RTL-SDR Spectrum Analyzer Project

Adam Aukamp

April 25, 2023

Design Overview

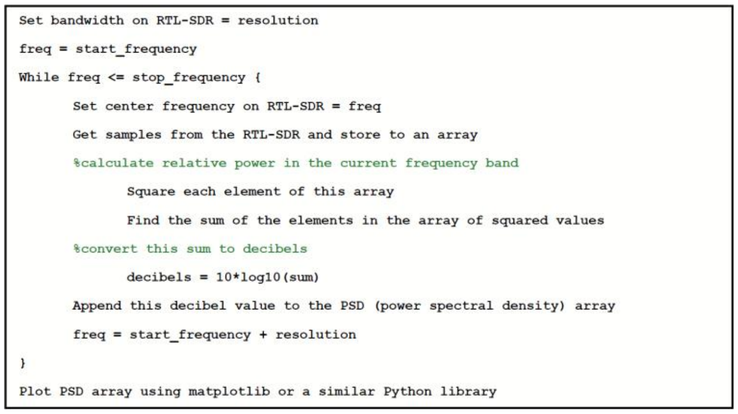
The goal of this project was to design an application that utilized the RTL-SDR software defined radio USB dongle as a spectrum analyzer. The RTL-SDR delivers I-Q samples tuned at a certain frequency to the computer that is connected to it. These samples can then be processed in software to recover the data or audio that was transmitted.

The purpose of this project was to write a program that collected samples from the RTL-SDR and used them to generate wideband power spectral density (PSD) plots. Additionally, a GUI was to be written that allowed the user to enter parameters like start frequency, stop frequency, resolution, and plot settings. Python was chosen as the programming language for this project, since there are many libraries available for math operations and plotting (e.g., NumPy and Matplotlib). Additionally, pyrtlsdr is a simple Python library that can be used for communication with the RTL-SDR.

Original Design

A limiting factor of the RTL-SDR is that it has a maximum tuning bandwidth of about 2.4 MHz [1]. This means that if it tuned to a frequency f_c, the RTL-SDR can only "see" frequencies between $f_c - 1.2\ \mathrm{MHz}$ and $f_c + 1.2\ \mathrm{MHz}$. However, the tuning frequency of the RTL-SDR can be adjusted from about 30 to 1700 MHz. Thus, to scan a band of spectrum greater than 2.4 MHz, software could adjust the tuning frequency of the RTL-SDR in 2.4 MHz steps and compute the spectrum for each "chunk".

Two initial designs were proposed for the spectrum generating algorithm. The first algorithm written in code swept a range of frequency bands, computed the power in each band, and appended each power value to a PSD array. A very rough version of the pseudocode for this algorithm (from the initial proposal) is shown in figure 1 below.

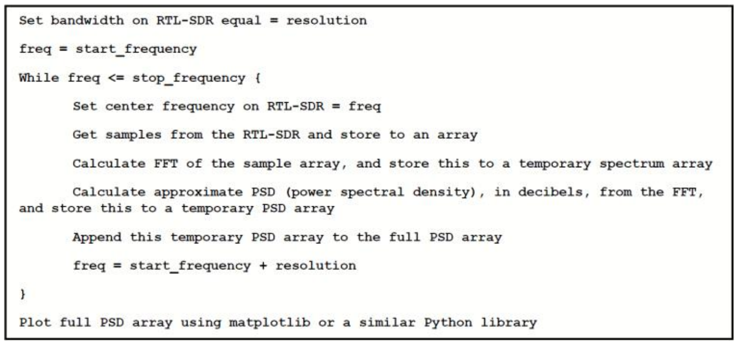


```
Set bandwidth on RTL-SDR = resolution
freq = start_frequency
While freq <= stop_frequency {
        Set center frequency on RTL-SDR = freq
        Get samples from the RTL-SDR and store to an array
        %calculate relative power in the current frequency band
                Square each element of this array
                Find the sum of the elements in the array of squared values
        %convert this sum to decibels
                decibels = 10*log10(sum)
        Append this decibel value to the PSD (power spectral density) array
        freq = start_frequency + resolution
}
Plot PSD array using matplotlib or a similar Python library
```

Figure 1: Pseudocode – Power in Each Band Method

The first design proved to be insufficient in many cases. I quickly discovered that the RTL-SDR has a minimum tuning bandwidth of 0.25 MHz, so generating plots with a resolution of finer than 0.25 MHz (250 kHz) was not possible using this method! This was problematic since many signals are narrower than 250 kHz, including broadcast FM radio stations which are only about 200 kHz wide. Additionally, the part of the code that turned out to take the longest amount of time (by far), was collecting samples from the RTL-SDR. So, sweeping chunks of spectrum took a considerably long time at the finest resolution.

The second algorithm written in code computed an FFT for each frequency band tuned, rather than just a total power. This allowed the plotted PSD to have much finer resolution than the tuning bandwidth of the RTL-SDR. Below in figure 2 is the rough pseudocode for this version, as described in the initial proposal assignment.



```
Set bandwidth on RTL-SDR equal = resolution
freq = start_frequency
While freq <= stop_frequency {
        Set center frequency on RTL-SDR = freq
        Get samples from the RTL-SDR and store to an array
        Calculate FFT of the sample array, and store this to a temporary spectrum array
        Calculate approximate PSD (power spectral density), in decibels, from the FFT,
and store this to a temporary PSD array
        Append this temporary PSD array to the full PSD array
        freq = start_frequency + resolution
}
Plot full PSD array using matplotlib or a similar Python library
```

Figure 2: Pseudocode – FFT in Each Band Method

Due to the inherent limitations of the first method (power in each band), the second method (FFT in each band) was chosen for the final design.

Changes to the Design

Dealing with Noisy Samples

The first problem discovered was that the RTL-SDR delivered noise for the first ~1700 samples each time samples were read in from it. Figure 3 below shows what was happening in the discrete-time domain when the center frequency of the RTL-SDR was set to 89.3 MHz (the frequency of nearby WQED-FM).

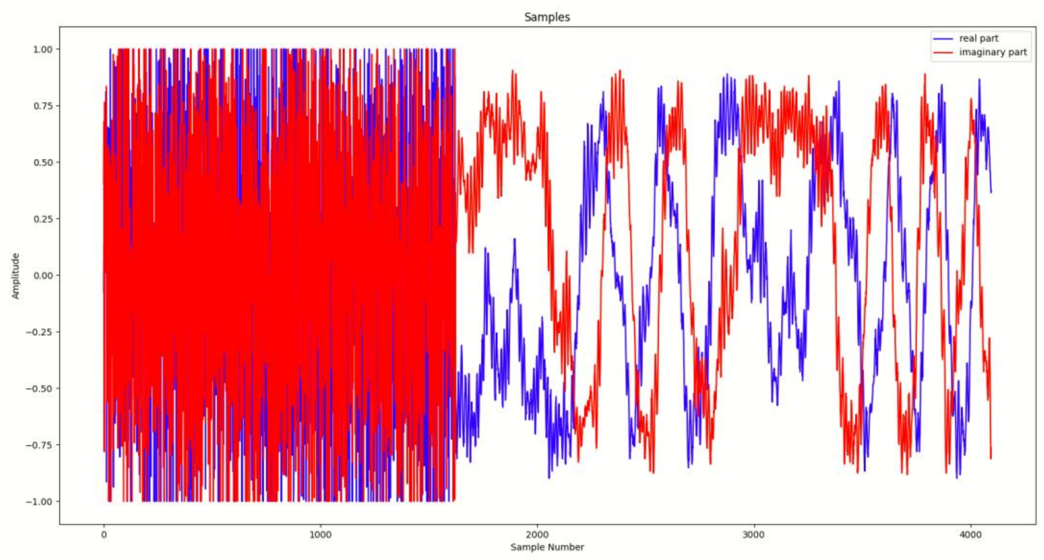


Figure 3: Example of 4096 samples read in from the RTL-SDR

This phenomenon caused the spectrum to just look like noise when calculating the N-length FFT from the first N samples in the signal. For example, figure 4 below shows the magnitude spectrum calculated from the first 1024 samples in the signal from figure 3.

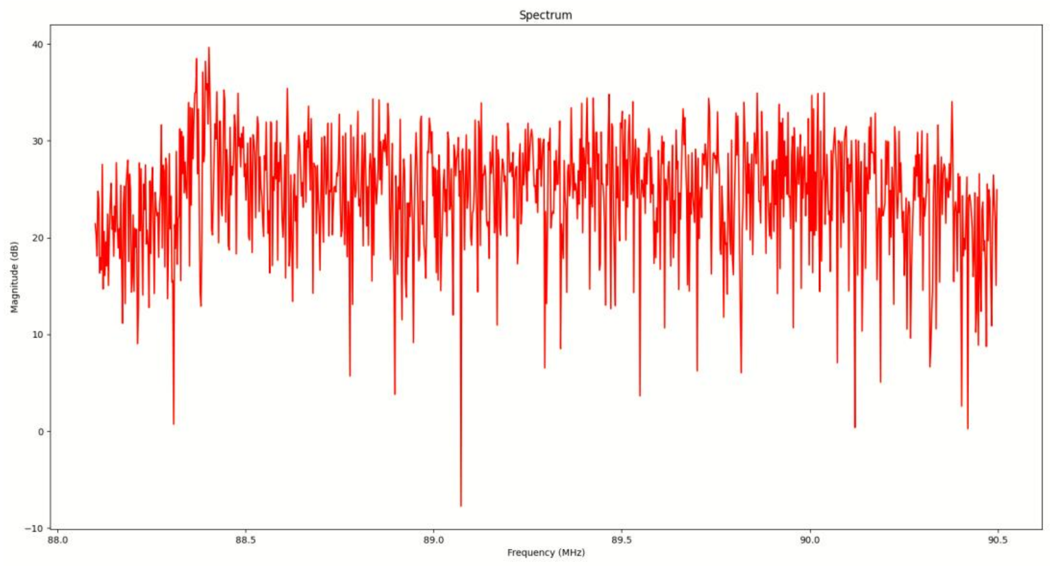


Figure 4: Magnitude spectrum of first 1024 samples

However, when the last 1024 samples of the signal were used for the FFT, the calculated spectrum (figure 5) showed a sharp peak at 89.3 MHz as expected.

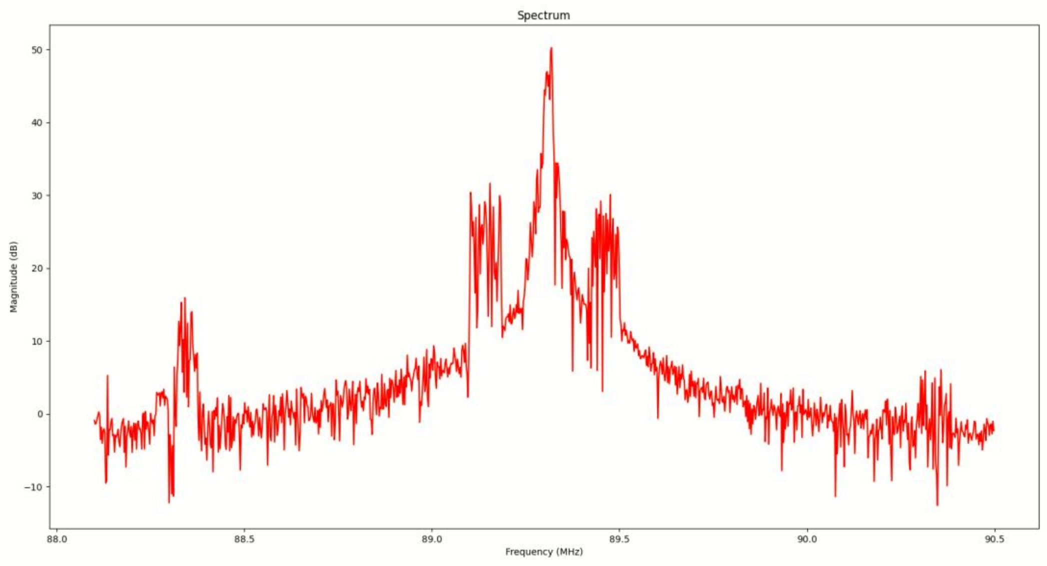


Figure 5: Magnitude spectrum of last 1024 samples

The solution to this phenomenon was simple: don't use any of the first ~2000 samples read in from the RTL-SDR for calculating the spectrum.

<u>Accounting for the Curved Frequency Response of the RTL-SDR</u>

Another problem noted was that the frequency response of the RTL-SDR is not flat over its tuning bandwidth. Frequencies towards the edge of its bandwidth are attenuated. A plot of a single 2.4 MHz band with no transmissions (i.e., only noise) is depicted in figure 6 below, with the rolloff at the band edges circled.

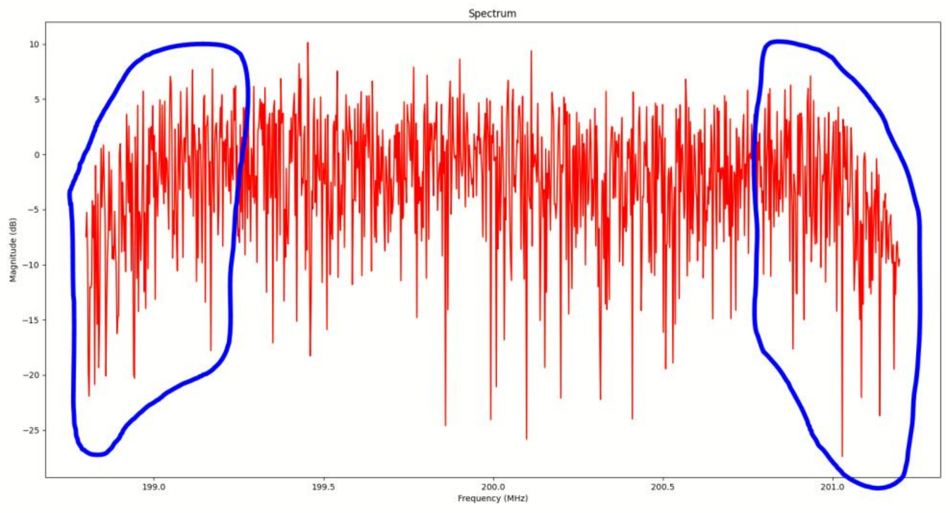


Figure 6: RTL-SDR frequency response rolloff at band edges

This issue was mitigated by discarding the FFT points for frequencies near the edge of the RTL-SDRs tuning bandwidth. For example, if the RTL-SDR was set to a tuning bandwidth of 2.4 MHz (± 1.2 MHz from center frequency), only frequencies ±0.6 MHz from center were considered for the PSD plot.

This also meant that if the current center frequency was f_c, the next center frequency had to be $f_c + 1.2\ \mathrm{MHz}$ instead of $f_c + 2.4\ \mathrm{MHz}$.

Smoothing Out Noise in Power Spectral Density Plots

It was quickly discovered that power spectral density plots using the second method (FFT in each band) were very noisy. An example is shown in figure 7 below for the UHF television band (channels 14-36, 470-608 MHz).

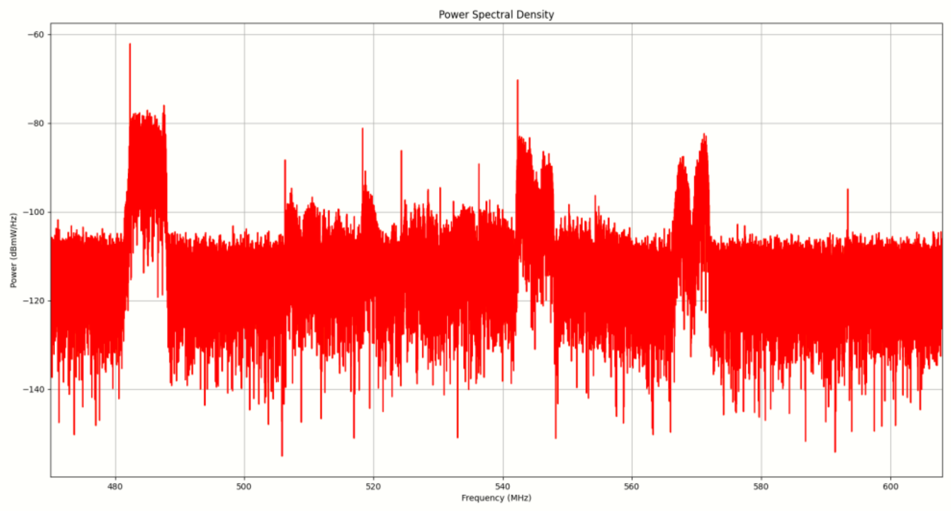


Figure 7: PSD plot for 470 – 608 MHz

One solution envisioned for this was to average the PSD plot over multiple scans. However, this did not work well. Figure 8 shows the result of averaging the PSD over 100 scans. Not only did this take forever, but the PSD plot looked just as noisy. Additionally, spectral impulses occurred at each of the tuning frequencies, as shown below. These spectral impulses indicate an unwanted DC offset in the tuned signal. Why this did not occur when averaging was not used, however, is a mystery.

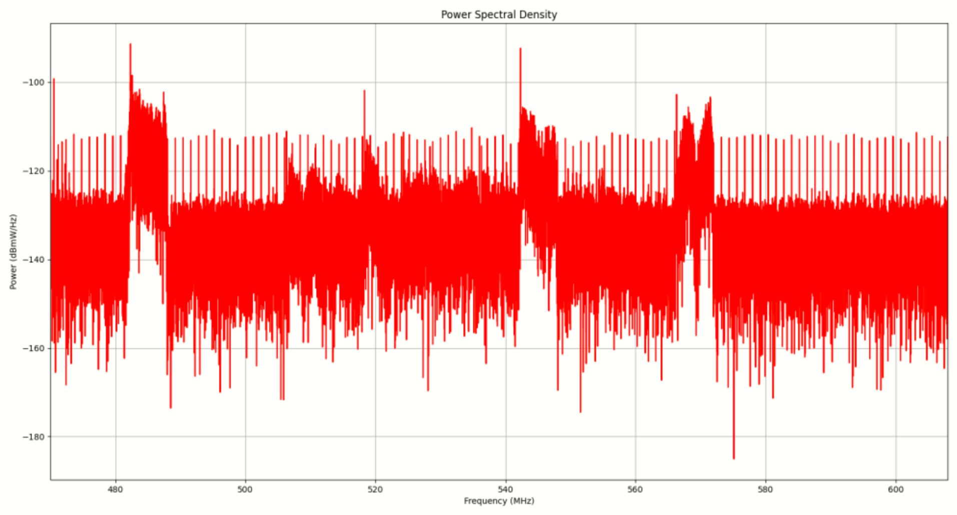


Figure 8: PSD plot for 470 – 608 MHz averaged over 100 trials

Subsequently, a smoothing filter was tested instead to deal with the noise. A Savitzky–Golay filter in the SciPy package was used for smoothing the data [2]. When the filter length was set appropriately, the noise in the PSD was greatly reduced, as shown in figure 9 below.

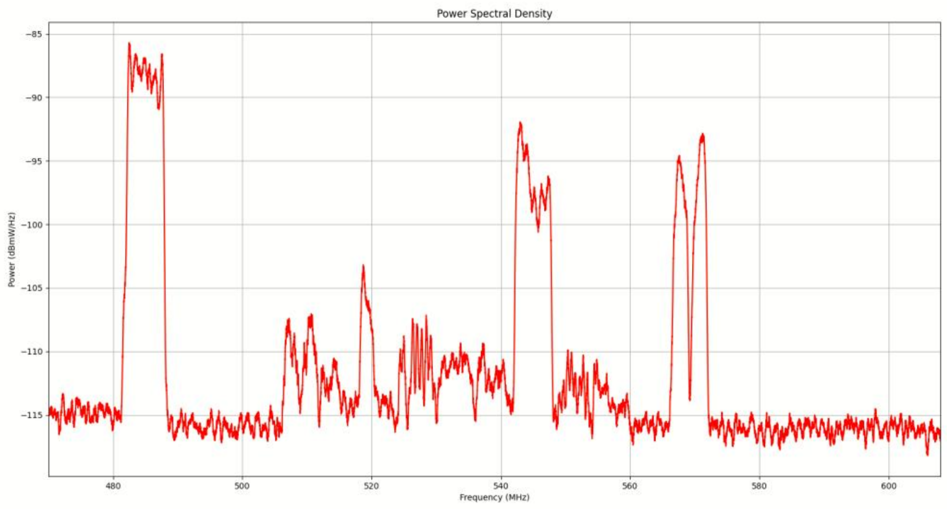
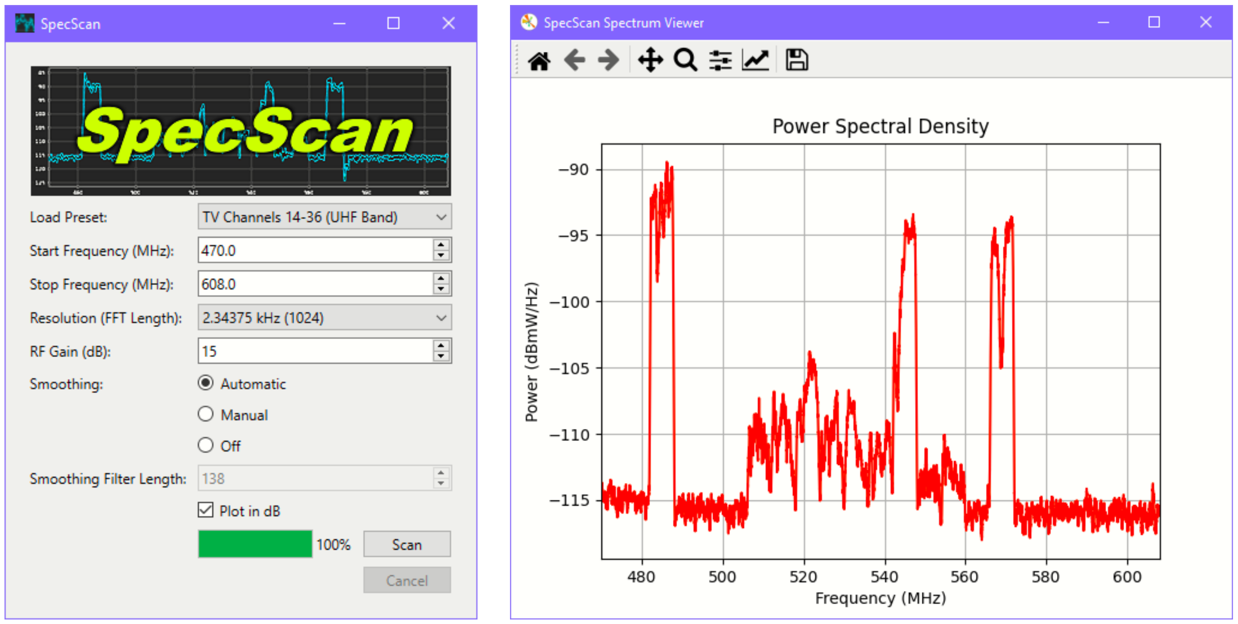


Figure 9: PSD plot for 470 – 608 MHz smoothed with Savitzky-Golay filter

Final Design

The final design incorporated all the modifications listed above. The GUI was created using PyQt. Various options were added for the user to customize their scan. Matplotlib was used to display spectrum plots, as shown in the "SpecScan Spectrum Viewer" window in figure 10 below.



Load Preset: TV Channels 14-36 (UHF Band)
Start Frequency (MHz): 470.0
Stop Frequency (MHz): 608.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: [✓ Automatic] Manual Off
Smoothing Filter Length: 138
[✓ Plot in dB]
100%

Figure 10: Final GUI

Preliminary Design Verification

Reading Samples from the RTL-SDR

The first step in this project was evaluating the feasibility of using Python for reading samples from the RTL-SDR. Installing all the dependences in the proper locations proved to be a bit challenging. However, once this was done I was able to read in complex samples from the RTL-SDR and plot them, as shown in the figure 11.

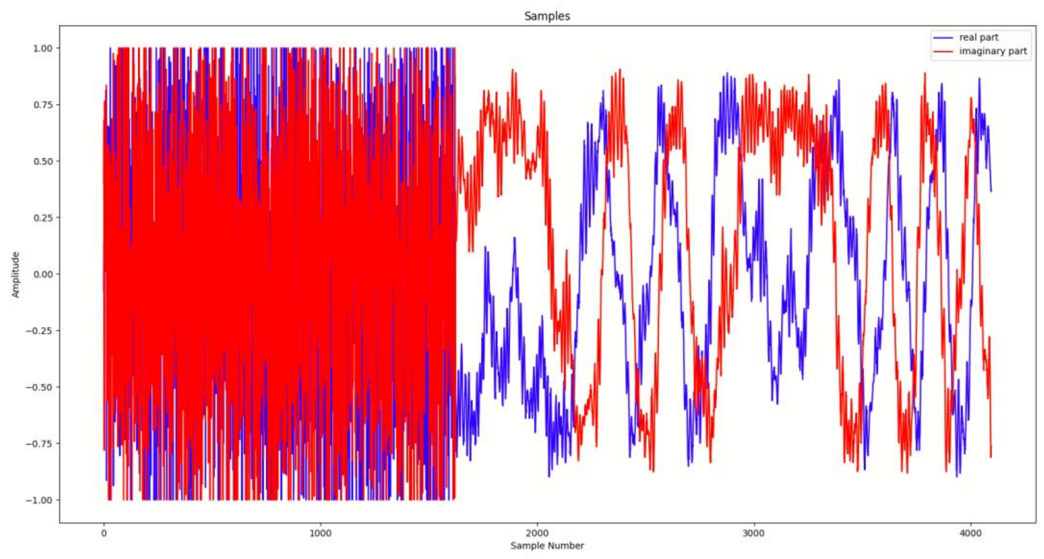


Figure 11: Example of 4096 complex samples read in from the RTL-SDR

Calculating the Power Spectral Density from Samples

The next part of the verification phase was to verify generating a power spectral density plot from the samples collected. As shown above, the first ~1700 samples contained only noise each time samples were read in from the RTL-SDR. This was not an expected result! Thus, these samples were not used for calculating the power spectral density.

To calculate the power spectral density in a frequency band, NumPy’s **`fft`** function was used to calculate the spectrum of a subset of the samples that were not noise. The **`fftshift`** function was to shift the spectrum around the center frequency the RTL-SDR was tuned to. The power spectral density was then calculated by taking the magnitude of the spectrum and squaring it. Figure 12 shows the power spectral density calculated for a 1.2 MHz frequency band, centered at 89.3 MHz, the frequency of nearby WQED-FM.

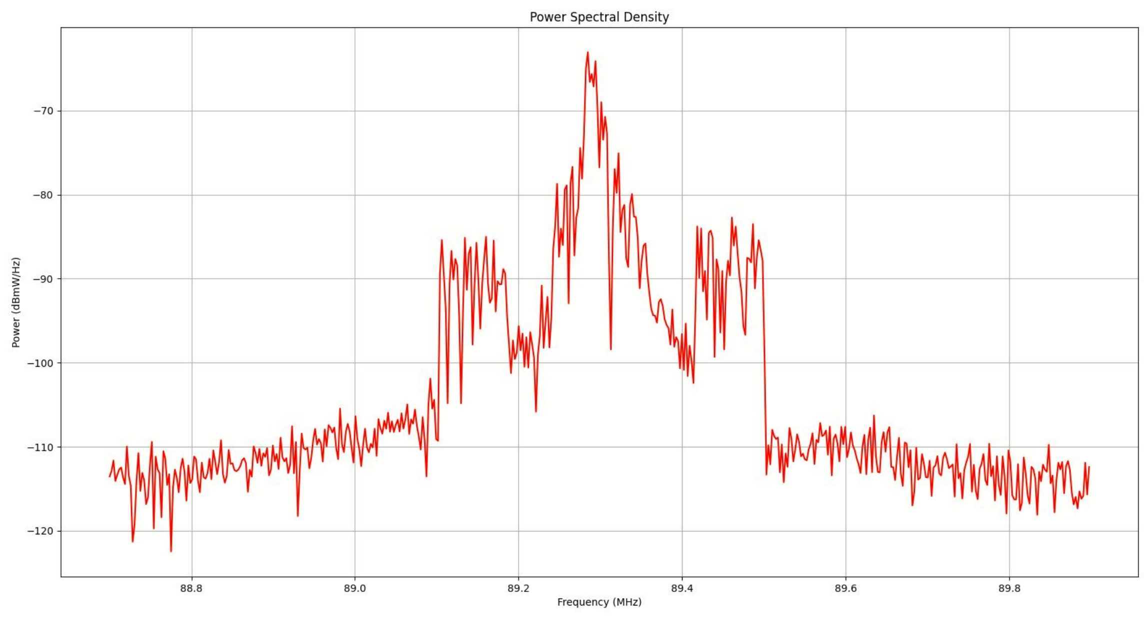


Figure 12: PSD plot for single 1.2 MHz bandwidth, centered at 89.3 MHz

Concatenating PSD Arrays

The next step of the verification process was to generate a power spectral density plot for a wider band by consecutively setting the RTL-SDR to adjacent frequency bands, calculating the PSD for each band, and concatenating these PSD arrays together to create one large PSD array. An example of this for the entire FM band (~88 – 108 MHz) is shown in figure 13 below.

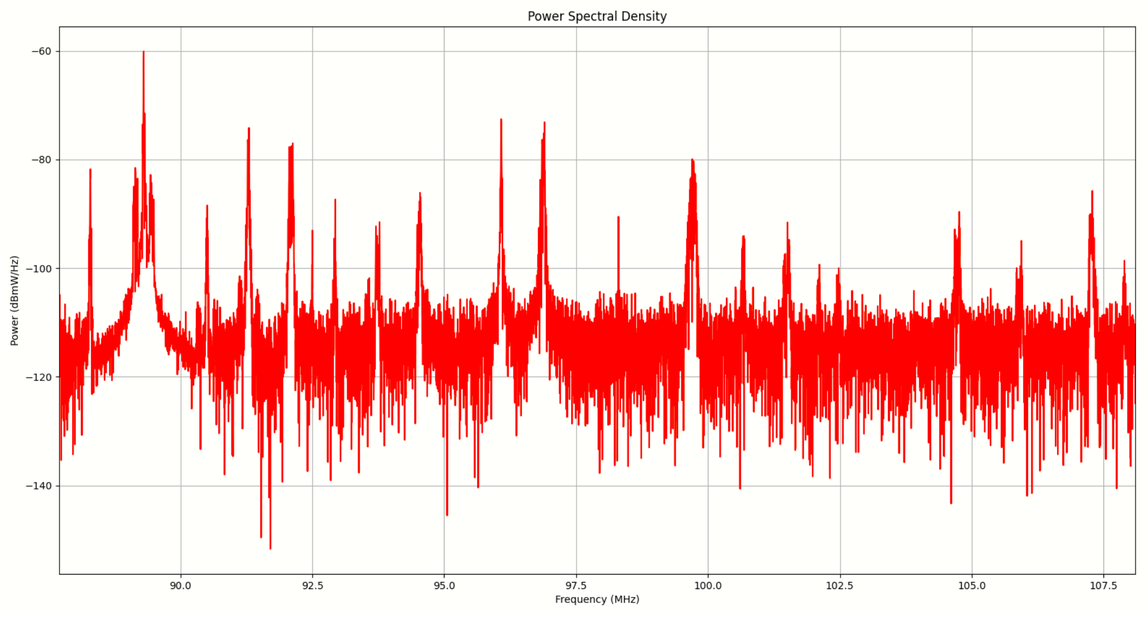
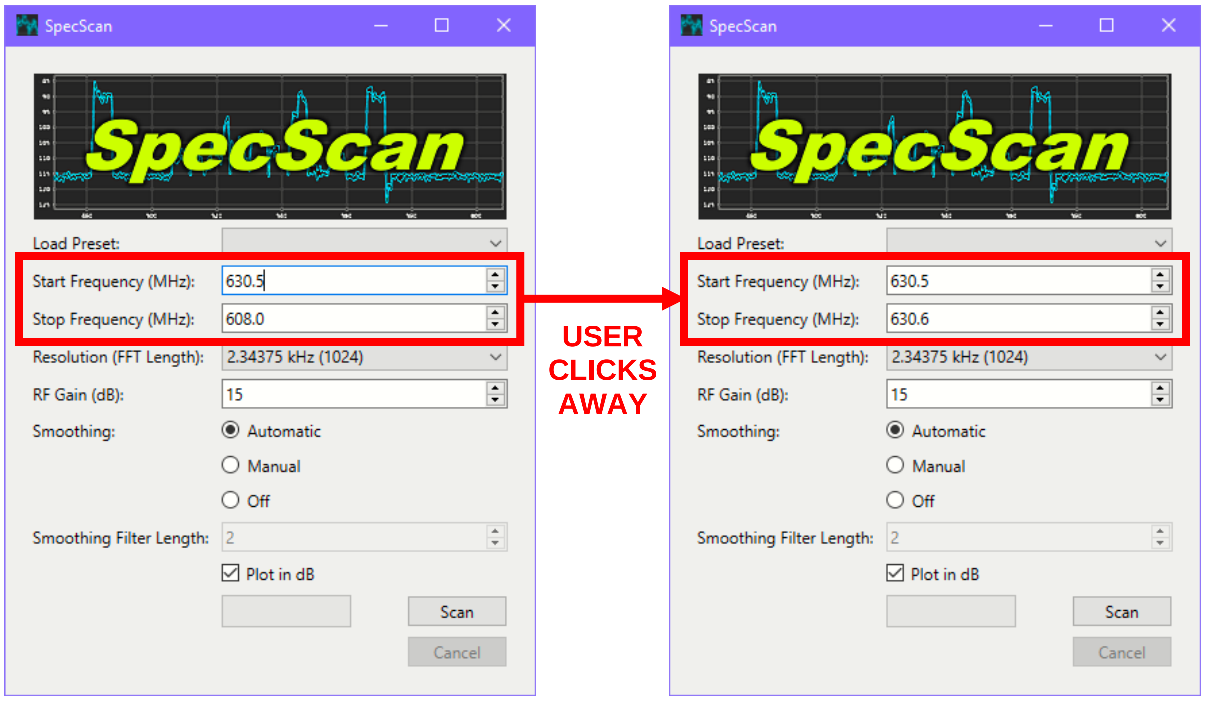


Figure 13: Concatenated PSD plot for entire FM band (~20 MHz wide)

User Interface Design

The final part of the design verification process was to create a graphical user interface (GUI). PyQt was selected as the platform to use for the GUI. A tutorial series on YouTube was first followed to learn the basics of the latest version of PyQt, PyQt6 (https://www.youtube.com/playlist?list=PLHwXkLexR9MAm0UZsX0SGFOSNVlCYLJxx).

From these videos I learned how to place and position "widgets" like entry boxes, dropdown menus, buttons, etc. I also learned to connect actions in the GUI (e.g., clicking a button) to code routines. The main routine for this program is scanning the spectrum when the user clicks the "Scan" button. However, several other routines were created to make the program more robust. For example, if a user enters a start frequency that is greater than the stop frequency, the stop frequency is nudged a step above the start frequency as soon as the user clicks away.



	Before	After
Load Preset:		
Start Frequency (MHz):	630.5	630.5
Stop Frequency (MHz):	608.0	630.6
Resolution (FFT Length):	2.34375 kHz (1024)	2.34375 kHz (1024)
RF Gain (dB):	15	15
Smoothing:	[✓ Automatic] Manual Off	[✓ Automatic] Manual Off
Smoothing Filter Length:	2	2
	[✓ Plot in dB]	[✓ Plot in dB]

Figure 14: Example of Program Automatically Correcting Invalid User Input

Design Implementation

Spectrum Scanning Algorithm

The most important part of this design is the spectrum scanning algorithm. Although it was described in the overview section with rough pseudocode, the flowchart in figure 15 depicts this process in much greater detail. For clarity, flowchart blocks relating to the frequency are colored red and those relating to the GUI are colored yellow.

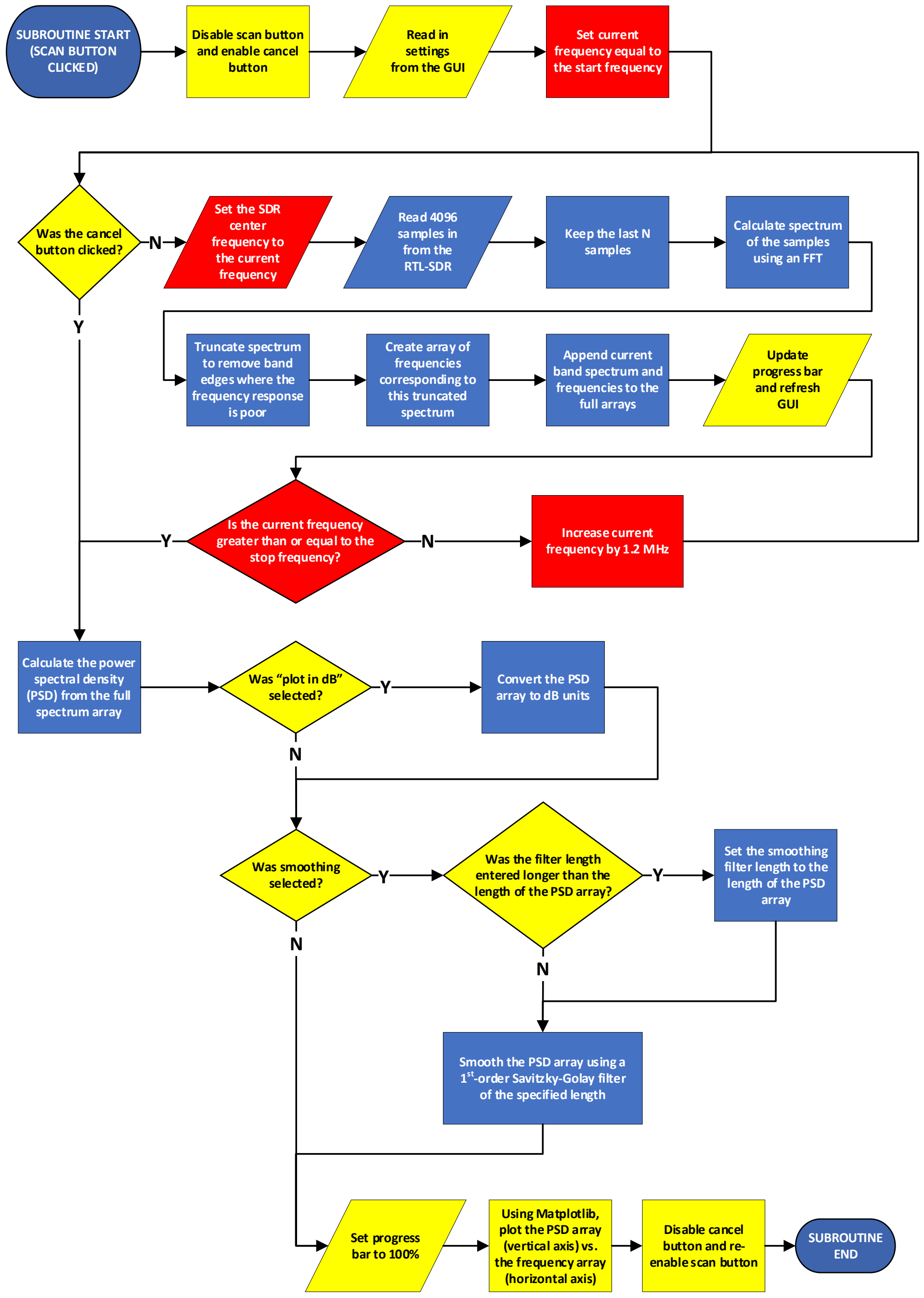
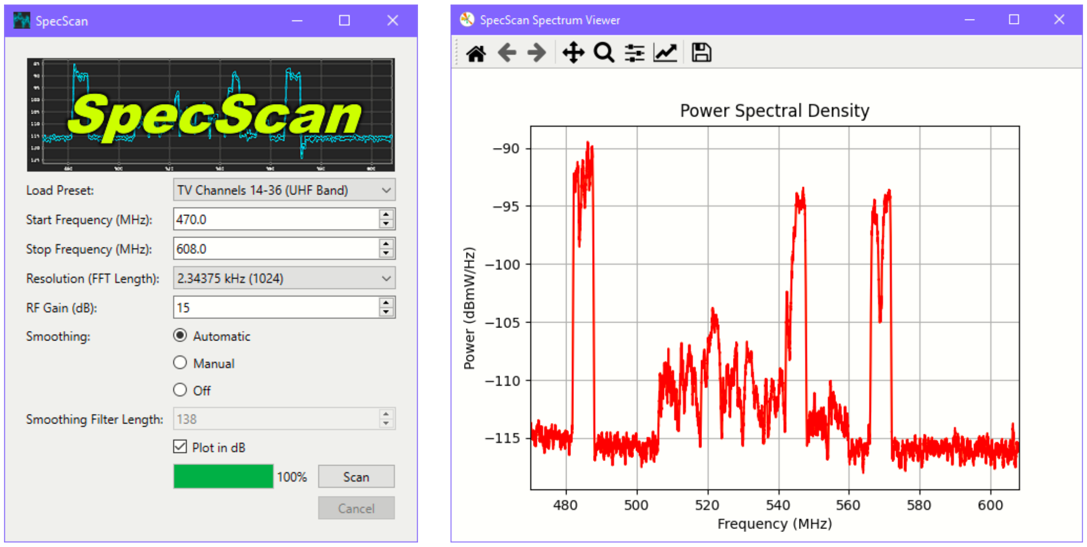

SUBROUTINE START (SCAN BUTTON CLICKED)
Disable scan button and enable cancel button
Read in settings from the GUI
Set current frequency equal to the start frequency
Was the cancel button clicked?
N
Set the SDR center frequency to the current frequency
Read 4096 samples in from the RTL-SDR
Keep the last N samples
Calculate spectrum of the samples using an FFT
Truncate spectrum to remove band edges where the frequency response is poor
Create array of frequencies corresponding to this truncated spectrum
Append current band spectrum and frequencies to the full arrays
Update progress bar and refresh GUI
Y
Is the current frequency greater than or equal to the stop frequency?
Y
N
Increase current frequency by 1.2 MHz
Calculate the power spectral density (PSD) from the full spectrum array
Was "plot in dB" selected?
Y
Convert the PSD array to dB units
N
Was smoothing selected?
Y
Was the filter length entered longer than the length of the PSD array?
Y
Set the smoothing filter length to the length of the PSD array
N
N
Smooth the PSD array using a 1st-order Savitzky-Golay filter of the specified length
Set progress bar to 100%
Using Matplotlib, plot the PSD array (vertical axis) vs. the frequency array (horizontal axis)
Disable cancel button and re-enable scan button
SUBROUTINE END

Figure 15: Flowchart depicting spectrum scanning algorithm

User Interface

The User Interface was designed using PyQt. The GUI elements (e.g., labels, drop-down menus, entry boxes, buttons, etc.) were defined and positioned on a grid. Additionally, specific user interactions with GUI elements are connected to subroutines. Most notably, clicking the "Scan" button initiates the algorithm described on the previous page. However, there were additional subroutines added.



Load Preset:	TV Channels 14-36 (UHF Band)
Start Frequency (MHz):	470.0
Stop Frequency (MHz):	608.0
Resolution (FFT Length):	2.34375 kHz (1024)
RF Gain (dB):	15
Smoothing:	[✓ Automatic] Manual Off
Smoothing Filter Length:	138
	☑ Plot in dB
	100%

Figure 16: Final GUI

Loading Presets

Whenever the selection in the "Load Preset" box is changed, a subroutine automatically changes the start and stop frequencies to those of the preset (e.g., 87.7 – 108.1 MHz for the FM Radio preset).

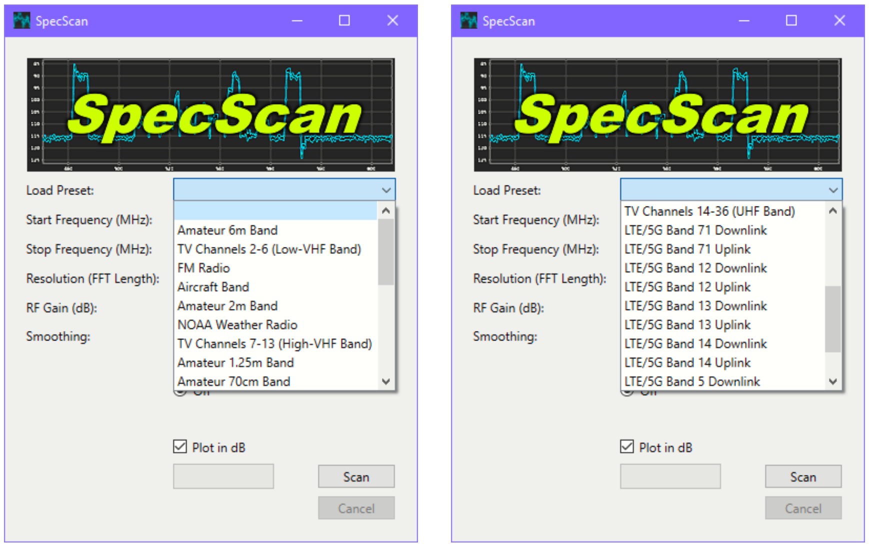


Figure 17: Some of the Presets Available

Additionally, whenever the user changes the start or stop frequency, a subroutine changes the "Load Preset" dropdown to the blank option.

Start and Stop Frequency Verification

Whenever the user finishes editing the start frequency (i.e., they click away from the box), a subroutine verifies that the stop frequency is greater than the start frequency. If this is not the case, the stop frequency is automatically adjusted to be 0.1 MHz greater than the start frequency.

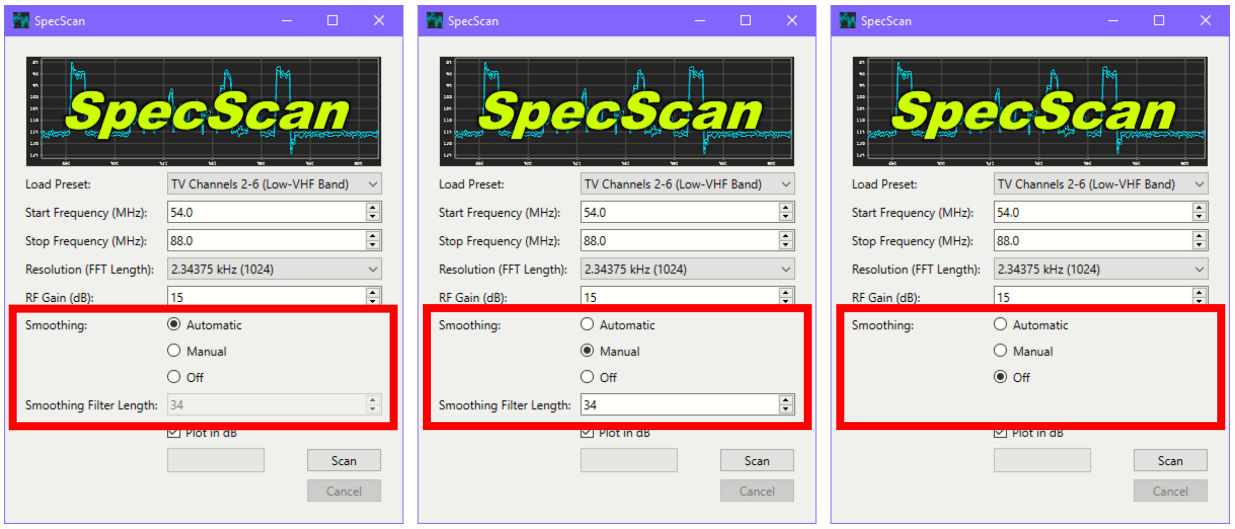
Similarly, whenever the user finishes editing the stop frequency, a subroutine verfies that the start frequency is less than the stop frequency. If this is not the case, the start frequency is automatically adjusted to be 0.1 MHz less than the stop frequency.

Automatic Smoothing Filter Length Adjustment

Additionally, if the smoothing mode is set to "Automatic", whenever the user changes the start frequency, stop frequency, or resolution, a subroutine automatically updates the filter length. The automatically determined filter length is [stop freq. in Hz – start freq. in Hz] * (FFT length) / 1024, rounded to the nearest integer. This equation was determined empirically. The smoothing filter length box is also disabled (i.e., grayed out) whenever automatic smoothing is selected, preventing the user from editing the filter length directly.

Hiding the Filter Length Box

If the user determines that they want to turn smoothing off, a subroutine hides the smoothing filter length box when the "Off" radio button is clicked.


SpecScan

Load Preset: TV Channels 2-6 (Low-VHF Band)
Start Frequency (MHz): 54.0
Stop Frequency (MHz): 88.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: [✓ Automatic] Manual Off
Smoothing Filter Length: 34
Plot in dB ✓

SpecScan

Load Preset: TV Channels 2-6 (Low-VHF Band)
Start Frequency (MHz): 54.0
Stop Frequency (MHz): 88.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: Automatic [✓ Manual] Off
Smoothing Filter Length: 34
Plot in dB ✓

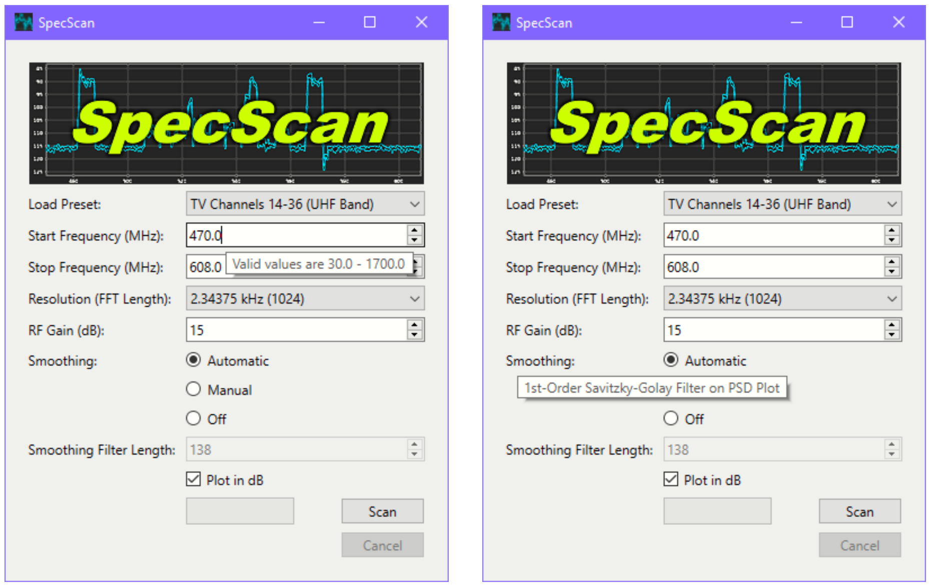
SpecScan

Load Preset: TV Channels 2-6 (Low-VHF Band)
Start Frequency (MHz): 54.0
Stop Frequency (MHz): 88.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: Automatic Manual [✓ Off]
Plot in dB ✓

Figure 18: The smoothing filter length box when different smoothing options are selected

These various GUI-related subroutines were introduced to make the software more intuitive and prevent erroneous user input from causing problems.

Additionally, tooltips were created to appear when the user mouses over an option or label in need of further explanation. Ranges were also set on the various boxes to restrict user input (e.g., frequencies only between 30.0 – 1700.0 MHz and gains only between 0 – 30 dB.)



Load Preset: TV Channels 14-36 (UHF Band)
Start Frequency (MHz): 470.0
Stop Frequency (MHz): 608.0 Valid values are 30.0 - 1700.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: [✓ Automatic] Manual Off
Smoothing Filter Length: 138
[✓ Plot in dB]

Load Preset: TV Channels 14-36 (UHF Band)
Start Frequency (MHz): 470.0
Stop Frequency (MHz): 608.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 15
Smoothing: [✓ Automatic] 1st-Order Savitzky-Golay Filter on PSD Plot Off
Smoothing Filter Length: 138
[✓ Plot in dB]

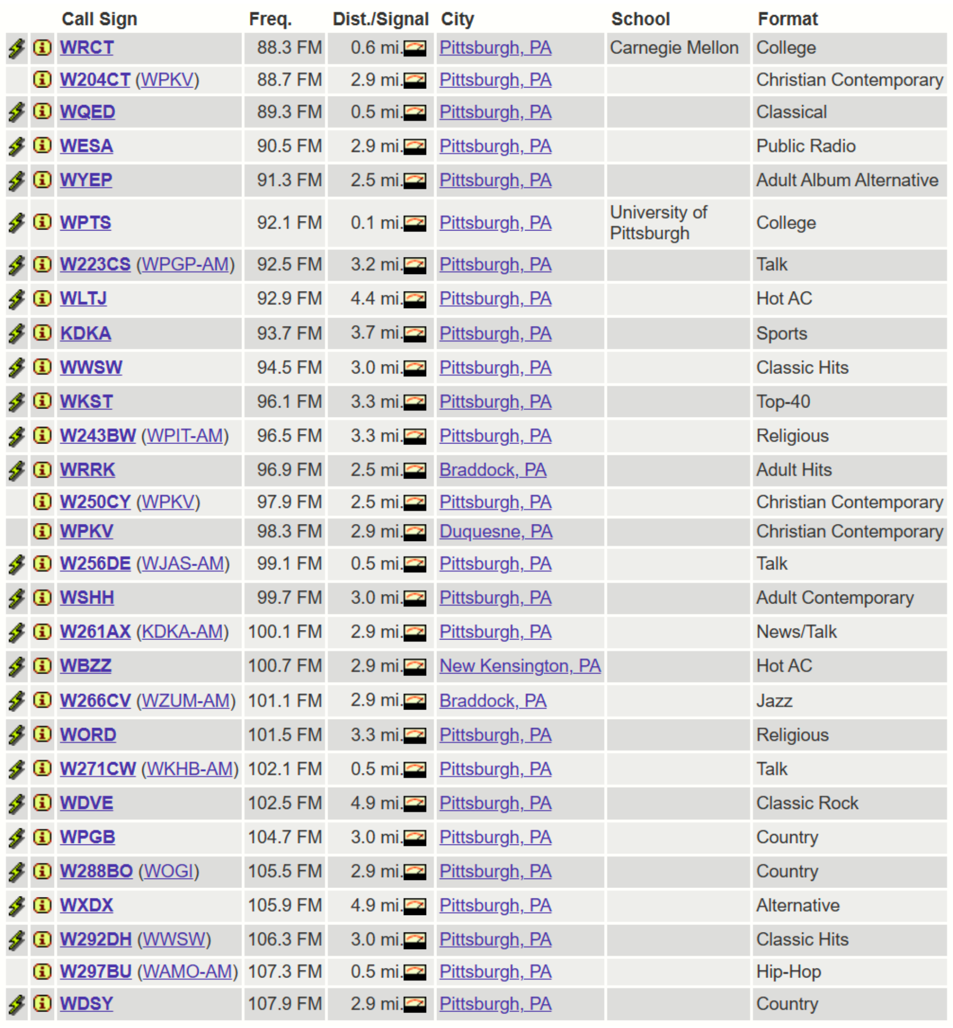
Figure 19: Examples of tooltips that appear when the user mouses over an option or label

Design Testing

Spectrum Plotting

Known broadcasts were critical for testing this program. For example, FM radio and TV stations are assigned to fixed frequencies and broadcast locations. Publicly accessible databases like Radio-Locator, RabbitEars, and the National Weather Service were consulted to verify the frequencies detected by the spectrum analyzer program [3][4][5].

For example, a list of local FM broadcasts obtained from Radio-Locator is depicted in figure 20 below, and figure 21 shows a spectrum scan with many of these identified and annotated.



		Call Sign	Freq.	Dist./Signal	City	School	Format
⚡	ⓘ	WRCT	88.3 FM	0.6 mi.	Pittsburgh, PA	Carnegie Mellon	College
	ⓘ	W204CT (WPKV)	88.7 FM	2.9 mi.	Pittsburgh, PA		Christian Contemporary
⚡	ⓘ	WQED	89.3 FM	0.5 mi.	Pittsburgh, PA		Classical
⚡	ⓘ	WESA	90.5 FM	2.9 mi.	Pittsburgh, PA		Public Radio
⚡	ⓘ	WYEP	91.3 FM	2.5 mi.	Pittsburgh, PA		Adult Album Alternative
⚡	ⓘ	WPTS	92.1 FM	0.1 mi.	Pittsburgh, PA	University of Pittsburgh	College
⚡	ⓘ	W223CS (WPGP-AM)	92.5 FM	3.2 mi.	Pittsburgh, PA		Talk
⚡	ⓘ	WLTJ	92.9 FM	4.4 mi.	Pittsburgh, PA		Hot AC
⚡	ⓘ	KDKA	93.7 FM	3.7 mi.	Pittsburgh, PA		Sports
⚡	ⓘ	WWSW	94.5 FM	3.0 mi.	Pittsburgh, PA		Classic Hits
⚡	ⓘ	WKST	96.1 FM	3.3 mi.	Pittsburgh, PA		Top-40
⚡	ⓘ	W243BW (WPIT-AM)	96.5 FM	3.3 mi.	Pittsburgh, PA		Religious
⚡	ⓘ	WRRK	96.9 FM	2.5 mi.	Braddock, PA		Adult Hits
	ⓘ	W250CY (WPKV)	97.9 FM	2.5 mi.	Pittsburgh, PA		Christian Contemporary
	ⓘ	WPKV	98.3 FM	2.9 mi.	Duquesne, PA		Christian Contemporary
⚡	ⓘ	W256DE (WJAS-AM)	99.1 FM	0.5 mi.	Pittsburgh, PA		Talk
⚡	ⓘ	WSHH	99.7 FM	3.0 mi.	Pittsburgh, PA		Adult Contemporary
⚡	ⓘ	W261AX (KDKA-AM)	100.1 FM	2.9 mi.	Pittsburgh, PA		News/Talk
⚡	ⓘ	WBZZ	100.7 FM	2.9 mi.	New Kensington, PA		Hot AC
⚡	ⓘ	W266CV (WZUM-AM)	101.1 FM	2.9 mi.	Braddock, PA		Jazz
⚡	ⓘ	WORD	101.5 FM	3.3 mi.	Pittsburgh, PA		Religious
⚡	ⓘ	W271CW (WKHB-AM)	102.1 FM	0.5 mi.	Pittsburgh, PA		Talk
⚡	ⓘ	WDVE	102.5 FM	4.9 mi.	Pittsburgh, PA		Classic Rock
⚡	ⓘ	WPGB	104.7 FM	3.0 mi.	Pittsburgh, PA		Country
⚡	ⓘ	W288BO (WOGI)	105.5 FM	2.9 mi.	Pittsburgh, PA		Country
⚡	ⓘ	WXDX	105.9 FM	4.9 mi.	Pittsburgh, PA		Alternative
⚡	ⓘ	W292DH (WWSW)	106.3 FM	3.0 mi.	Pittsburgh, PA		Classic Hits
	ⓘ	W297BU (WAMO-AM)	107.3 FM	0.5 mi.	Pittsburgh, PA		Hip-Hop
⚡	ⓘ	WDSY	107.9 FM	2.9 mi.	Pittsburgh, PA		Country

Figure 20: FM broadcasts originating from within 5 miles of zip code 15213 [3]

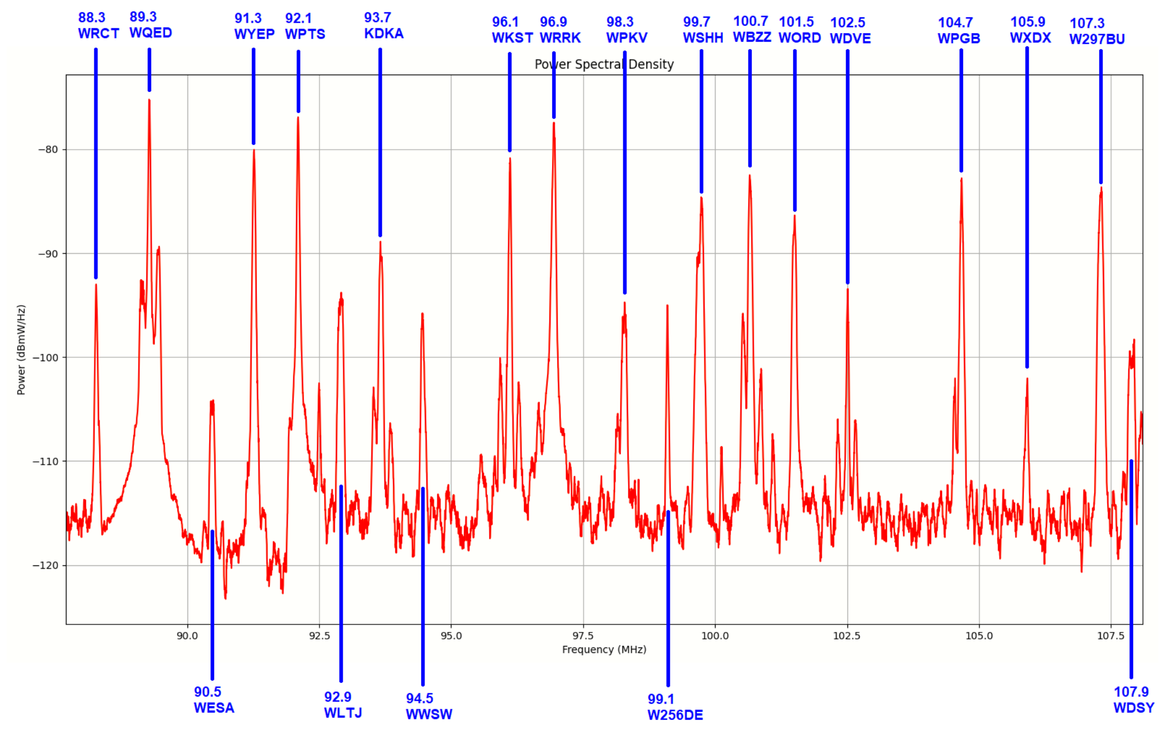


Figure 21: FM band spectrum with local broadcasts identified

Scanning was also carried out on other bands, including TV and weather radio, to verify that the program was generating accurate plots.

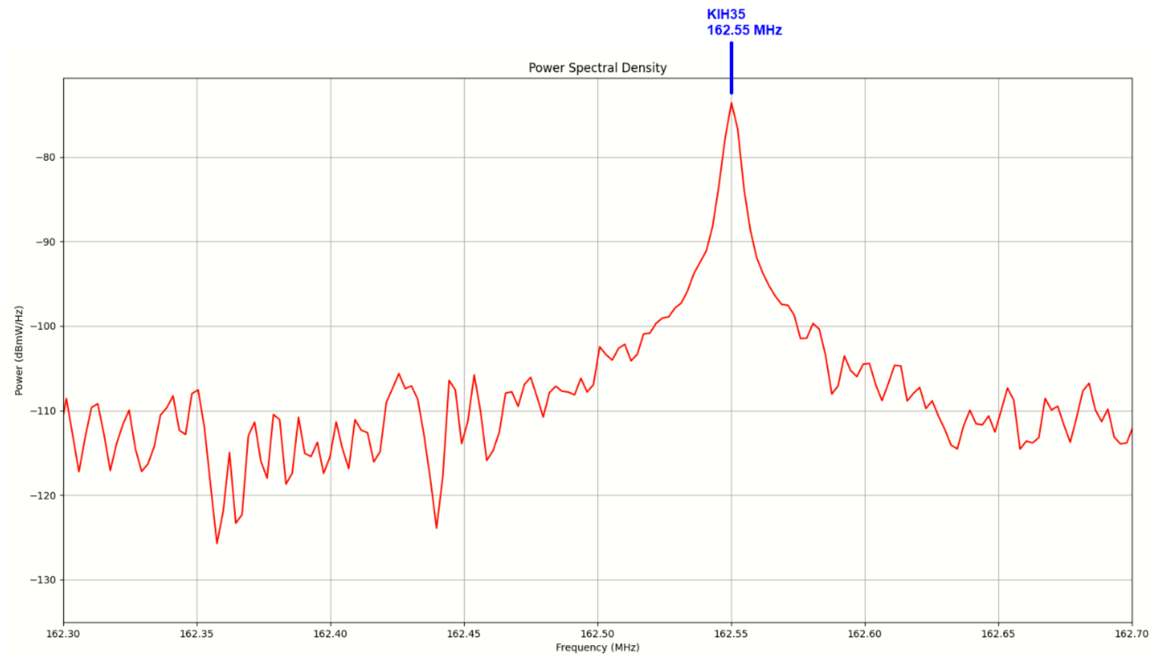


Figure 22: NOAA weather radio band

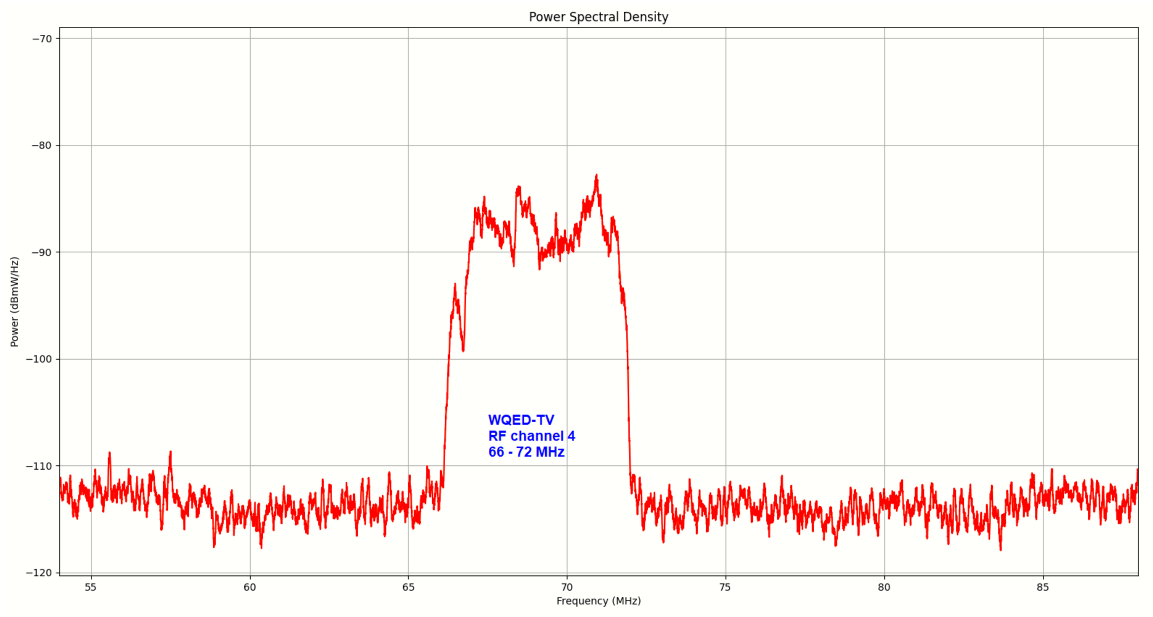


Figure 23: Low-VHF television band (channels 2-6)

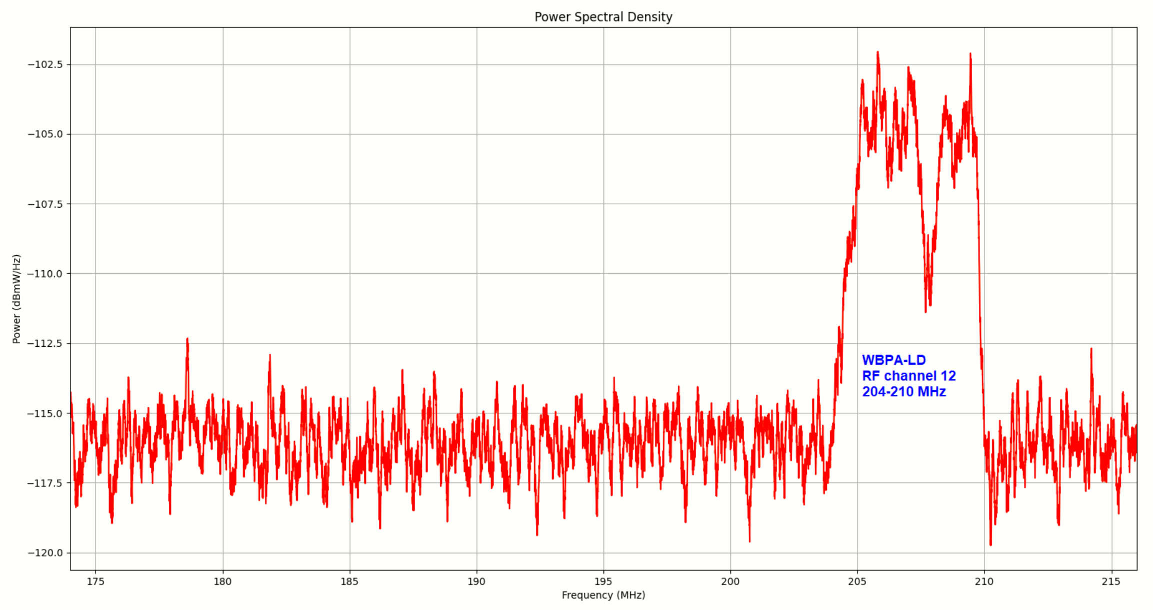


Figure 24: High-VHF television band (channels 7-13)

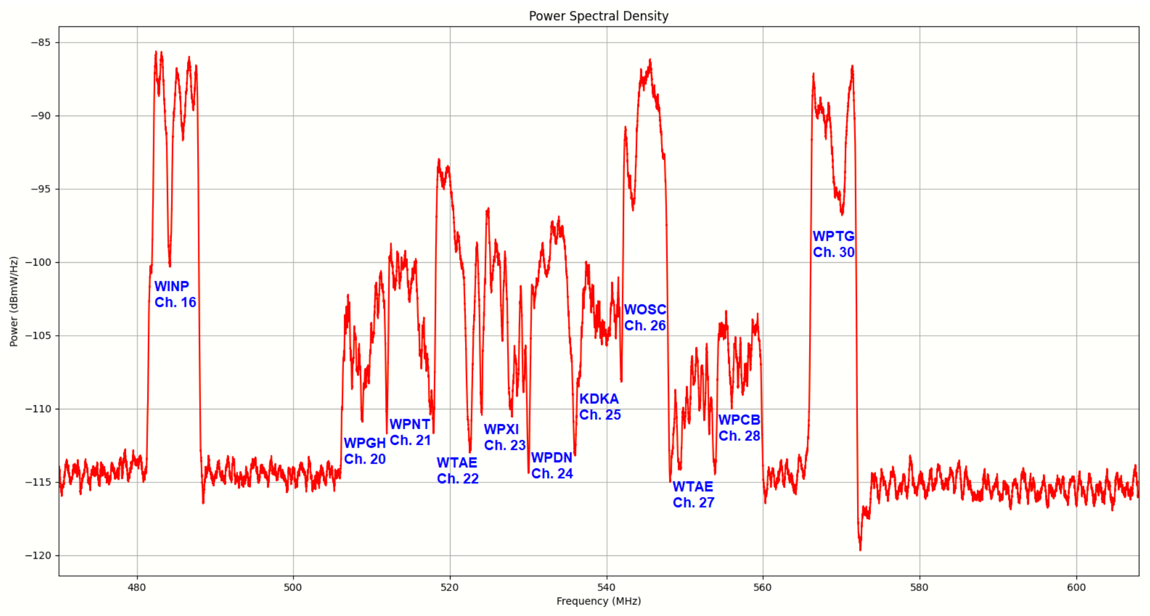


Figure 25: UHF television band (channels 14-36)

Debugging the Spectrum Plots

Although the spectrum plots in the final design are accurate, this was not initially the case. Initially the spectra were not accurate, and this was discovered to be due to the first ~1700 samples from the RTL-SDR containing only noise.

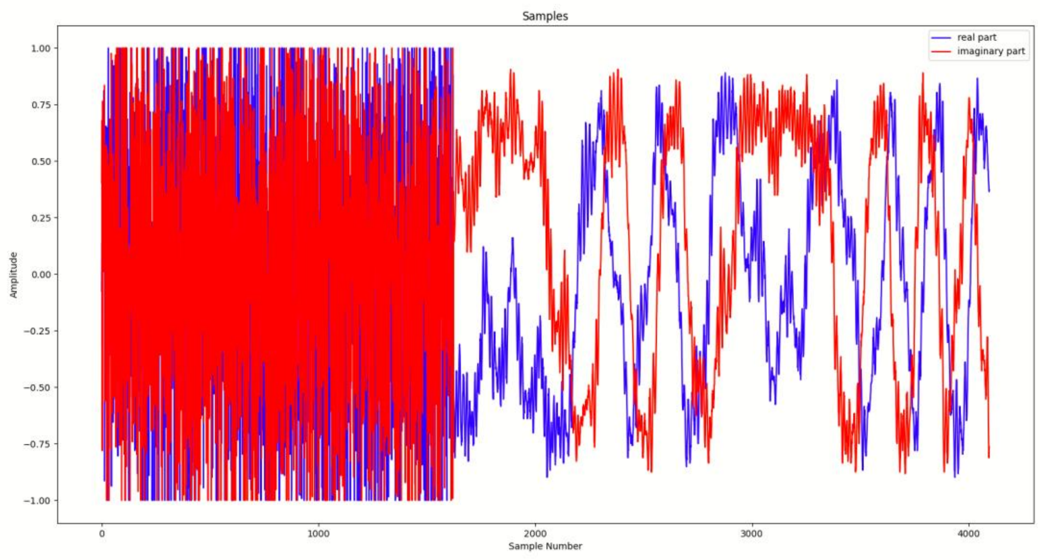


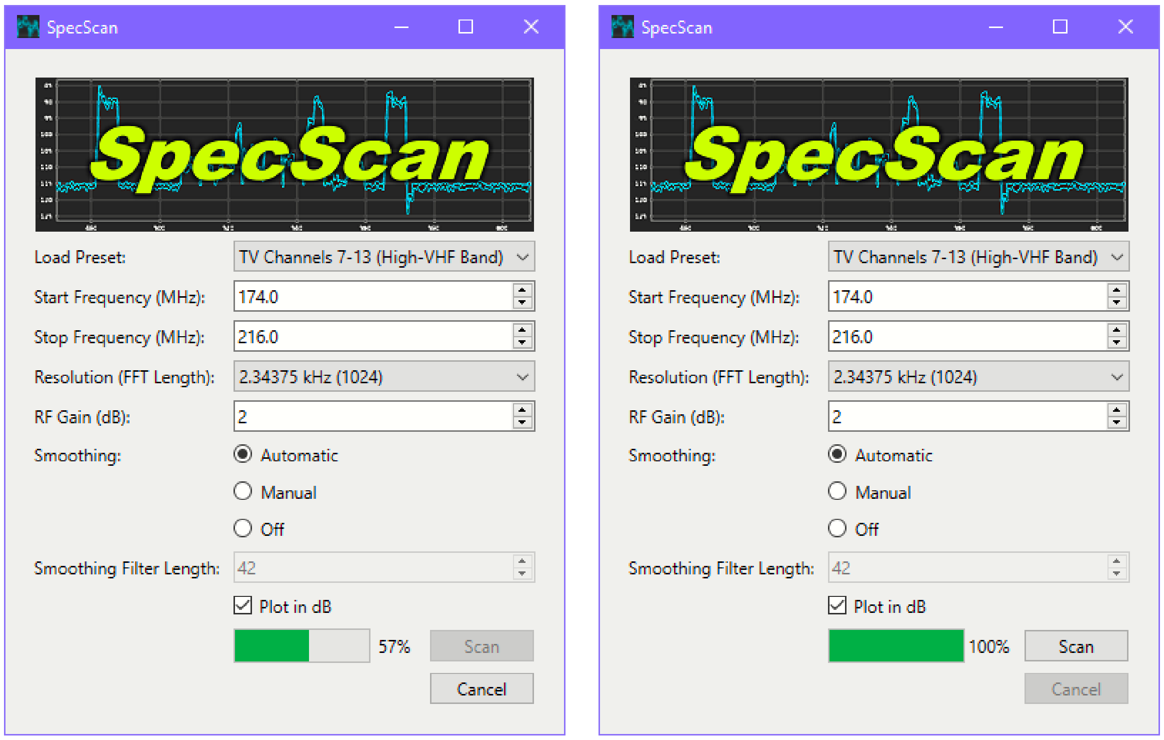
Figure 26: Example of 4096 samples read in from the RTL-SDR

This behavior from the RTL-SDR was not expected, and I originally thought it was an error in my code! As previously described in the “Changes to the Design” section of the Overview, the solution was to not include the first ~2000 samples read in from the RTL-SDR when performing an FFT to calculate the spectrum. A few other tweaks to improve the spectrum plots, including a smoothing filter, were also implemented. These changes are described in the Overview section as well.

User Interface

The user interface of this software was also thoroughly tested. The goal was to prevent the user from crashing the program. For example, initially the user could enter a start frequency that was greater than the stop frequency. If they did this and clicked "Scan", the program would crash. The fix for this was to call a subroutine that corrected the input after the user clicked away. These subroutines are described in the Design Implementation section under "Start and Stop Frequency Verification". Limits were also set on the entry boxes so that the user cannot, for example, set a frequency that is outside the tuning range the RTL-SDR (30 – 1700 MHz).

It was also found that clicking the "Scan" button while a scan was already in progress crashed the program. The solution to this was to disable the "Scan" button (gray it out) at the beginning of the scanning subroutine and re-enable it at the end of the subroutine. A cancel button was then added to allow the user to stop a scan without having to wait for it to finish.



Load Preset: TV Channels 7-13 (High-VHF Band)
Start Frequency (MHz): 174.0
Stop Frequency (MHz): 216.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 2
Smoothing: [✓ Automatic] Manual Off
Smoothing Filter Length: 42
[✓] Plot in dB
57%

Load Preset: TV Channels 7-13 (High-VHF Band)
Start Frequency (MHz): 174.0
Stop Frequency (MHz): 216.0
Resolution (FFT Length): 2.34375 kHz (1024)
RF Gain (dB): 2
Smoothing: [✓ Automatic] Manual Off
Smoothing Filter Length: 42
[✓] Plot in dB
100%

Figure 27: "Scan" and "Cancel" buttons when a scan is in progress (left) and finished (right)

Summary, Conclusions, and Future Work

In summary, the program that was created allows a user to scan and plot a desired range of radiofrequency spectrum using an RTL-SDR software defined radio tuner. The program scans a set of center frequencies, calculates the spectrum at each, and concatenates these spectra, presenting a plot of the power spectral density over the entire range the user requested. Various options, including presets of certain bands (e.g., TV, FM radio, cellular), FFT length, gain, and smoothing are presented to the user in the form of a GUI designed with PyQt.

This project greatly enhanced my understanding of software defined radio and digital signal processing in general. Although the RTL-SDR is sufficient for creating a hobbyist grade spectrum analyzer software, it is probably not sufficient for critical applications. For example, a lot of noise was observed in spectrum plots. Additionally, the frequency response and calibration of the RTL-SDR over its tuning bandwidth is unknown.

If this design were to be continued, in the next iteration I would add a continuous scan mode where the spectrum would be scanned continuously, and the plot automatically updated. This feature would allow the user to more easily detect transient transmissions, such as those that occur in amateur radio bands.

References

[1] About RTL-SDR. *RTL-SDR.COM*. Retrieved April 16, 2023, from https://www.rtl-sdr.com/about-rtl-sdr/

[2] Savitzky-Golay Filter. *SciPy v1.10.1 Manual*. Retrieved April 16, 2023, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html

[3] Radio Stations in 15213. *Radio-Locator*. Retrieved April 16, 2023, from https://radio-locator.com/cgi-bin/locate?select=city&city=15213&state=PA&band=Both&is_lic=Y&is_cp=Y&is_fl=Y&is_fx=Y&is_fb=Y&format=&dx=3&radius=5&freq=&sort=freq

[4] Digital TV Market Listings. *RabbitEars*. Retrieved April 16, 2023, from https://rabbitears.info/market.php?mktid=29

[5] NWR Pennsylvania Station Listing. *National Weather Service*. Retrieved April 16, 2023, from https://www.weather.gov/nwr/stations?State=PA