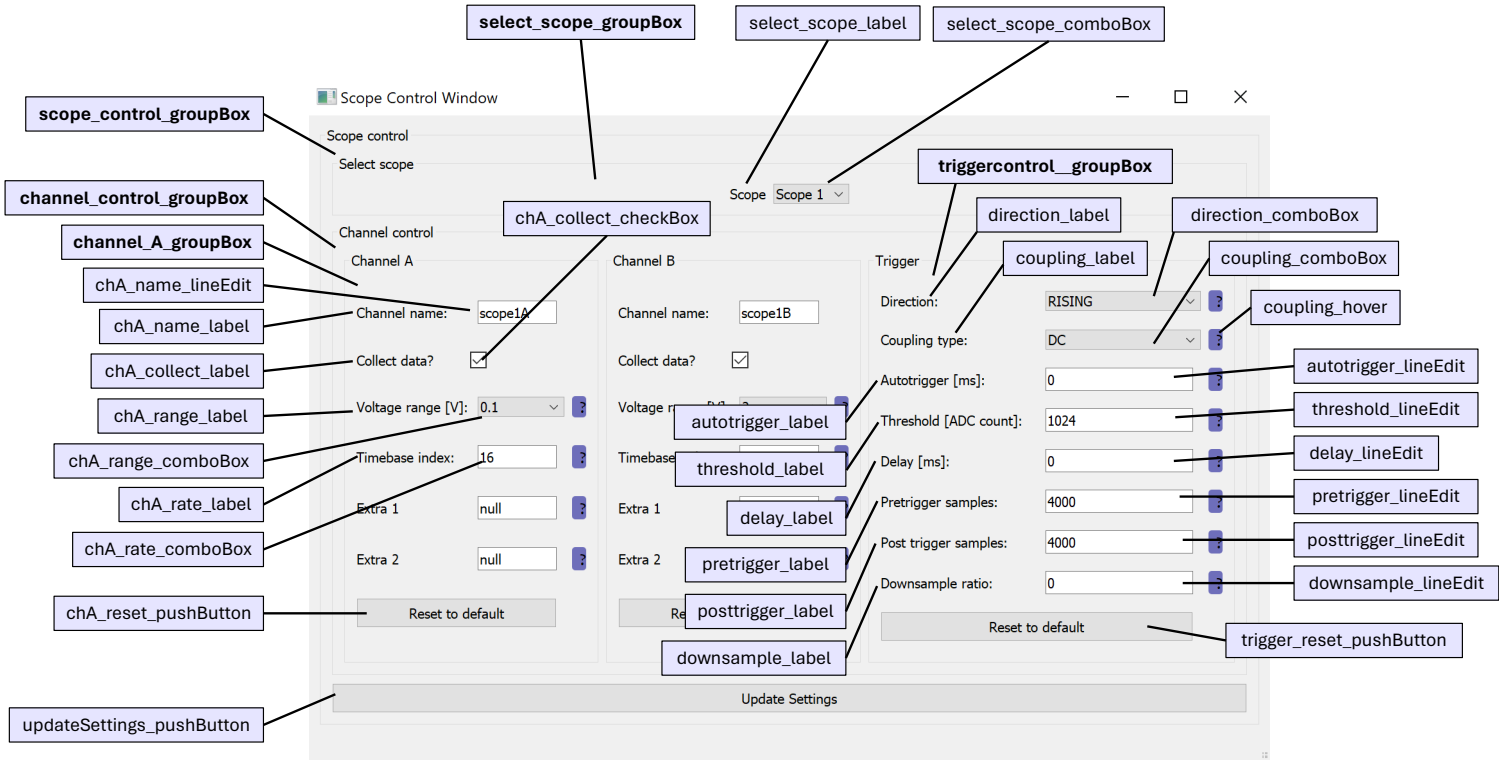

Figure E.2: The secondary window for the graphical user interface with all widgets labelled using their corresponding name in the source code. The user has the ability to switch between scopes using the "Scope" dropdown. Three different control boxes contain parameters for channel A, channel B and the trigger. Each parameter is either a dropbox, checkbox or text edit line. The applications settings are automatically updated when a user changes a setting. To save the settings to an external file, the "Update Settings" button must be clicked.