Institut für Hochfrequenztechnik

RWTH AACHEN UNIVERSITY

Essay

Development of remote control functionalities and a long-term measurement option for the handheld field strength measurement device VIAVI ONA-800

by Marvin Rosenpländer
Matr.-Nr. 3592458

Supervised by:
Prof. Dr. Hans-Joachim Krause
Thanh Tam Julian Ta, M.Sc.

Essay
Aachen,
7th January, 2025

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Development of remote control functionalities and a long-term measurement option for the handheld field strength measurement device VIAVI ONA-800

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Marvin Rosenplänter

Aachen, den Saturday 28th December, 2024

Unterschrift des Studenten

Summary

1 Abstract

The goal of this essay is the development of an application that extends and enhances the functionalities provided by the field strength measurement device VIAVI ONA-800. The device is used during the Institute of High Frequency Technology's research on electromagnetic environmental impact and offers multiple measurement modes. However, the functionalities of some could be expanded upon, which will be a central goal of the application. This essay will discuss the devices functionalities, as well as its shortcomings, in greater detail and will thereby derive requirements and constraints for the application. Whereafter, the framework choice will be discussed and the eventual choice of Qt [Qt2] will be substantiated. Specific implementation details will also be discussed in this essay, including, but not limited to, the internal representation of measurement options and specific data structures necessary to efficiently render a spectrogram for the real-time spectrum visualization. Finally, this essay will show that the application, while satisfying all requirements, outperforms the device's logging functionality, which can be used to record measurements.

Contents

2 Motivation

The VIAVI ONA-800 is a handheld field strength measurement device, which is used during research at the Institute of High Frequency Technology (IHF). It provides multiple measurement modes, of which the Real-time Spectrum Analysis, Spectrum Analysis and 5G NR Beam Analysis are particularly relevant for said research. In the Real-time Spectrum Analysis and Spectrum Analysis modes, the device measures signals in an adjustable frequency spectrum. While long-term measurements for these modes would aid the IHF's research, the device does not support them for either. It instead allows users to save the current measurement or record measurements using a logging function. However, both functionalities are insufficient. Even though the 5G NR Beam Analysis mode does support long-term measurements, the device displays the data in a way which requires researchers to manually take notes. The device can be controlled remotely, which includes reading its raw sensor data. It is therefore feasible to develop an application that enables long-term measurements and visualizes 5G NR Beam Analysis measurements in the desired way.

3 Requirement Analysis

3.1 VIAVI ONA-800

The IHF conducts research on electromagnetic environmental impact. Its current focus is mobile telephony, specifically the current generation, 5G. In its research, the IHF uses the VIAVI ONA-800, a modular measuring device produced by VIAVI Solutions ([Via]) that provides multiple measurement and analysis modes. Only the following are relevant to this essay: Spectrum Analysis, Real-time Spectrum Analysis and 5G NR Beam Analysis. The Spectrum Analyser measures signals in an adjustable frequency spectrum. Additionally, the resolution bandwidth (RBW) and video bandwidth (VBW) can be adjusted. Neither will be discussed in greater detail, however both of them have an effect on the time it takes for the device to measure the entire spectrum (sweep time). Larger sweep times may result in shorter pulses not being detected correctly. The Real-time Analyser (RTSA) offers a persistent power measurement in real time of an adjustable spectrum. It is used to characterize signals with respect to their power, frequency and time. It can for example be used to locate 5G signals. The 5G NR Beam Analysis' EMF Analyser measures the field strength of 5G Synchronization Signal Blocks (SSB) of a station. Additionally, it decodes them and maps them to their respective Physical Cell IDs (PCI). While the device

does allow for measurements to manually be saved in different formats, only the current measurement can be saved this way. The device does offer a logging mode, which is only available in certain measurement modes. The log file mostly contains binary data, which the device can use to replay measurements like a video. Extracting the measurement data from the log file into a useful format for analysis is not possible, as VIAVI Solutions does not provide a tool to do so. While an application that allows users to operate the device remotely does exist, it only allows for a remote-desktop connection and therefore does not offer any functionality beyond what is already provided by the device. This means that neither the device itself nor an application provided by the manufacturer offers a way to measure field strength changes over time. In order to remedy this major drawback of the device, an application, which allows for long-term measurements and their visualization, should be developed.

3.2 Requirements and Constraints

In the following section, requirements and constraints of the application will be derived, while their implementation will be discussed in chapter 5. Figure 3.1 shows an Unified

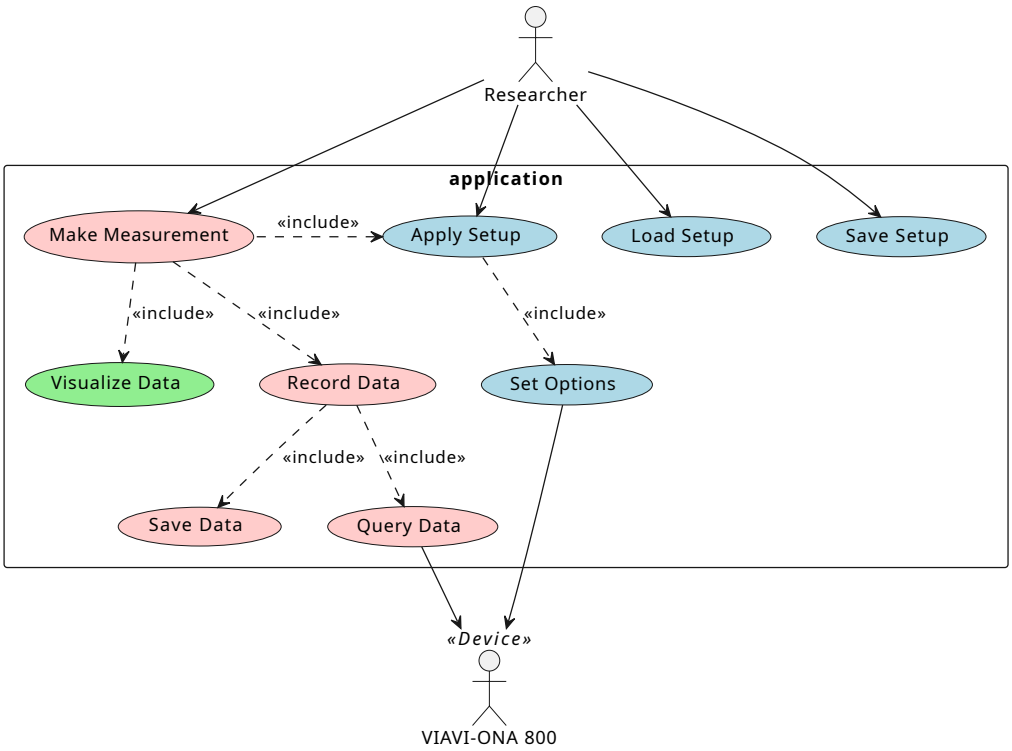

Fig. 3.1: Use case diagram of the application.

Modeling Language (UML) diagram, modelling the application's use cases, which can be grouped into three major functionalities that the application has to provide. The first of these is enabling long-term measurements, which applies to the Spectrum Analysis and Real-time Spectrum Analysis modes, as the device does not provide the functionality for

either. The processes that make up the first functionality are coloured in red. The second functionality is the visualization of measurement data, coloured in green, while the third is the handling of measurement setups, which includes saving, loading and applying the setups. Processes belonging to the third functionality are coloured in blue. From these functionalities, the first three requirements can be derived.

Functional Requirement 1: functional perspective

The application should enable the user to make long-term measurements.

Functional Requirement 2: functional perspective

The application should visualize measurement data during measurements.

Functional Requirement 3: functional perspective

The application should enable the user to handle measurement setups.

In the following, additional requirements, grouped by the aforementioned major functionalities, will be derived, beginning with F.R. 1. As extensions of F.R. 1, the application should allow the user to set a measurement's duration (F.R. 4), as well as stop a currently running measurement (F.R. 5), in case it was started erroneously. Furthermore, stopping a measurement should not result in a loss of measurement data in case the measurement was stopped manually.

Functional Requirement 4: functional perspective

The application should enable the user to set a measurement's duration.

Functional Requirement 5: functional perspective

The application should enable the user to stop a running measurement, without measurement data being lost.

In order to allow for intuitive operation and simplify the application's visualization aspect, it should include a Graphical User Interface (GUI). This GUI should provide input elements for the most used options (F.R. 6). However, it should not limit the options that can be set to those that have a dedicated input element. In order to maximize the application's functionality, the GUI should instead provide an input field through which all commands relevant to the current measurement mode, can be sent to the device (F.R. 7).

Functional Requirement 6: functional perspective

The application should contain input fields for the most used measurement options.

Functional Requirement 7: functional perspective

The application should enable the user to send any command, valid in a relevant measurement mode, to the device.

After specifying requirements pertaining to how the application can facilitate the process of recording measurements, the following will explore how the application should persist measurement data in order to simplify its analysis. From lightweight databases such as SQLite to text based formats, there is a multitude of ways to store tabular data. However, one format perfectly fits this use case, as it is extensively supported, simple and human-readable, and is therefore easy to work with. The comma-seperated values (CSV) format is used to store tables of data in textual form and can be imported in most programs concerned with data analysis. While there is a standard for the format, in practice many variants of the format are used, leading to some ambiguity as to what is and is not allowed within CSV files. One ambiguity, especially relevant to this essay, is whether CSV files may contain multiple tables. In this essay, files containing multiple tables are considered valid CSV, as in this case post-processing is simplified by having the measurement option table and measurement data both be contained in one file. The data should be saved at regular intervals during a running measurement in order to prevent data-loss due to for example loss of power (F.R. 10).

Functional Requirement 8: functional perspective

The application should store measurement-data in CSV files.

Functional Requirement 9: functional perspective

The application should store options that have an input field in the GUI, as well as custom options that were used during a measurement in the beginning of the CSV file.

Functional Requirement 10: functional perspective

The application should regularly save measurement data while it is being recorded.

After having specified more requirements related to making measurements, additional requirements related to the application's visualization capabilities will now be derived (F.R. 2). Measurement data should be visualized by the application, in order to enable users to immediately see abnormalities in the measurement and minimize the necessary direct interactions with the device. During default real-time spectrum measurements, the device visualizes the data as a line plot, however, the application should enable a visualization similar to the spectrogram.

Functional Requirement 11: functional perspective

The application should visualize real-time spectrum measurements as a spectrogram.

Functional Requirement 12: functional perspective

The application should visualize spectrum measurements as a line plot.

Functional Requirement 13: functional perspective

The application should visualize emf-beam measurements as a table.

As the first two major functionalities have now been expanded upon, leaving only the handling of measurement settings to be explored further, it will now be discussed in greater detail. As stated in F.R. 3, the application should allow for the reuse of measurement settings. Therefore, it must enable the user to save and load measurement setups. Setups should consist of the mode they are meant to be used for, the measurement options required by the GUI and the custom options provided by the user. Additionally, the application should enable the user to apply the currently selected settings to the device in order to ensure that they have the desired effect.

Functional Requirement 14: functional perspective

The application should enable the user to save setup files.

Functional Requirement 15: functional perspective

The application should enable the user to load previously saved setup files.

Functional Requirement 16: functional perspective

The application should enable the user to apply a setup to the device.

Functional Requirement 17: data perspective

The application should store the measurement mode, as well as required options and custom options alongside their values in setup files.

After working out the requirements, the constraints will now be discussed. The application should be able to run on a diverse set of platforms, as the type of device the application needs to run on may change in the future. In order to ensure that the application can run on many devices, a cross-platform framework should be used. Additionally, external dependencies should only be used when strictly necessary, as they would also have to be platform independent in order to satisfy Con. 1, and deploying them increases complexity.

Einschränkung 1

The application should be able to run on Windows 10/11 (x86_64), Linux (x86_64, arm64).

Einschränkung 2

The application should only use external libraries if it is not feasible to implement the functionality manually.

4 Identification and Choice of Tooling

4.1 Framework

Given the considerable range of GUI frameworks, selecting an appropriate one can be challenging. The number of frameworks is already greatly reduced by Con. 1, as frameworks that are not cross-platform do not need to be evaluated. By only considering well established frameworks, this number can be reduced further. As this project may be expanded upon in the future, it is important to identify those that are likely to do so. The only groups that are likely to work on the application in the future are the IHF's research assistants and mathematical-technical software development trainees. In order to reduce the amount of frameworks that need to be considered even further, those that do not support a language the trainees are likely to learn during their training are eliminated from consideration. Due to all trainees also studying Applied Mathematics and Computer Science (B.Sc.) at the FH-Aachen, languages prevalent during the course of study are particularly relevant to this discussion. Specifically Java and JavaScript are a required part of it. In addition to these, trainees are likely to be sufficient in C++ and Python, as learning C++ is encouraged and Python is used in multiple modules. Thus, only well established frameworks available in Java, C++, JavaScript or Python remain in consideration. The frameworks that will be considered are Qt (C++) [Qt2], GTK (C++) [Tea], Kivy (Python) [Kiv], Electron (JavaScript) [Ele] and JavaFX (Java) [Jav], as all are well-established and each relevant language is represented. Furthermore, criteria that can be used to evaluate the aforementioned options need to be defined. The language proficiency of trainees and research assistants may be used as such, as both might need to extend the application in the future. Another important criterion is the degree to which trainees or research assistants have experience with a framework. In order to estimate the language proficiencies of research assistants and trainees, the relevant groups were asked to rate their proficiency in the relevant languages on a scale from zero to three. They were also asked to rate their level of experience with the preselected frameworks on the same scale. Three trainees and two research assistants were questioned. Each groups language proficiency results were summed up and are presented in table 4.1. Similarly, the results of the framework experience survey are shown in table 4.2. As the criteria are deemed equally important, a framework's viability score is the unweighted sum of the language proficiency score and the framework experience score that were just derived. The results of this evaluation are shown in table 4.3 and indicate that Qt is the framework that should be given precedence.

	C++	Java	JavaScript	Python
Trainees	5	8	7	4
Research assistants	4	2	0	3
Sum	**9**	**10**	**7**	**7**

Table 4.1: Language proficiency survey results

	Qt	GTK	Kivy	Electron	JavaFX
Trainees	2	0	0	2	0
Research assistants	1	0	0	0	1
Sum	**3**	**0**	**0**	**2**	**1**

Table 4.2: Framework experience survey results

	Qt	GTK	Kivy	Electron	JavaFX
Language proficiency	9	9	7	7	10
Prior experience	3	0	0	2	1
Sum	**12**	**9**	**7**	**9**	**11**

Table 4.3: Framework comparison

4.2 Language Selection

Qt [Qt2] is primarily a C++ framework but provides Python bindings as well [Qtp]. Therefore, the language that best fits the requirements needs to be determined. With languages as dissimilar as C++ and Python, differences in performance and memory consumption have to be considered as criteria for language selection. Whether the amount of measurement points per second that can be observed using Python differs from the amount observable with C++ is of particular interest. This will be tested by continuously querying the device for data while the device is set up to measure with a low sweep time for five minutes. Because the device is queried much more frequently than it generates new data, two things are being tracked during the measurements: firstly, the total number of values received; secondly, the number of unique values received. As table 4.4 shows,

	Unique measurements	Total measurements	Time (ms)	Total points per second
Python	7499	16238	300009	54.125
C++	7563	17129	300004	57.096

Table 4.4: Querying speed comparison in a five-minute interval Python vs. C++.

C++ only recorded slightly more points than Python. However, this can be explained by the communication via TCP taking a majority of the time. The *socket* package [Pytb] directly calls operating system APIs, thus the actual networking is handled by the operating system, which is written in a much faster language. Therefore, the bulk of the Python program's runtime is not spend executing actual Python code. Taking this into account, the difference between the two languages is more significant than it appears. The result, however, is not surprising, given that Python can be orders of magnitude

slower, as the benchmarks game made by the benchmarks game team at Debian shows [Pyta]. In addition to performance criteria, the development process must also be examined. Development with Qt is primarily done in the Qt Creator [Qtd], a cross-platform integrated development environment. The creator contains the design tab, where graphical user interfaces can be designed. These designs are stored in *.ui* files, a XML based format. During building, the *.ui* files need to be converted to C++ or Python files. Qt Creator automatically handles these conversions, meaning neither language provides any benefit in this area. When developing a C++ application with Qt Creator, compile times can quickly be multiple minutes long when a full compilation is triggered, while Python startup times remain fairly consistent. This can become an issue when changes to the build pipeline need to be made, as is may become necessary to repeatedly recompile the program. However, these changes rarely need to be made, and full compilations are also rarely necessary, meaning Python has only a negligible advantage in this area. Given the reasons outlined above, the application will be written in C++.

5 Implementation

5.1 Remote Connection

A connection to the device may be established via USB or by TCP connection using either Ethernet or WIFI. Only the TCP connection via Ethernet will be discussed in this essay. Qt already provides a class that handles the TCP connection; its *QTcpSocket* class. It provides blocking functions that wait for specific events, such as receiving data. However, their use is discouraged in the framework's documentation, as they are unreliable in certain circumstances, like the application running on Windows. As stated in Con. 1, the application should explicitly be able to run on Windows, therefore, the aforementioned functions will not be used. Instead, the connections state will be modelled as a finite state machine. This approach is easily extendable and allows for simple state management and automated reconnection, should the connection be lost unexpectedly. A complete visualization of the states can be found in figure 5.1. The initial state is the *Connecting* state, in which an attempt to connect to the device is made. This attempt may time out, resulting in a change of state to *ConnectionTimeout*, which, when entered, aborts the connection attempt and re-enters the *Connecting* state. The *QTcpSocket* may also notice that an error occurred or that it is disconnected. In case of an error occurring, the state changes to *Error*. When the *Error*-state is entered, an attempt to disconnect from the device is made. If successful, the state changes to connecting and a new connection attempt is made, else the *DisconnectionTimeout*-state is entered. When entering this state, the socket is deleted and a new socket is created. Afterwards the *Connecting*-state is entered and a new connection attempt is made. Deleting the socket is necessary, because the framework's event loop may otherwise become stuck in a state that does not allow further connection attempts to succeed. If a connection attempt is successful, the *Connected*-state is entered, from which, depending on calls to the classes methods, either

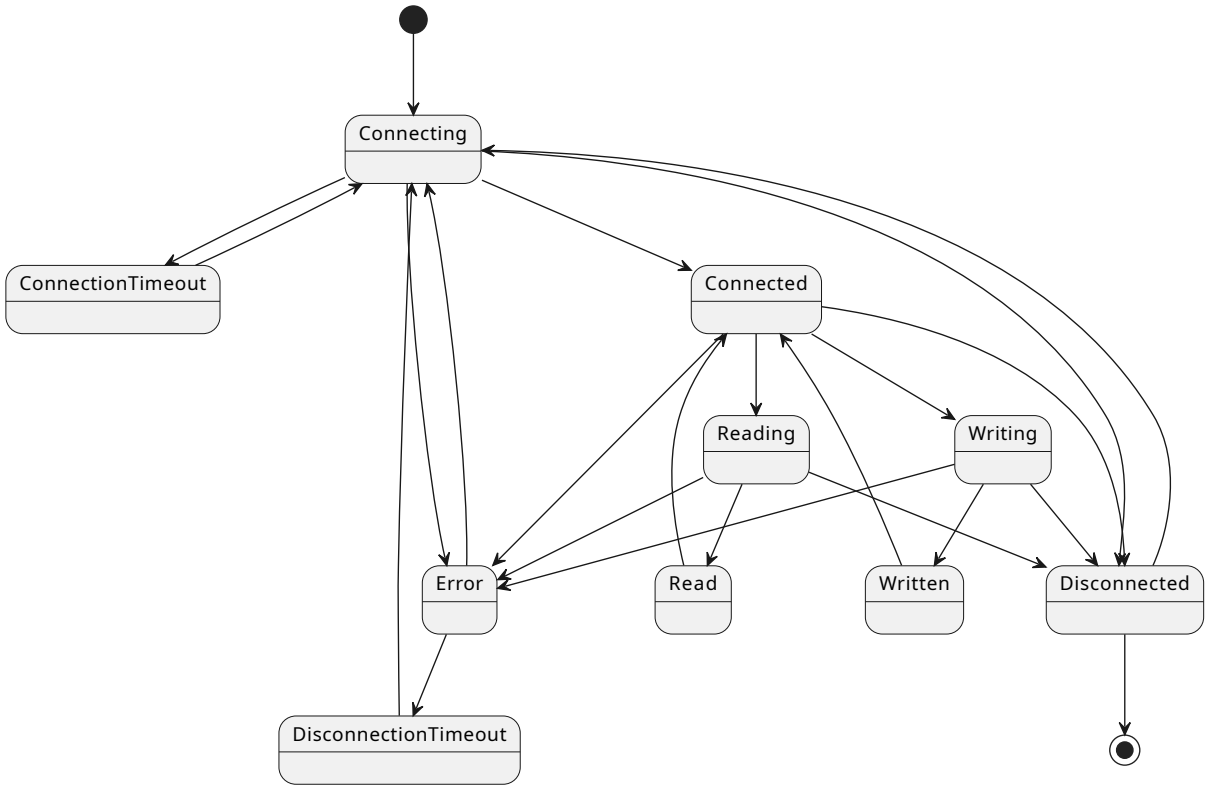

Fig. 5.1: Connection State Diagram

the *Reading*- or *Writing*-state will be entered. Afterwards, in case the reading or writing operation succeeds the *Read*- or *Written*-state is entered. From both the state is changed to *Connected*. When in the *Reading*-, *Write*-, *Read*-, *Written*- or *Connected*-state a state change to *Error* or *Disconnected* may occur.

5.2 Operating the Device

The device can be controlled remotely by sending commands to it. These commands are documented and have a consistent structure. An exemplary command which sets the real-time measurements starting frequency to 5 GHz is: *REALtime:FREQuency:STARt 5 GHz*. The commands consist of fragments separated by ":". Commands that set an options value are followed by the desired value. Those that query an options value are instead followed by a "?". A command must end with a newline character in order to be recognized. Each fragment may consist of upper- and lowercase letters, however, only the uppercase letters are necessary to identify the fragment. The device's documentation specifies that the lower case letters may be omitted, thus shortening the commands. However, in order to increase readability, both for the user and during development, only the complete form will be used. Malformed commands may result in the device no longer responding to any other command. Sending them must therefore be avoided. The GUI provides input fields for the most used settings; therefore, a setup file must include a value for these

settings. Since there are relatively few required options, it is feasible to enforce their correctness. Required options are represented internally by the class hierarchy shown in figure 5.2. Some general classes, that are useful for adding additional required options are already provided. *OptionWithUnit*, for example, provides a class from which other classes, that represent options whose values have a unit, can inherit. The *FrequencyOption* class is used to represent the *FREQuency:STARt*, *FREQuency:STOP*, *FREQuency:CENTer* and *FREQuency:SPAN* options; additionally, it handles conversion between units. These options are modelled together in the wrapper class *RtsaFreqOptions*, which automatically updates the other options when one of them is changed. For example, in case the starting frequency is changed the center frequency and frequency span need to be adjusted as well in order to keep all options consistent. To enable users to use the full functionality of the

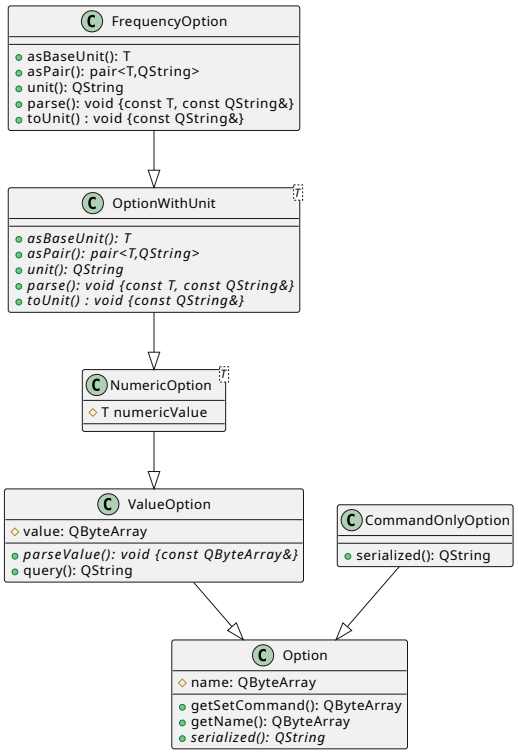

Fig. 5.2: UML class diagram of the *Option* classes

device, as well as allowing them to save their complete measurement setups within the application (F.R. 14), the GUI also provides a text input field through which the user can enter additional commands (F.R. 7). The input field allows entering arbitrary text. In order to reduce the likelihood of sending malformed commands, user-provided commands are validated. During validation, each provided line is verified to be one of the possible commands, by comparing it to a pre-generated list of all possibly relevant commands. The list was generated using a Python script by parsing the command documentation and isolating the commands that belong to one of the relevant modes. However, it is not feasible to validate the arguments of each user-provided command within the scope of this essay.

5.2.1 Persistence of Measurement Data

Measurement data is saved as *CSV* files (F.R. 8) in the application's *savedata* subdirectory. The files consist of two tables, which are separated by a blank line. The first table contains the measurement options that were used, including their values, as per F.R. 9. It is prefixed accordingly by the header "option,value". In addition to measurement options, this table also contains additional information such as its duration and starting datetime. The second table contains the measurements values, prefixed by a fitting header. In the Real-time Spectrum Analysis and Spectrum Analysis modes, the data has headers consisting of the frequencies the measurements were taken at. When in 5G NR Beam Analysis mode, the header consists of all SSB and PCI pairs that occurred during the measurement. Exemplary save files of a real-time analysis and a 5G NR beam analysis can be seen in listings 5.1 and 5.2 respectively. In order to satisfy F.R. 10, 5G NR Beam Analysis data is written to file after every measurement. Even though writing to disk is a slow process, doing so does not risk missing new measurements because the device generates new data only once every two seconds in this mode. Saving every measurement immediately is not possible for the other modes, as the device generates data much more frequently. The application buffers the measurements and only writes to a file when the buffer size surpasses one megabyte, thereby still offering protection against data loss. The specific buffer size ensures that in practice at most the last ten seconds of a measurement are lost, which is deemed exceptable.

```
1  option,value
2  datetime,Thu Dec 5 15:35:05 2024
3  duration,300
4  MODE,realtimeAnalyzer
5  REALtime:FREQuency:CENTer,5325
6  REALtime:FREQuency:SPAN,50
7  REALtime:FREQuency:STARt,5300
8  REALtime:FREQuency:STOP,5350
9  REALtime:RBW,3
10 REALtime:TRAce:SELect,Trace01
11
12 5300000000,5300062500,5300125000,5300187500,...
13 -69.6990620900936,-69.7224995900936,-69.7576558400936,
       -69.7928120900936,...
14 -68.3826558400936,-68.3826558400936,-68.3943745900936,
       -68.4178120900936,...
15 -69.4178120900936,-69.4764058400936,-69.5232808400936,
       -69.5701558400936,...
16 ...
```

Listing 5.1: Real-time Spectrum Analysis save file example

```
1 option,value
2 datetime,Mon Dec 9 11:32:50 2024
3 duration,300
4 EMF:FREQuency:CENTer,3574.56
5 EMF:FREQuency:STARt,3524.56
6 EMF:FREQuency:STOP,3624.56
7
```

```
260(1),260(2),260(0),260(3),260(4),260(6),260(5)
-86.21258004893201,-98.58935289370777,-100.4800816729178,
    -105.6342560308601,-108.305205842014,-110.0361573080027,
    -110.1330062275053
-86.21258004893201,-98.58935289370777,-100.4800816729178,
    -105.6342560308601,-106.7800828905056,-110.0361573080027,
    -109.0955254409092
-86.21258004893201,-98.58935289370777,-100.4800816729178,
    -105.6342560308601,-106.2261475232816,-108.5728725027861,
    -108.783421286506
-86.21258004893201,-98.58935289370777,-100.4800816729178,
    -105.6342560308601,-106.2261475232816,-108.5728725027861,
    -107.2354536215282
...
```

Listing 5.2: 5G NR Beam Analysis save file example

5.2.2 Persistence of Measurement-Setups

Measurement setups have to be stored persistently in order to satisfy requirements F.R. 15 and F.R. 14. They are stored in the subdirectory *setup_files* of the application's directory. Each setup is saved in a file prefixed by *spectrum*, *realtime*, or *emfbeam*, depending on the setup's measurement mode, followed by the setup's name and the *.txt* file extension. This makes it easier to find a specific file in the GUI's dropdown menu. Setups are saved in a text based format, as listing 5.3 shows. The first line begins with *#mode* , followed by either *REALtime*, *SPECtrum*, or *EMF:NRBEam*, identifying the measurement mode the file can be used for. The commands following the first line are those that have a corresponding input field in the GUI, such as the starting frequency *REALtime:FREQuency:STARt*. These are followed by *#customOptions*, marking the end of the specifically implemented command and the beginning of the custom commands that users may provide in the GUI. Commands are separated by "=" from their value; if a command does not set a value, the "=" is immediately followed by a newline character. In addition to satisfying F.R. 17, this makes for an easily human-readable format.

```
#mode REALtime
REALtime:FREQuency:STARt=5.3 GHz
REALtime:FREQuency:STOP=5.35 GHz
REALtime:FREQuency:CENTer=5.325 GHz
REALtime:FREQuency:SPAN=50 MHz
REALtime:RBW=3 MHz
REALtime:SCALe:AUTO=
#customOptions
REALtime:TRAce:SELect Trace01
```

Listing 5.3: Setup file example

5.3 Graphical Representation

After examining how the device is operated remotely, this section will outline how measurement data is visually represented in the application. First, however, the GUIs general structure will be examined. As figure 5.3 shows, the GUI consists of two major sections: the visualization section to the left and the settings bar on the right. The latter consists of multiple sections, that are shown in figure 5.4 as coloured boxes. The connection's status is shown at the top, highlighted by the yellow box. At the bottom the measurement settings, such as its save file, are grouped together, highlighted by the green box. It also includes fields that allow the user to set a measurement's duration, as required in F.R. 4, and a button to stop an ongoing measurement, as per F.R. 5. Options pertaining to the measurement setup can be found in the blue box, as well as a button that enables the user to apply the setup to the device (F.R. 16). It also contains the setup options that have their own input fields as specified in F.R. 6, which are shown in the red box. These are the only element, of the sidebar that are subject to change, depending on the selected measurement mode. After examining the GUI's general structure, the following sections will explore how the measurement data is visualized in each mode.

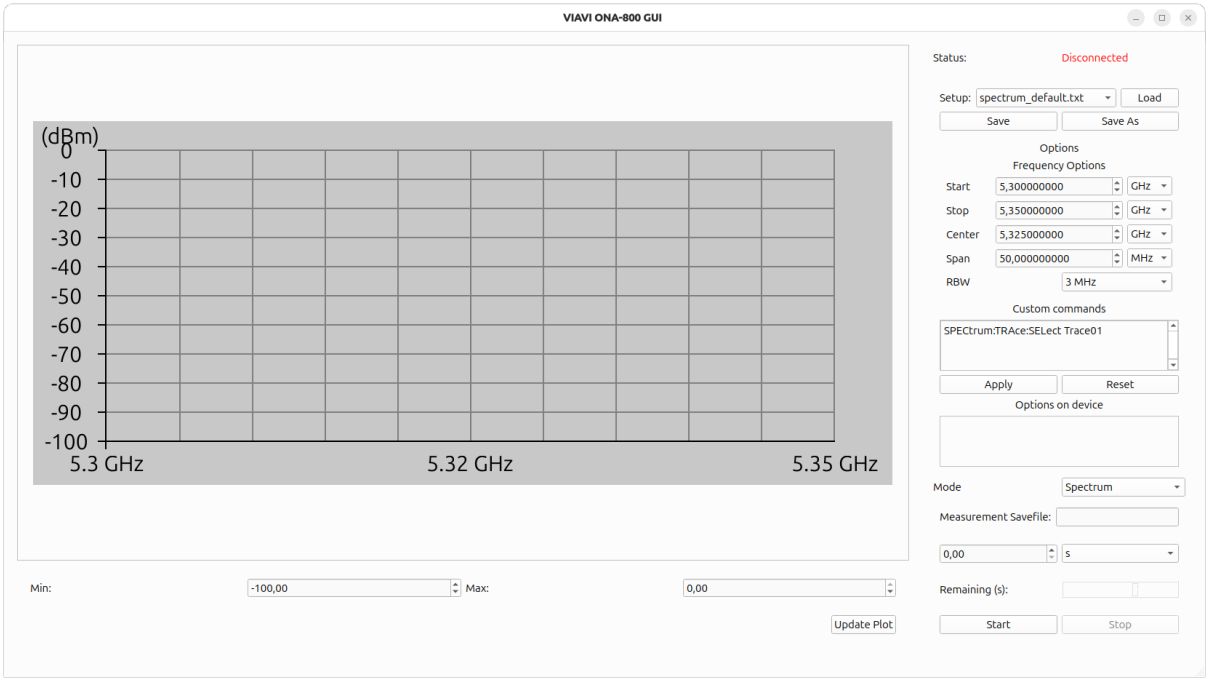

Fig. 5.3: GUI in Spectrum Analysis mode (Linux version)

5.3.1 Spectrum Analysis

Drawing the spectrum modes plot is more complex because the plotting modules Qt provides are too slow to keep up with the incoming data. While there are libraries that provide plots for technical applications such as Qwt [Qwt], Con. 2 states that external

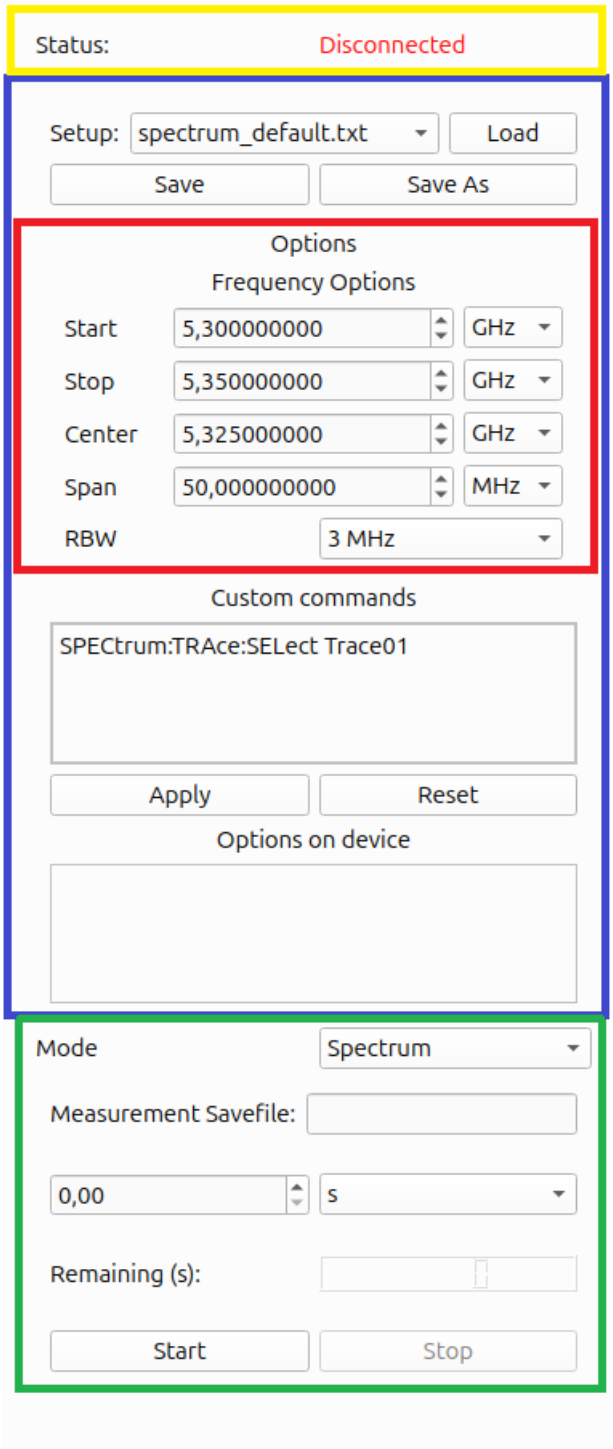

Fig. 5.4: Sidebar close up (Linux version)

dependencies should only be used when necessary. Therefore, the plot must be drawn manually using features already provided by Qt. Qt provides multiple classes for handling image data; of these *QPixmap* is the class best suited to this particular use case, as it is optimized to show images on screen [Qtqc]. It can be displayed by setting it as the *pixmap*

property of a *QLabel* [Qtqa]. As the data should be visualized as a line plot (F.R. 12), it can be decomposed into two parts: the background consisting of the axes, their labels and a grid meant to increase readability, and the line connecting the measured points. While the background only has to be updated when the user changes the plots scale, the line has to be redrawn for each measurement. Not re-rendering the background at every frame greatly increases performance. Since the background can be reused, it is stored in a separate *QPixmap*. When a new measurement is registered, the background is copied into the display *QPixmap* and the lines connecting the data points are rendered on top, using the *drawLines* function provided by *QPainter*. In addition to the plot, the display section for spectrum measurements also contains input fields that allow the user to update the section of the y-axis that is shown. The entire section can be seen in figure 5.5.

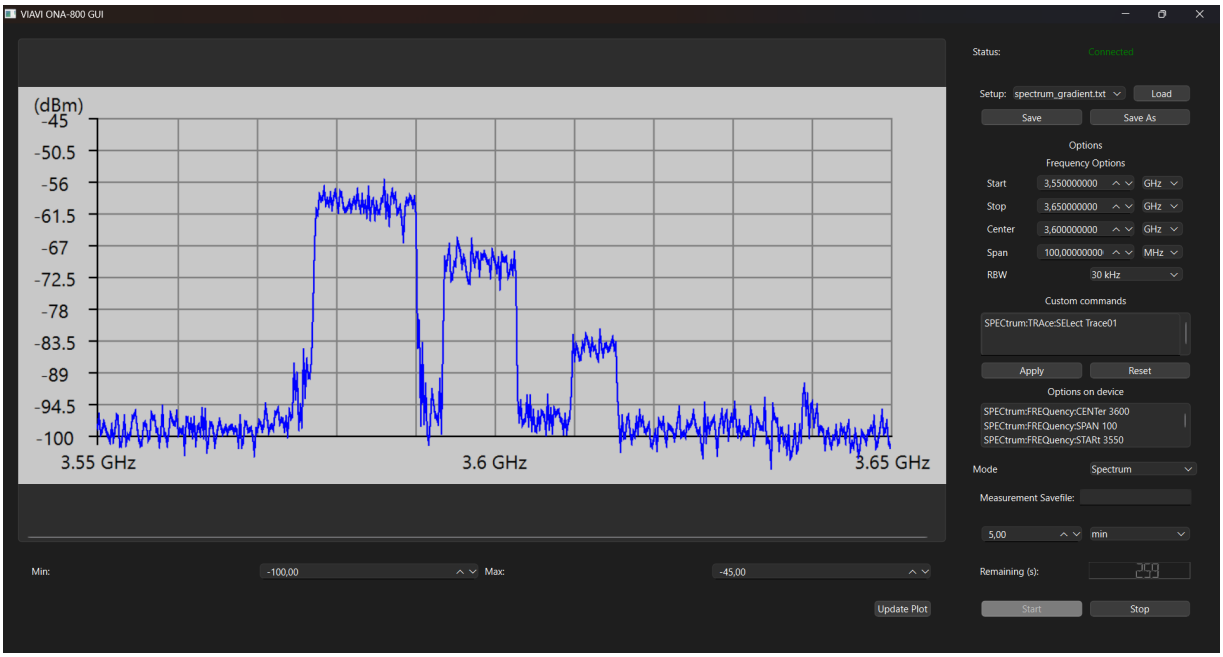

Fig. 5.5: Spectrum measurement (Windows version)

5.3.2 Real-time Spectrum Analysis

The Real-time Spectrum Analysers data has to be visualized as a heatmap, as specified by F.R. 11. Since Qt does not provide a module to draw heatmaps, the heatmap will have to be rendered manually as well. Like the previous section, this will be done by drawing on a *QPixmap* and displaying it in a label. Unlike the plot in previous section, the real-time spectrums heatmap does not require axes. It consists solely of points coloured according to the signal strength of the corresponding frequency at a point in time. Qt does not provide a way to generate the type of colour gradient required, thus, a colour gradient needs to be generated. In order to generate the gradient, colours are treated as points in a colour space with (red, green, blue) coordinates. A gradient of n colours, between colours $\vec{a}$ and $\vec{b}$ can then be generated by taking colours spaced across the line connecting them. This results in $(\vec{a}+i/(n-1)*(\vec{b}-\vec{a}))_{i=0}^{n-1}$ as the sequence of colours in the gradient.

However, as figure 5.6 shows, this method results in gradients with a very narrow range of colours, making them difficult to read. In order to remedy this, the following method is used. An array of different colours the gradient should contain is defined. Then, beginning with the second colour, the gradient from the previous to the current colour is generated using the method outlined above. The generated gradients are then merged in the order they were generated in, resulting in a final gradient that smoothly transitions between the colours defined above. This method results in gradients that are much more readable by providing a greater variety of colours, as figure 5.7 shows. As the gradients size is determined during development and the gradient does not need to be modified after initialization, it can be generated at compile time. This means that the generated gradient will be contained in the executable and does not need to be generated at runtime. To efficiently render the measurement points, they need to be grouped by colour and be

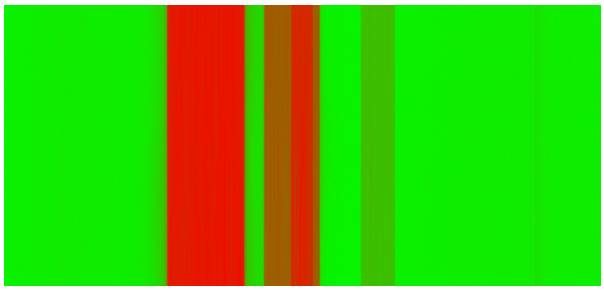

Fig. 5.6: Gradient from yellow to red

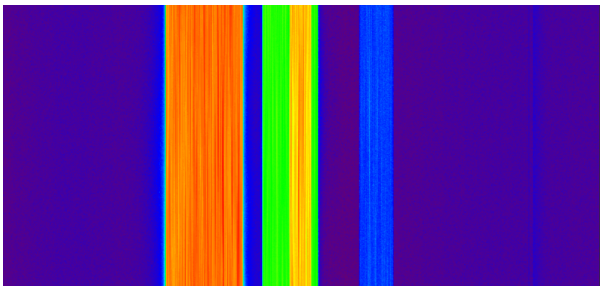

Fig. 5.7: Gradient between multiple colours

stored contiguously in memory to facilitate the use of the *drawPoints* function provided by *QPainter* [Qtqb]. Since only a certain number of measurements should be rendered, a data structure which allows for efficient deletion of old measurements and adding of new measurements is necessary. One such structure is a queue. Specifically, an implementation that uses a circular buffer, since the data has to be laid out contiguously. When storing the points for each colour separately in circular buffers, one only needs to keep track of the amount of points that were added to each colour per measurement in order to drop the correct amount of points from each colour and add the newest points afterwards. Similarly, the point count for each colour per measurement is also stored in a circular buffer. The display section of the real-time mode also includes input elements in addition to the plot. These enable the user to either set the plots scale statically or enable dynamic scaling. The section is shown in figure 5.8.

5.3.3 5G NR Beam Analysis

Representing the 5G NR Beam Analysis data is the simplest as it can simply be represented in a table (F.R. 13). The data will not need to be updated many times per second, thus, the table class provided by the framework is sufficient. The table consists of three columns: the first simply being an index, the second being the PCI and SSB index and the third being the maximum, as can be seen in figure 5.9. This representation is easily extendable, as one can simply add another column if, e.g., the average value should also be tracked.

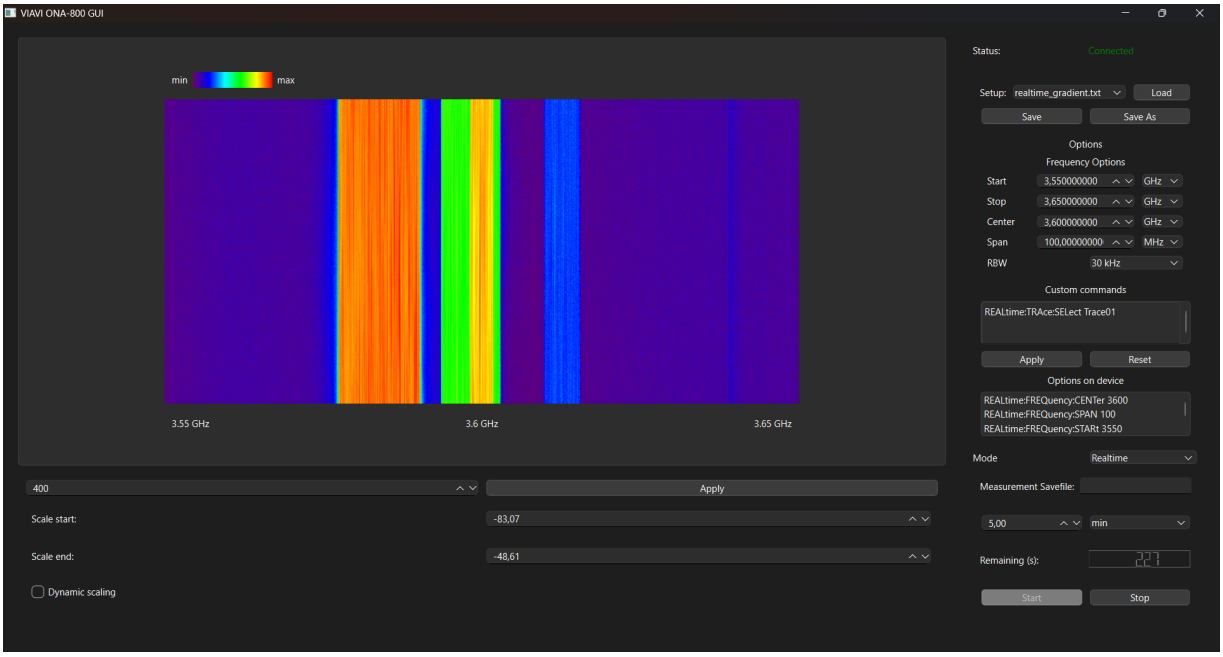

Fig. 5.8: Real-time measurement (Windows version)

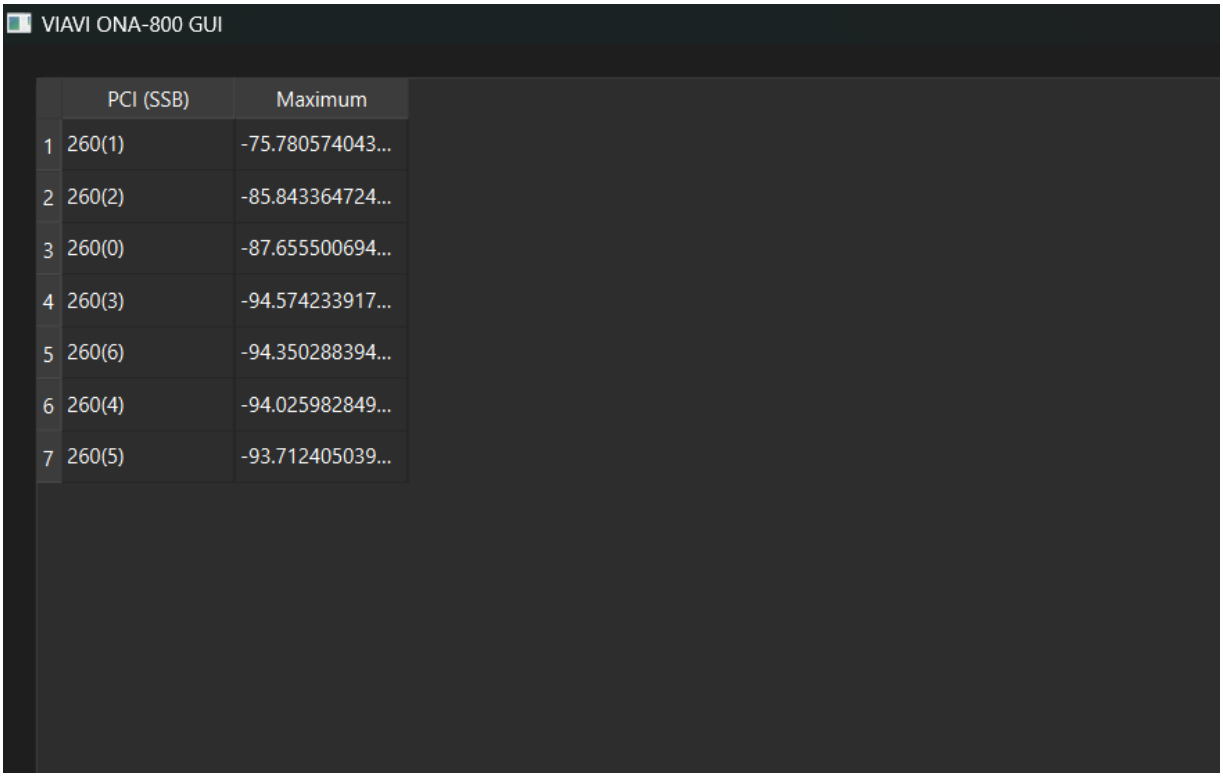

Fig. 5.9: 5G NR Beam measurement section (Windows version)

6 Conclusion

After deriving requirements for the application in chapter 3, choosing a framework in chapter 4 and discussing the implementation of the application in chapter 5, this chapter will provide this essay's conclusion and give an outlook on future work. The application fully satisfies the requirements that were derived in chapter 3. Furthermore, the application satisfies both constraints, as Con. 1 was a major factor during framework selection and the choice to manually draw the necessary plots, instead of relying on an external library, was made in order to satisfy Con. 2. However, in order to evaluate the application's usefulness for making long term measurements, a comparison to the device itself is needed. Comparing the application's performance to the device's is challenging, as the device does not provide the same functionality. Nevertheless, as stated in section 3.1, it provides logging functionality that records measurements and can therefore be compared to the application's ability to make long term measurement. In order to compare them, each recorded a persistent real-time spectrogram for a duration of five minutes. Even though the data recorded by the device can not be accessed directly, the number of measurements is displayed in its Real-time Spectrum Replayer. While the device recorded 1528 measurements during a five-minute period, the application was able to record 2998 measurements. Therefore, the application would be superior to the devices logging mode even if the manufacturer published a tool which allowed users to extract measurement data from the log files. It can therefore be concluded that the application represents a major improvement of the VIAVI ONA-800's functionality. While the application fully satisfies the requirements, there are additional functionalities that could be implemented, leaving room for future work. For example more of the measurement options could be integrated into the GUI. As mentioned in section 5.2, sending malformed commands to the device must be avoided. However, the application still allows for some malformed commands to be sent to the device, as it does not validate the arguments of custom commands. This could be remedied in the future. Furthermore, the application could be extended to allow users to select the unit in which measurement data should be represented in a save file, as the device supports multiple units as well.

List of Figures

List of Tables

Listings

Bibliography

[Ele] *Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron — electronjs.org.* https://www.electronjs.org/. [Accessed 16-12-2024].

[Jav] *JavaFX — openjfx.io.* https://openjfx.io/. [Accessed 16-12-2024]. (Visited on 12/16/2024).

[Kiv] *Kivy: Cross-platform Python Framework for NUI — kivy.org.* https://kivy.org/. [Accessed 16-12-2024]. (Visited on 12/16/2024).

[Pyta] *Python 3 vs C++ g++ - Which programs are fastest? (Benchmarks Game) — benchmarksgame-team.pages.debian.net.* https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gpp.html. [Accessed 26-11-2024]. (Visited on 11/26/2024).

[Pytb] *socket — Low-level networking interface — docs.python.org.* https://docs.python.org/3/library/socket.html. [Accessed 26-11-2024]. (Visited on 11/26/2024).

[Qt2] *Qt | Tools for Each Stage of Software Development Lifecycle.* en. URL: https://www.qt.io/ (visited on 11/04/2024).

[Qtd] *Embedded Software Development Tools & Cross Platform IDE | Qt Creator — qt.io.* https://www.qt.io/product/development-tools. [Accessed 26-11-2024]. (Visited on 11/26/2024).

[Qtp] *Python UI | Design GUI with Python | Python Bindings for Qt — qt.io.* [Accessed 04-11-2024]. URL: https://www.qt.io/qt-for-python (visited on 11/04/2024).

[Qtqa] *QLabel Class | Qt Widgets 6.8.0 — doc.qt.io.* https://doc.qt.io/qt-6/qlabel.html. [Accessed 28-11-2024]. (Visited on 11/28/2024).

[Qtqb] *QPainter Class | Qt GUI 6.8.0 — doc.qt.io.* https://doc.qt.io/qt-6/qpainter.html. [Accessed 11-11-2024].

[Qtqc] *QPixmap Class | Qt GUI 6.8.0 — doc.qt.io.* https://doc.qt.io/qt-6/qpixmap.html. [Accessed 28-11-2024]. (Visited on 11/28/2024).

[Qwt] *Qwt User's Guide: Qwt - Qt Widgets for Technical Applications — qwt.sourceforge.io.* https://qwt.sourceforge.io/. [Accessed 28-11-2024]. (Visited on 11/28/2024).

[Tea] The GTK Team. *The GTK Project - A free and open-source cross-platform widget toolkit — gtk.org.* https://www.gtk.org/. [Accessed 16-12-2024]. (Visited on 12/16/2024).

[Via] *VIAVI Solutions Inc. | — viavisolutions.com.* https://www.viavisolutions.com. [Accessed 11-12-2024]. (Visited on 12/11/2024).