Gisma University of Applied Sciences

Gisma University of Applied Sciences

- - - Business Project in Computer Science (M608) - - -

‘’ Low-Cost SDR-Based Wireless Data Analysis System for IoT and Smart Energy Applications ‘’

Batuhan Öztürk - GH1031500

1-INTRODUCTION

The wireless communication systems are to such an extent that they have become the backbone of modern technologies like the IoT and Smart Energy applications, where the bits and bytes are going through the air amid non-ideal conditions. The factor of noise, the interference, and the environment itself may lower the quality of the signal, which plainly affects the reliability of the received data.

This whole situation is very much a problem in the Smart Energy systems as the whole system of operation is based on the accurate data coming from the remote sensors and smart meters. Thus, it becomes a must to have a clear picture of how noise influences the quality of the signal, and where to draw the line for communication to still be reliable.

Software-Defined Radio (SDR) is the technology that allows for the maximum flexibility and cost-effectiveness in wireless signal analysis as the processing moves from hardware to software. Coupled with various DSP techniques such as filtering, spectral analysis, and noise reduction, SDR presents not only a qualitative but also a quantitative measurement of the signal performance under realistic conditions.

The project is about constructing a wireless SDR-based system that is not expensive and is enabled for analyzing the noise impact and the signal-to-noise ratio (SNR) on signal quality. The processing and evaluation of FM signals are done through a software-defined pipeline using GNU Radio and Python. The emphasis on algorithmic processing, data analysis, and performance evaluation categorizes this research project as a Computer Science Business Project.

2-PROJECT GOAL

The project's primary objective is to create and assess a cost-effective, software-defined wireless signal processing system, alongside a study of noise's influence on signal quality in practical communication scenarios. A reliable wireless data transmission, especially in case of non-ideal channel conditions, has become a major concern with the increasing adoption of IoT and Smart Energy systems.

The project that SDR can act as a flexible and economical substitute to conventional hardware-based radio receivers. Most of the system functionalities have been moved to

software, and one SDR device is being used to demonstrate that very complex signal processing can be done without the aid of proprietary hardware.

The project has an additional significant point, and it is that noise and Signal-to-Noise Ratio (SNR) would be analyzed in terms of their effects on the quality of the received signal. Wireless systems in the real world are rarely operated at perfect conditions hence, different noise levels are looked at to see how much the signal quality deteriorates and at what point communication gets unreliable.

The project is not only about developing an operational SDR-based receiver but also performing quantitative performance evaluations. Different signal processing techniques ranging from simple filtering to spectral-based noise reduction are being applied, and their effectiveness under varying noise conditions is being compared.

In short, the project joins together the software system design, algorithmic signal processing, and data-driven evaluation, thus satisfying the requirements of a Business Project in Computer Science. The results obtained are studied in the context of IoT and Smart Energy applications, where the communication is reliable, and thus, monitoring, control, and decision-making become possible.

3- TECHNICAL BACKGROUND

This part of the project discusses the theories that lay the foundation for understanding the system built in this project. Deep math is not the focus here; rather, the core ideas are being explained in a straightforward and practical manner. The intention is to give such an extensive background that the person reading it with no prior knowledge of SDR or signal processing will be able to follow the steps of implementation and analysis given later in the report.

3.1 Software-Defined Radio (SDR)

Conventional radio systems are generally constructed out of static hardware components, with the filtering, modulation, and demodulation of the signals done through dedicated hardware directly. Systems like that can be proficient but they are also very rigid since every change usually means that the hardware has to be altered physically.

The Software-Defined Radio (SDR) system is the total opposite of this. The received radio frequency signal is first digitized by the SDR systems hardware in other words the signal is converted from analog to digital form. After the digitization, the signals are processed mainly by software which is running on a computer. This means that the same hardware can be used for different applications just by changing the software.

SDR brings a plethora of benefits; it is flexible, it shortens the time to market, and it cuts down costs. The low cost, flexibility, and quick development time advantages are among the main reasons why an SDR is the best choice for experimental research projects and any scholarly work. SDR is chosen in this project so as to have complete control over the signal processing chain and also to carry out a thorough analysis of the signal's behavior under varying noise conditions.

3.2 FM Signal Basics

Frequency Modulation (FM) is an extremely popular modulation method, especially in radio broadcast systems. In FM, the carrier signal’s frequency is altered according to the information, while its amplitude remains mostly unchanged. This feature of FM signals makes them less susceptible to noise in certain situations than modulation schemes based on amplitude.

The FM broadcast band usually works within the frequency range of about 88 to 108 MHz. These signals are powerful, readily available, and suitable for experiments; thus, FM is the preferred choice of signals in SDR-based projects.

An FM signal is the wireless signal that stands for the whole project. It has been chosen because it is a practical and easy way to study the influence of noise, interference, and distortion in wireless communication, which is the final application of IoT and Smart Energy.

The FM signal demodulation step enables the system to change the captured radio frequency into an audio signal, which can then be processed with conventional signal processing methods.

3.3 DSP Building Blocks

Digital Signal Processing (DSP) is the term that applies to the use of algorithms on digital signals for analysis and modification. In SDR systems, DSP is the most important part of the process that gives the raw digital samples their real significance. The following are some of the major DSP building blocks that are used in this project:

Filtering:

By removing the unwanted frequency components from a signal, filters improve the signal's overall quality. Filtering is a very important part of a radio receiver that helps to enhance the desired signal and to keep it from mixing together with signals from nearby frequencies.

Resampling:

Resampling changes the rate at which a signal is sampled. This operation is done when the processing stages have different sample rate requirements, for example, when high-rate radio samples are converted into lower-rate audio signals.

Demodulation:

Demodulation means getting the original information back from a modulated carrier signal. FM demodulation is the method chosen in this work to extract the audio signal from the FM broadcast.

Spectral Analysis:

Among various approaches for the signal energy distribution assessment, Fourier Transform based techniques are one of the most commonly utilized. These methods are instrumental in keeping track of noise and signal quality.

Signal processing chain is the series of these blocks done one after another, each block doing its job, and together they form the engineering side of SDR receiver in this project.

3.4 Noise and Signal to Noise Ratio

Noise is generally present in every communication system and cannot be avoided. Noise can be electronic, it can be environmental interference or even the operation of other transmitters in the same frequency band. Noise reduces the signal or reception quality, and hence it affects communication reliability.

The SNR is perhaps the most widely used method to measure the effect of noise on the signal. It is the ratio of the power of the wanted signal to the power of the noise, usually expressed in decibels (dB). Therefore, a high SNR value means a good signal, while SNR that is low indicates the presence of noise to a degree that it is similar to or exceeds the signal.

In fact, in wireless communication systems, data corruption, information loss and other such errors are main issues caused by low SNR. This is particularly true for Smart Energy applications where accuracy in data is crucial.Therefore the SNR serves as the main performance metric in this project, enabling the evaluation and comparison of different signal processing techniques effectively.

4-SYSTEM DESIGN (GNU RADIO PIPELINE)

The following section gives a detailed description of the overall system architecture and the signal processing pipeline made with GNU Radio. The emphasis of this report is on the system design and the processing of the received radio signal in steps, starting from raw radio samples and leading to an audio output that is further analysable.

The system is built on software-defined architecture principles whereby all or almost all signal processing is done in software. The SDR hardware is only used for reception and digitization of the signals, the filtering process, resampling, demodulation, and output production being performed in the GNU Radio environment. This setup provides the system with the root of flexibility and ease of changeability which is a major advantage for any project dealing with experiments and analyses.

From a high standpoint, the system can be broken down into three primary phases:

1. The SDR device takes the signals

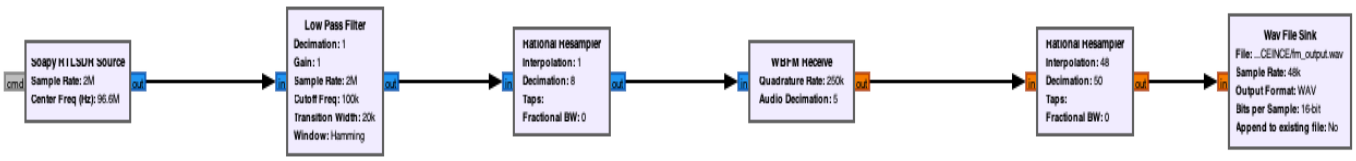
2. GNU Radio performs digital signal processing

3. Generation of audio output for further analysis

OVERALL PIPELINE CONCEPT

Initially, the SDR equipment picks up the radio frequency signal coming in and then it is transformed into complex baseband samples . The amplitude and phase of the signal are preserved in these samples, and they are the primal representation of the received signal. At this point, the signal is still contaminated with unwanted frequency components and noise.

The data is fed into GNU Radio through a processing pipeline. The flowgraph consists of different blocks performing specific tasks and data moves from one block to another in a sequential manner. The flow based programming paradigm makes it easier to design the system in a neat and modular way.



RTL SDR SOURCE

The first stage of the pipeline is the RTL-SDR Source. This block takes charge of receiving the radio signal from the SDR hardware. The block configures the center frequency and sampling rate, which are key parameters that identify the segment of the radio spectrum being captured and the coarseness of the signal sampling.

For this project, the center frequency is set within the FM broadcast band so that the system can pick up a signal that is strong and easy to get. The sampling rate is determined to be high enough to allow the required bandwidth to be captured while still being quick enough for real-time processing.

The output from this block consists of a continuous flow of complex IQ samples, which will be the input for the next processing stages.

LOW PASS FILTER

Next, after signal acquisition, the Low Pass Filter is applied to the IQ data. The filtering here is done to eliminate the frequency components that are not of interest and are thus undesired. This helps in diminishing the interference coming from adjacent channels and in controlling the noise that gets into the rest of the system.

At this point, filtering is very crucial because it not only enhances the overall signal quality but also prevents the processing of unnecessary frequency components thus reducing the workload. The filter specifications are set according to the FM signal bandwidth and the sampling rate employed by the SDR source.

RATIONAL RESAMPLER

The Rational Resampler block serves the purpose of changing the signal's sampling rate. Different sample rates are often needed in different processing stages in an SDR system. For instance, the radio frequency processing uses higher sample rates compared to the audio processing.

Signal resampling allows the system to have block compatibility and also to lessen the computational load. This step is particularly crucial before the demodulation process since it makes the signal perdurable and prompts fast and good processing.

WBFM RECEIVE

The WBFM Receive block is the one that really does the FM demodulation. It is the block's task to recover the audio from the frequency-modulated carrier. This block also provides sound that is modulated in such a way as to be intelligible, which signifies the content of the original broadcast.

FM demodulation marks the receipt of radio signal in a manner easy for the analyzer. After this step, the signal is no longer in the radio-frequency representation format and has already shifted to the audio-domain representation format.

AUDIO SINK AND WAV FILE SINK

In the last stage of the procedure, the audio signal that has been demodulated is directed to one or more output blocks. The Audio Sink provides listening in real-time, which is an advantage since it is then possible to check the receiver's proper functioning. At the same time, a WAV File Sink saves the audio signal as a file.

One of the reasons for saving the audio output is that it makes it possible to analyze it offline, using Python for instance. The fact that there are two distinct processing paths, one for real-time reception and the other for offline processing, makes the system more versatile and also aids in conducting a more detailed performance evaluation in the latter stages of the project.

5- IMPLEMENTATION

5.1 SETUP AND HARDWARE VERIFICATION

During the first week, the main activities of the project were those of setting up the hardware and software environment, overtly confirming the perfect functioning of the SDR system. During this stage, a cheap USB-based SDR unit called NooElec NESDR SMArt was utilized alongside a regular wideband antenna that is capable of receiving FM broadcasts.

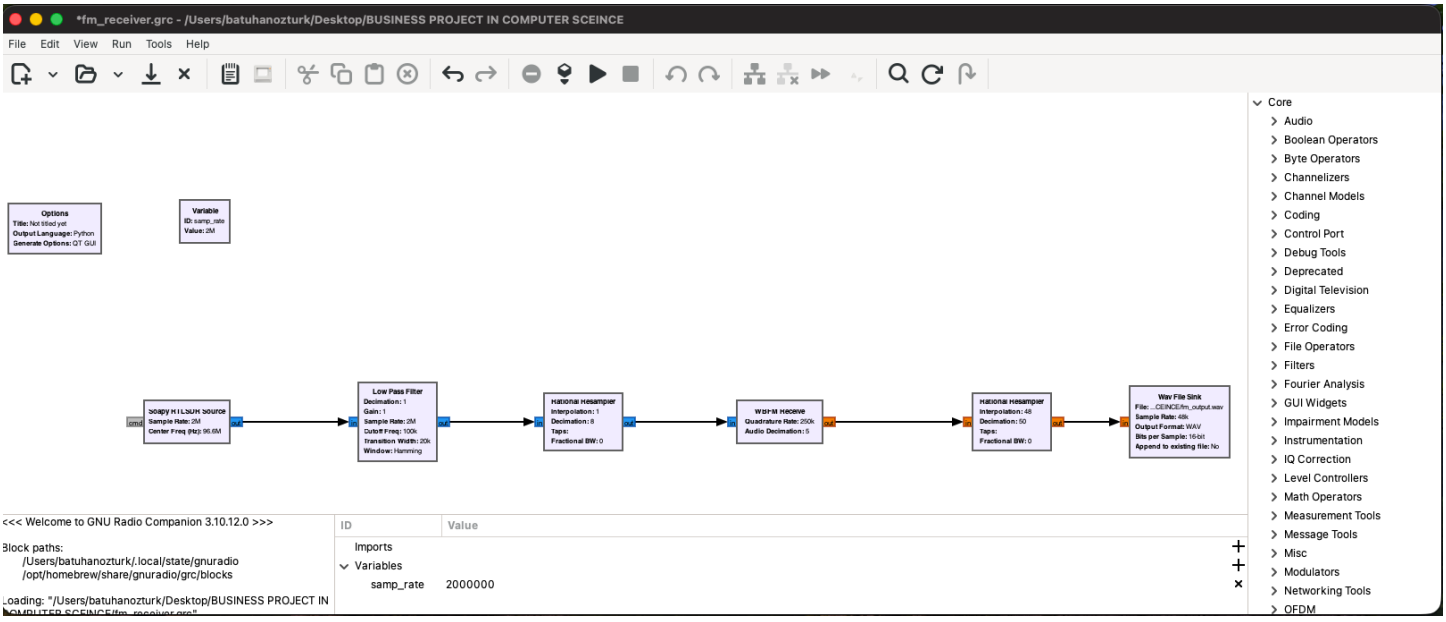
The process of signal reception and processing through software was made possible by the installation of GNU Radio and the necessary RTL-SDR drivers. To verify if the SDR device was running on the system, the rtl_test utility was ran. The results of the test were positive in that they indicated the device had been found successfully and that there would be no issue in streaming the sample data.

Following the hardware testing, a short FM band recording was made. Thus, confirming that the air radio signals could be received and stored for further processing. After these three steps the system was handed over to the next stage of the project for the development of the FM receiver.

5.2 FM RECEIVER IMPLEMENTATION

During the second week, the implementation of a full FM receiver was done via GNU Radio's companion software (GRC). The purpose of this step was to sketch out a very functional signal processing pipeline that would be able to receive an FM broadcast signal and then convert it into an audio waveform.

The pipeline kicks off with the RTL SDR Source capturing raw IQ samples from the FM band.Next, a Low Pass Filter is employed to separate the wanted channel and to lower the out-of-band noise. Subsequently, the Rational Resampler raising the sampling frequency to that compatible with the needs of the FM demodulation stage comes after the filtering.



By processing the signal through the WBFM Receive block, the FM demodulation is done, and the audio signal is taken out from the carrier. After that, another resampling stage takes the output to standard audio sampling rate, and the output is saved in a WAV File Sink. This basically means that the received audio can be later analyzed with Python-based signal processing tools.

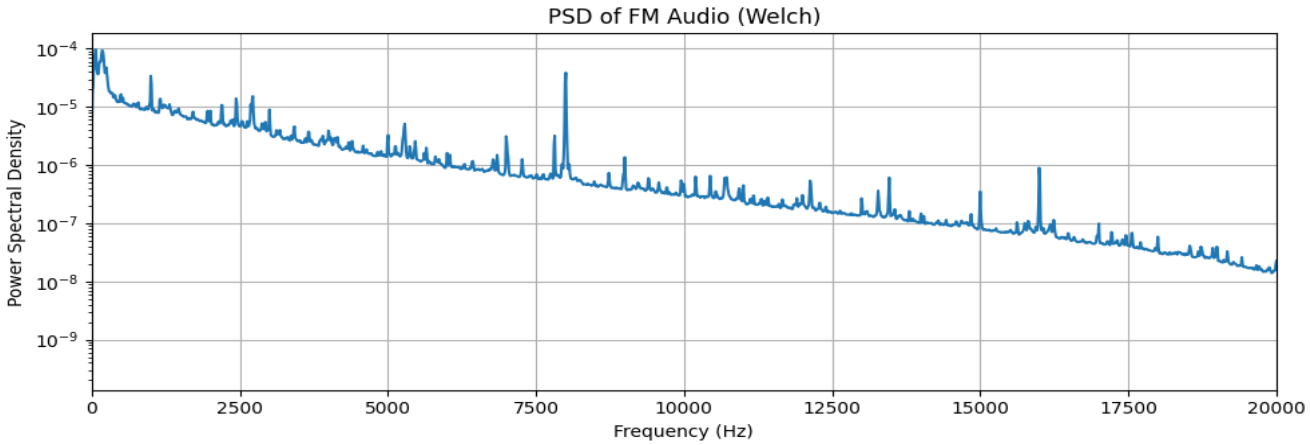
When this stage was over, the FM receiver pipeline was capable of real-time FM signal demodulation and stable audio output production. Furthermore, the generated WAV files were the principal input for the subsequent weeks' analysis and noise reduction steps.

5.3 PYTHON BASED SIGNAL ANALYSIS

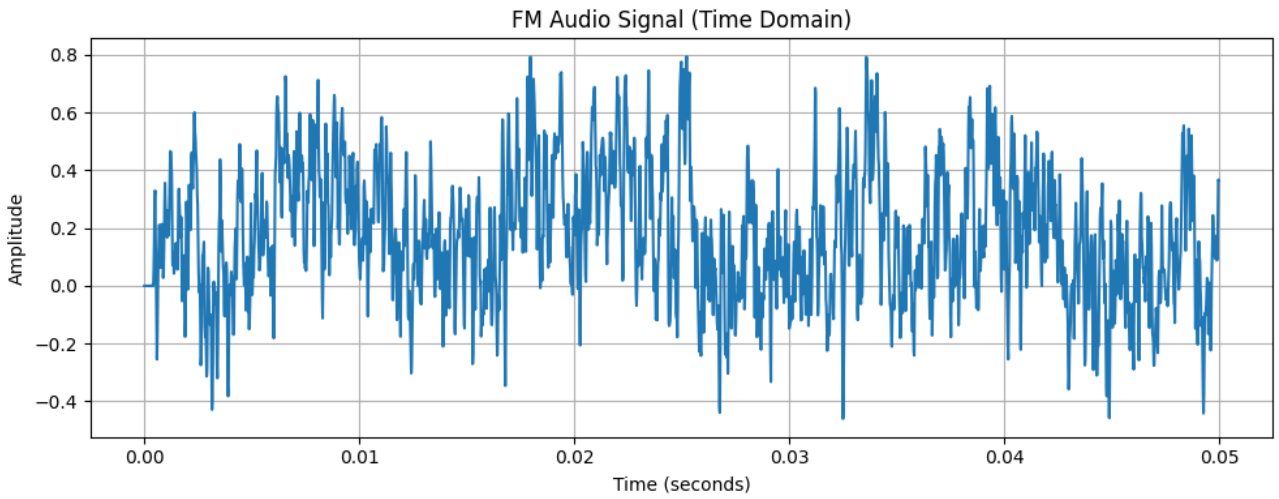
5.3.1 Time and Frequency Domain Inspection

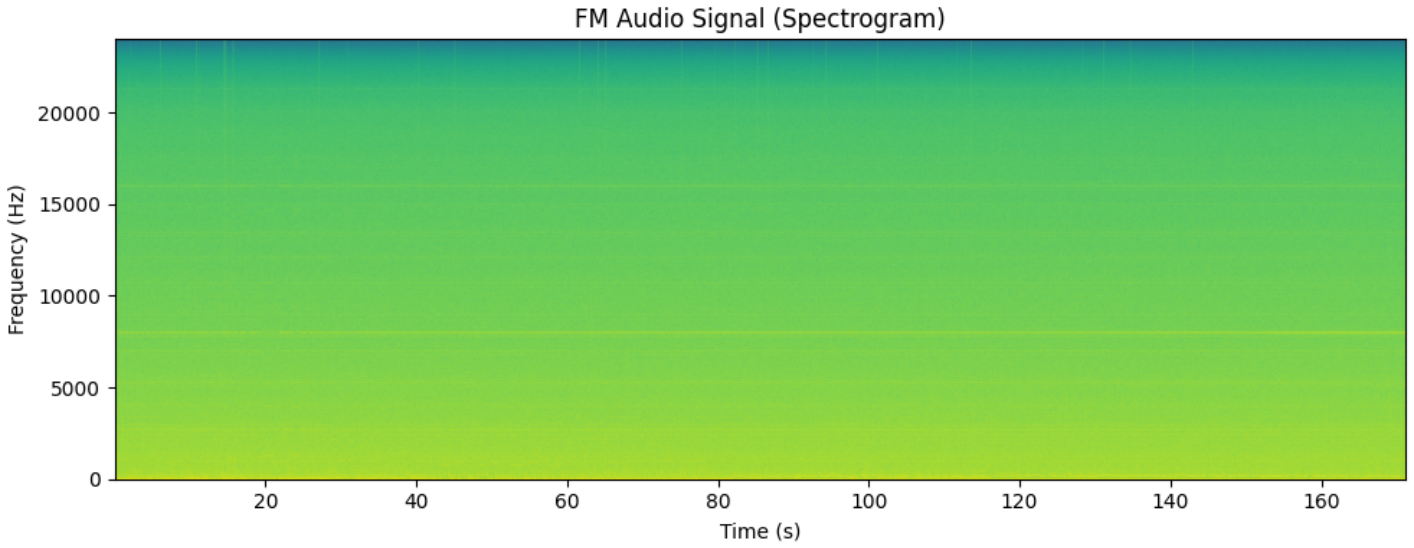
Upon obtaining the FM audio output (fm_output.wav) from the GNU Radio receiver, a signal inspection of the first level was carried out using a small Python script (inspect_wav.py). The purpose of this operation was to get a grip on the basic audio characteristics and the signal's content of the recording was checked visually before any noise reduction method was applied.

Firstly, the script makes use of scipy io wavfile read () to load the WAV file, then it prints out all the essential metadata like the sample rate, data type, and array shape. In case the audio is stereo,only one channel is picked up for the sake of consistency in the analysis. The next step is to normalize the signal to prevent scale differences during plotting.

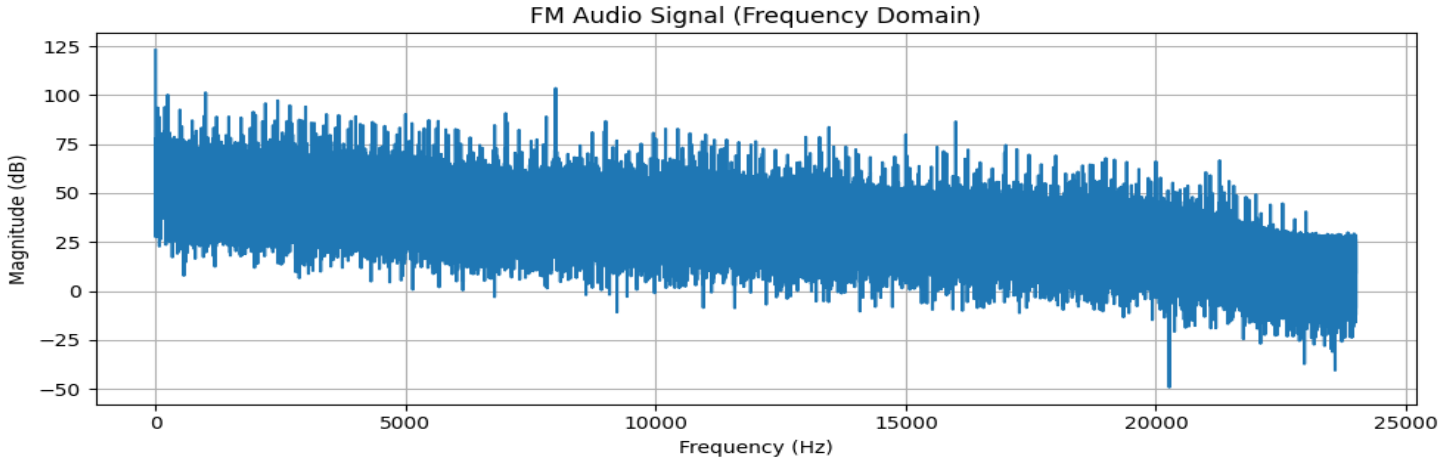


The script goes to the length of analyzing the signal in the time domain by plotting the waveform of the audio for the first 0.05 seconds. The quick visual confirmation this provides is that the recording indeed contains meaningful variations and not silence or clipping.





For analyzing the frequency domain, the script makes use of a real FFT (np.fft.rfft) that it calculates and subsequently plots the spectrum in dB.Doing this not only facilitates seeing how the energy is distributed across the entire range of sound but also whether the expected audio band is present.



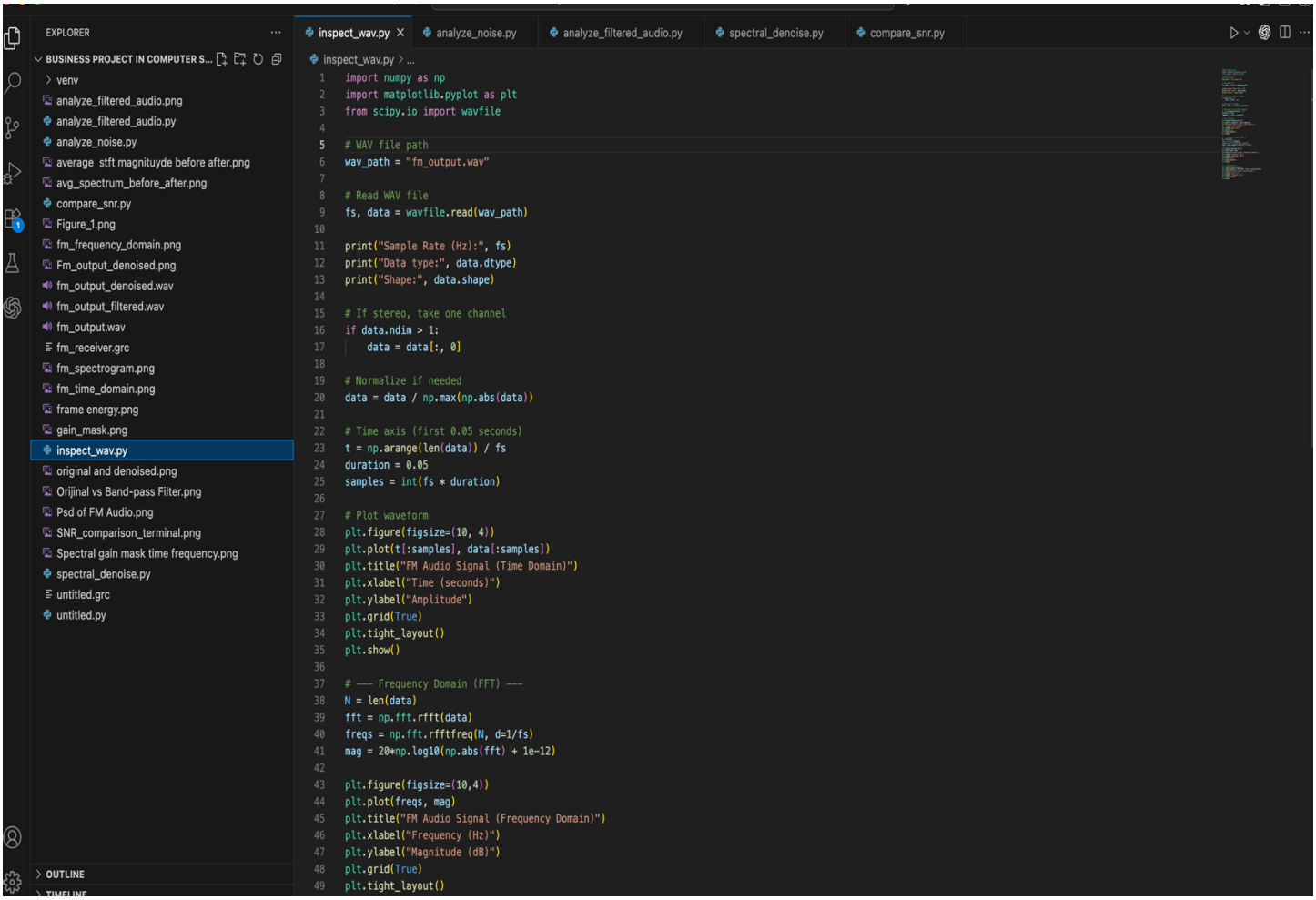
Inspect_wav.py Code Part

The inspect_wav.py script has been designed for the initial inspection of the demodulated FM audio file.

To start,the script reads the WAV file with the help of the SciPy library and shows the basic information like sampling rate,data type and signal shape. In case there are more than one channel in the audio file, the analysis will be carried out on just one channel to make it easier.

The next step will be normalizing the signal, which is to make sure that the amplitude scaling is the same everywhere. A small part of the signal (50 milliseconds) is taken out and its behavior is shown in a plot in the time domain, which is basically the visual form of the waveform.

To sum up, the whole signal undergoes a Fast Fourier Transform (FFT) to get the representation of the signal in the frequency domain. The magnitude spectrum obtained is plotted in decibels so that the dominant frequency components and the noise distribution are practically visible. By means of this script, one can easily and quickly check the validity of FM audio output, thus saving time and hassle of working with advanced noise analysis and denoising methods.

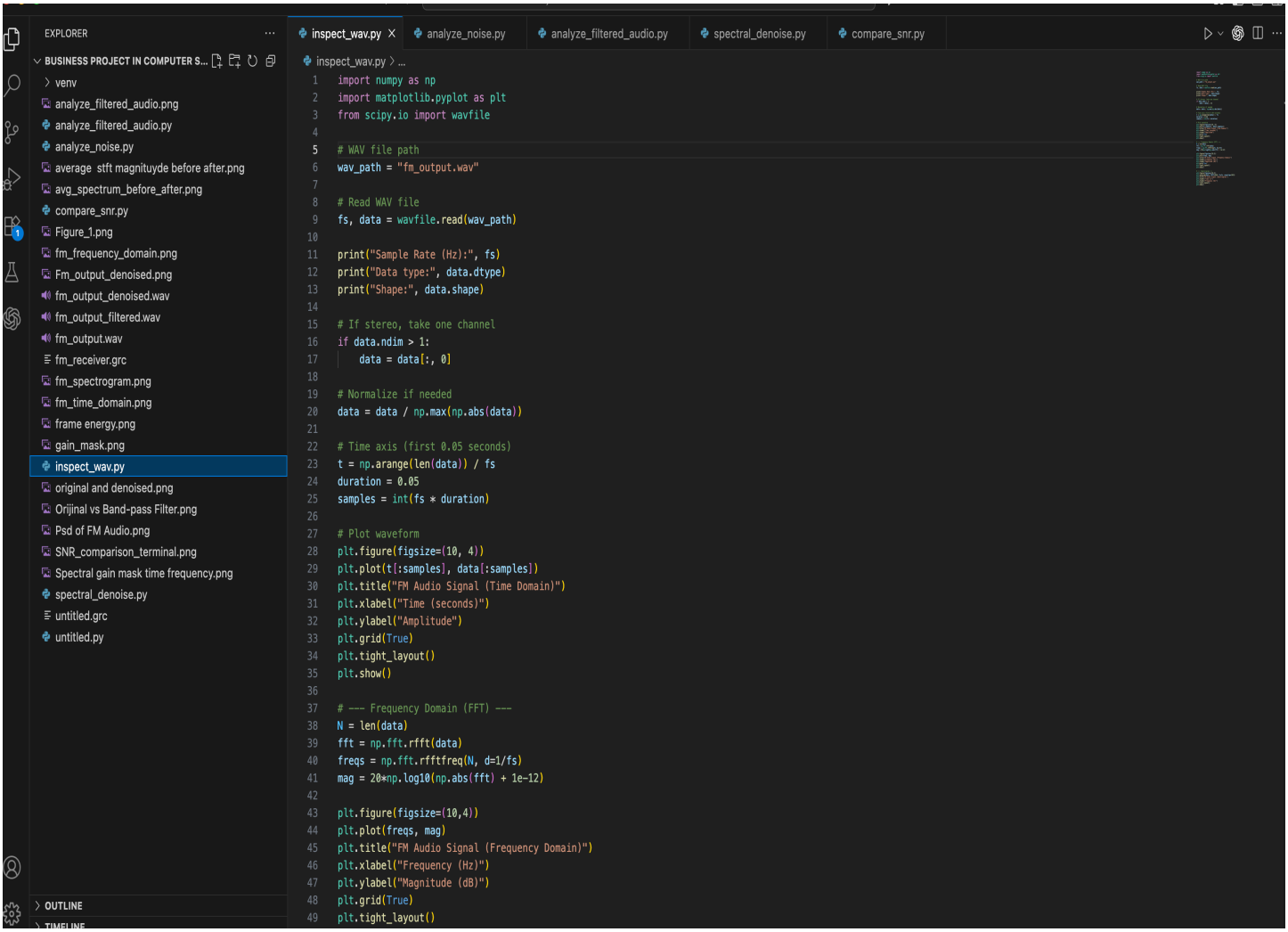


Analyze_noise.py Code Part

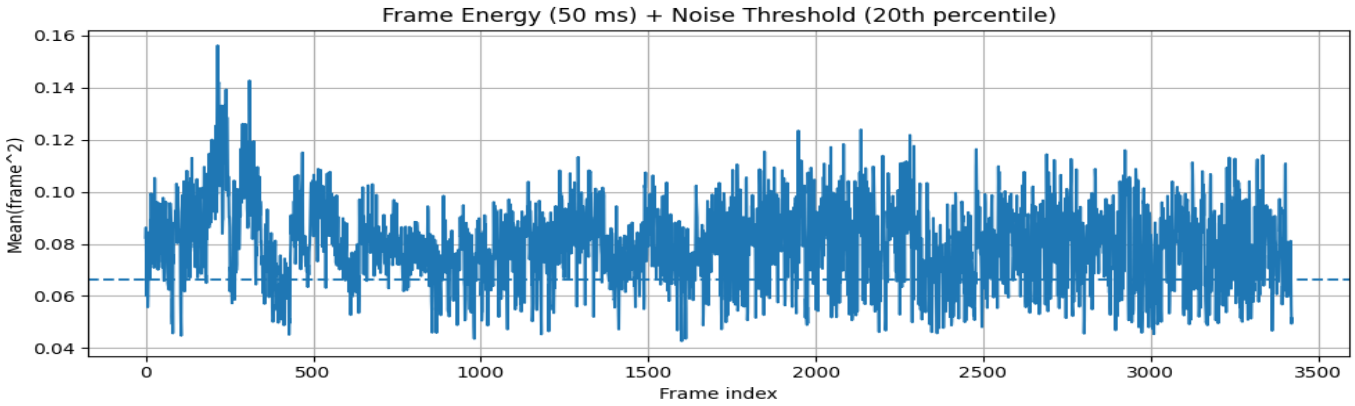
The noise characteristics of the FM audio signal were examined more quantitatively through the developing of the script analyze noise.py.

Rather than analyzing the entire signal, the audio was chopped into short segments of 50 milliseconds each. The signal energy was computed for every frame by calculating the mean squared value of the samples.

The use of this frame-based technique gives the opportunity to see at what points in time the signal energy is high or low and also to separate the areas that are regarding either high-energy (signal-dominant) or low-energy (noise-dominant) accordingly.



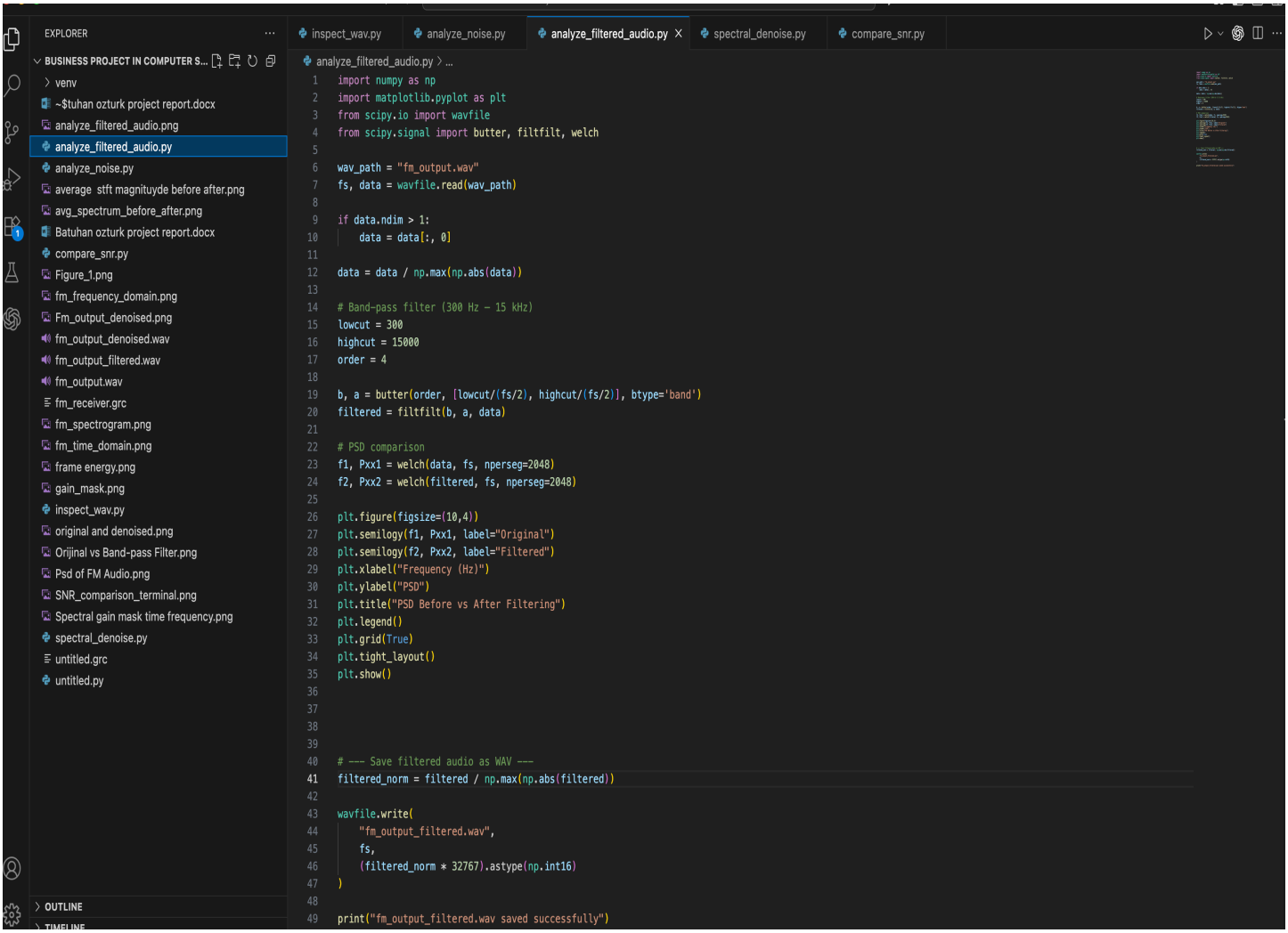
A noise threshold is then computed by using a percentile-based approach. It is considered that the frames which have energy values below this threshold are representing the background noise.
The final plot visually represents the distribution of the frame energy as well as the selected noise threshold, thus, giving a clear visual reference for later denoising steps.



Analyze_Filtered_Audio.py Code Part

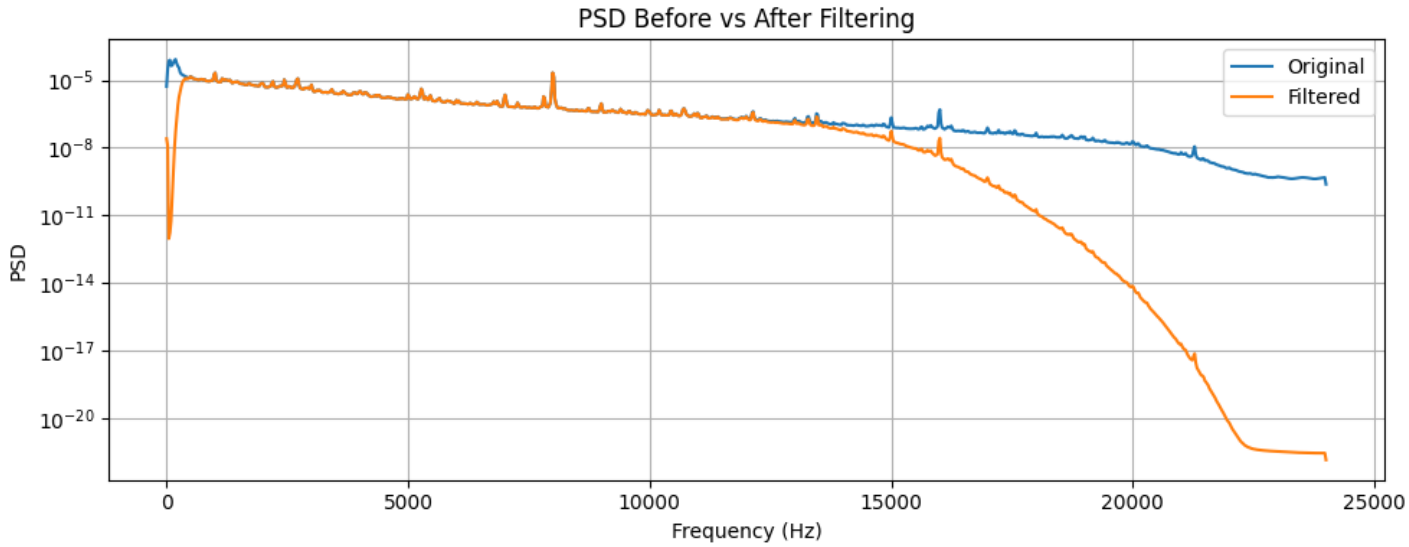
The script analyze_filtered_audio.py is intended to assess the influence of band-pass filtering on the demodulated FM audio signal in the most basic way.

The first operation performed by the script is to read the original FM audio file and if there are multiple channels in the audio, then it converts it into a mono signal. Then the normalized signal is prepared to guarantee numerical stability during processing and comparison.



The sound signal is first heavily filtered through a Butterworth oscillator with center frequencies of 300 Hz and 15 kHz. The range of these frequencies indicates the human speech and audio content in general and the filtering is considered to be noise outside this range.

That's why, to verify the filtering effect, the Power Spectral Density (PSD) of the original and the filtered audio signals is calculated via Welch's method. It is done in a safe rigorous way that allows to see the frequency-domain energy distribution of the signal.



The PSD plot produced presents the original and filtered signals on a logarithmic scale and it is evident how the band-pass filter has reduced the out-of-band noise. Moreover, the last activity is to normalize the filtered sound and save it as a new WAV file for the purpose of future comparison and evaluation.

5.4 NOISE REDUCTION TECHNIQUES

In the case of actual wireless systems, especially in IoT and Smart Energy, the received signals are usually dirty due to noise and interference. The FM audio signal has been obtained and its basic characteristics analyzed, now the quality of the signal is to be improved through application of noise reduction methods. In this project, two different methods were studied and their results were compared.

5.4.1 Band-pass Filtering

To start with, a band-pass filter was used in a very simple manner with the FM audio signal.

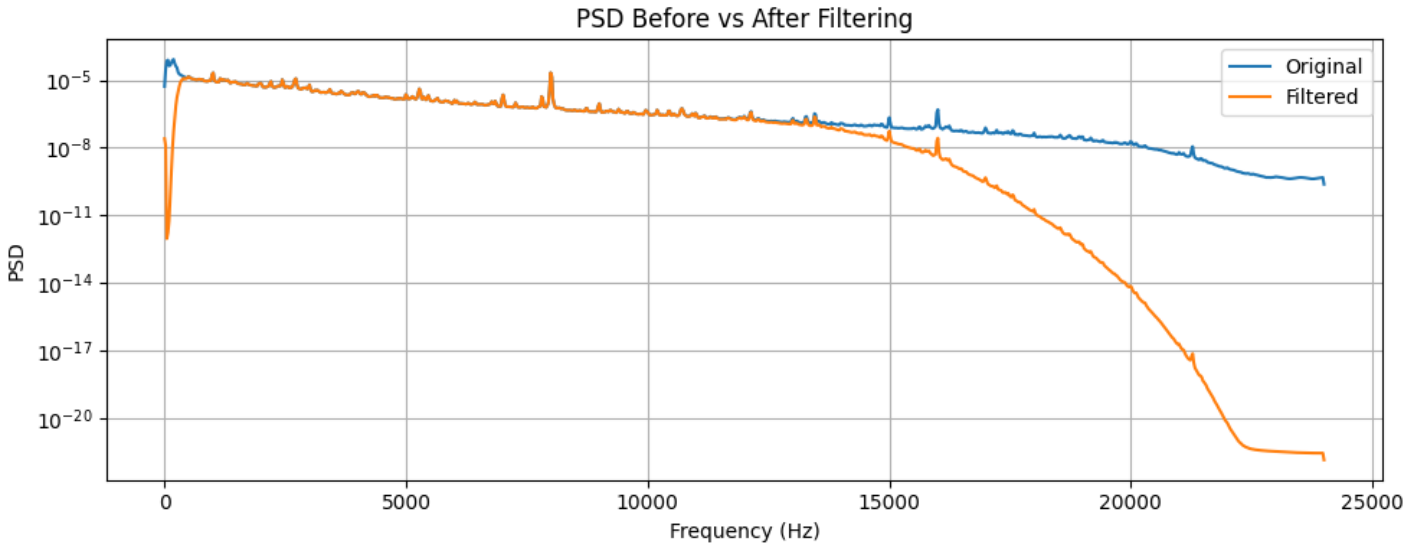
The function of this filter is:

- to eliminate low-frequency components that are not part of the useful audio signal,
- and to reduce high-frequency noise outside the usual FM audio band.

For this study,a band-pass filter allowing frequencies from about 300 Hz to 15 kHz was employed. This range includes mostly the relevant audio content while suppressing the unwanted noise.

While band-pass filtering does cut down the noise,it also brings along a major limitation:

It applies the same treatment to all frequencies within the band that is allowed through. Consequently, noise which accompanies the signal spectrum cannot be cancelled out properly.

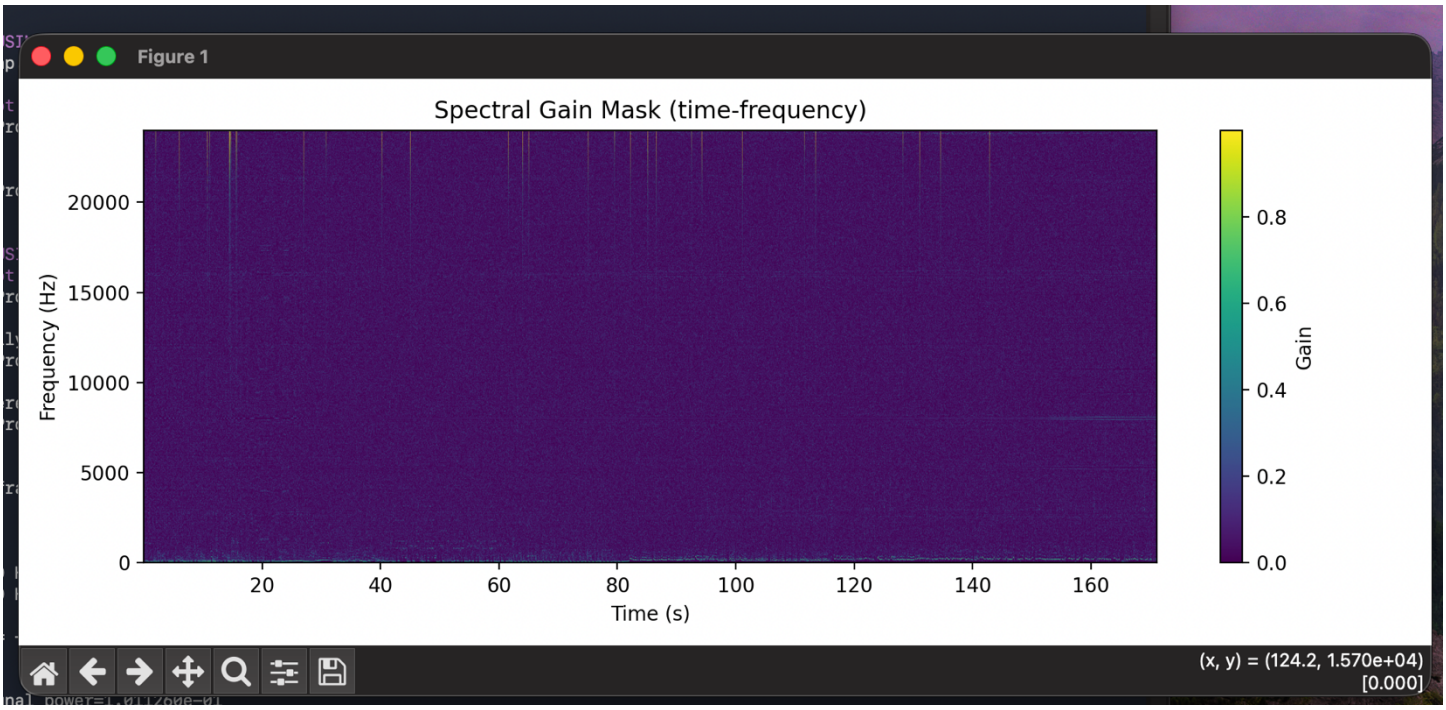


5.4.2 Spectral Denoising Using STFT

The limitations of simple filtering techniques were dealt with the implementation of a spectral denoising approach that was based on the Short-Time Fourier Transform (STFT).

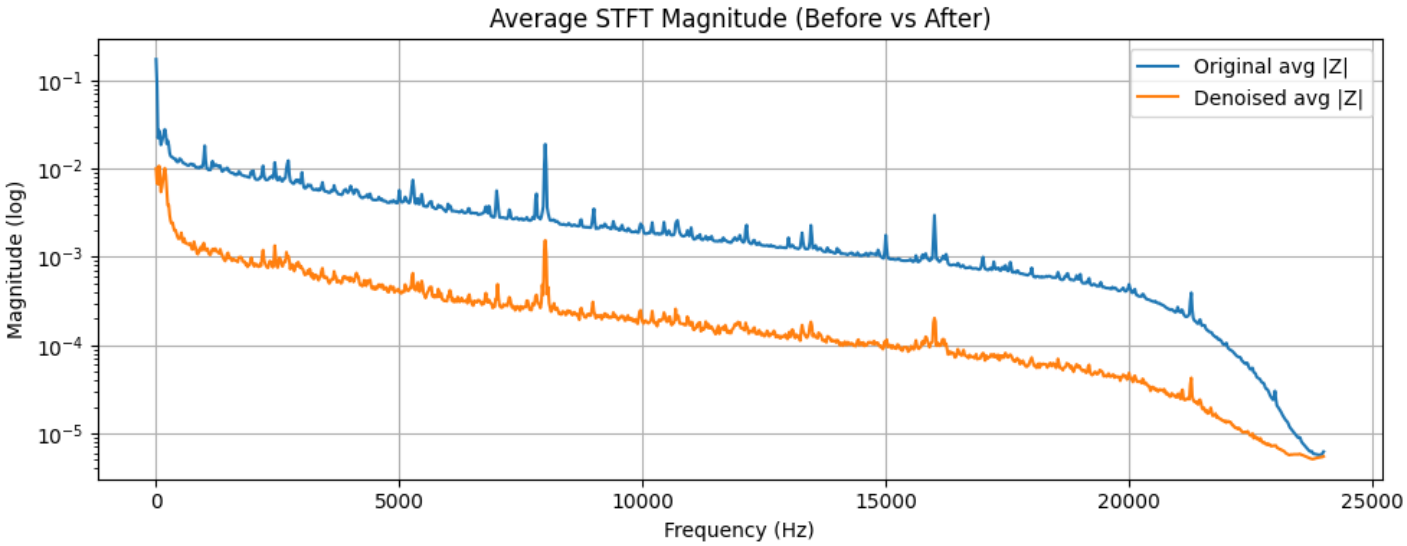
The central concept of this procedure is:

- to perform an analysis of the signal in the time-frequency domain,
- to assess the level of noise,
- and to noisily sharp frequency components to a selective extent.



The operation is composed of the following stages:

- The audio signal is cut into tiny overlapping snippets.
- A time-frequency picture is obtained through the application of STFT.
- A noise threshold is calculated from regions of low energy.
- A spectral gain mask is built to make noise-dominated components less audible.
- The denoised signal is made by using the inverse STFT.



This method, in contrast to band-pass filtering, is able to follow the changes in both time and frequency, thereby making it more appropriate for the real-world noisy environments.

5.4.3 Comparison of Noise Reduction Approaches

The findings indicate that:

- band-pass filtering involves little enhancement .
- spectral denoising actually provides great noise reduction with minor distortion of the useful signal.

Comparison above emphasizes the significance of algorithmic signal processing, which is the main idea in computer science–based signal analysis.

```
Saved: fm_output_denoised.wav
STFT params: nperseg=2400, noverlap=1800, noise_percentile=20, p=2.0
```

6-EVULATION

A quantitative Signal-to-Noise Ratio (SNR) analysis was used to determine the performance of the suggested system. The evaluation's primary goal was to indicate the degree to which the various noise reduction methods influenced the quality of the FM audio signal produced.

SNR Measurement Method

A frame based approach was used to calculate SNR.The audio signal was chopped into very small time frames (50 ms), and within each frame, the power of the signal and the power of the noise were evaluated. The segments with low energy were considered to be noise-dominant, while the segments with high energy were considered to be signal-dominant. This approach offers a reliable and reproducible manner to evaluate various stages of processing.

The same SNR calculation method was employed on:

- the FM audio signal without any modifications.
- the band-pass filtered signal.
- the spectrally denoised signal.

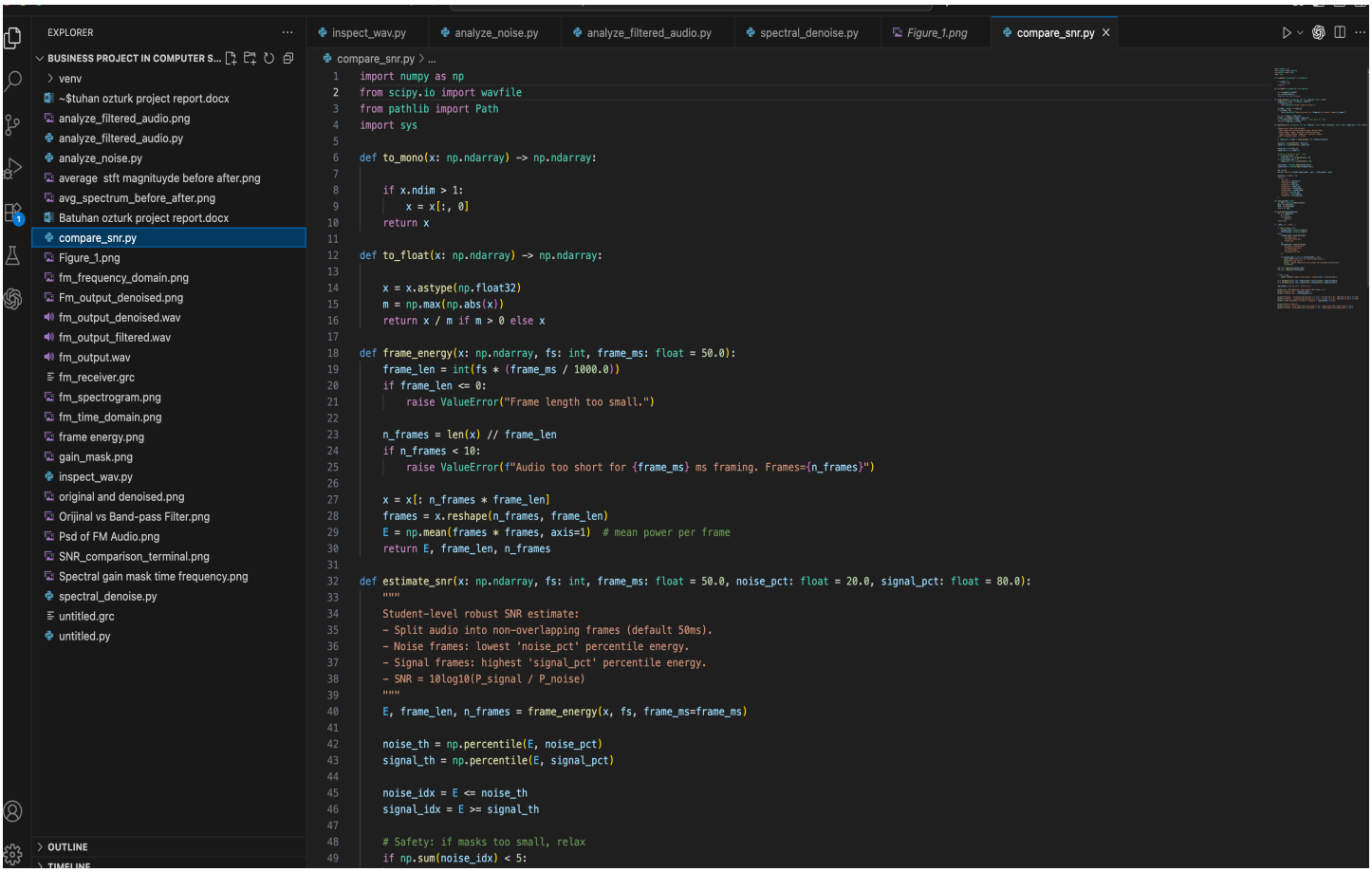
Results and Comparison

The low SNR of the original FM audio signal was due to the broadband noise and the distortion which were introduced during the reception process. The band-pass filter provided only a little improvement because it removed only the noise which was out of the band while leaving a considerable amount of in-band noise.

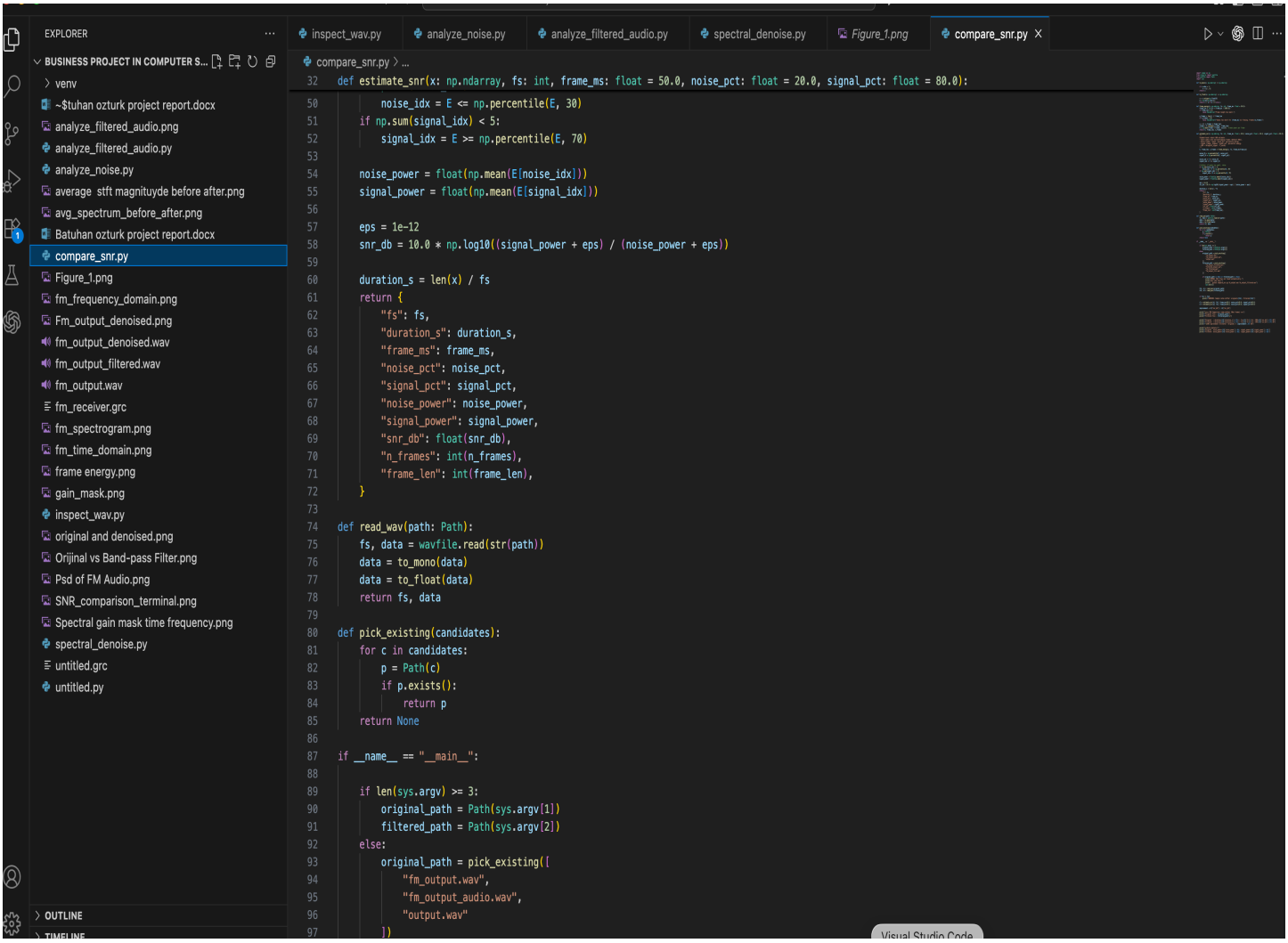
The spectral denoising technique led to SNR getting much better. The use of a time-frequenct gain mask suppressed the noise-dominated current of spectral components while the signal-dominant parts were still preserved. Therefore, the denoised signal attained a considerable gain in SNR compared to the original signal.

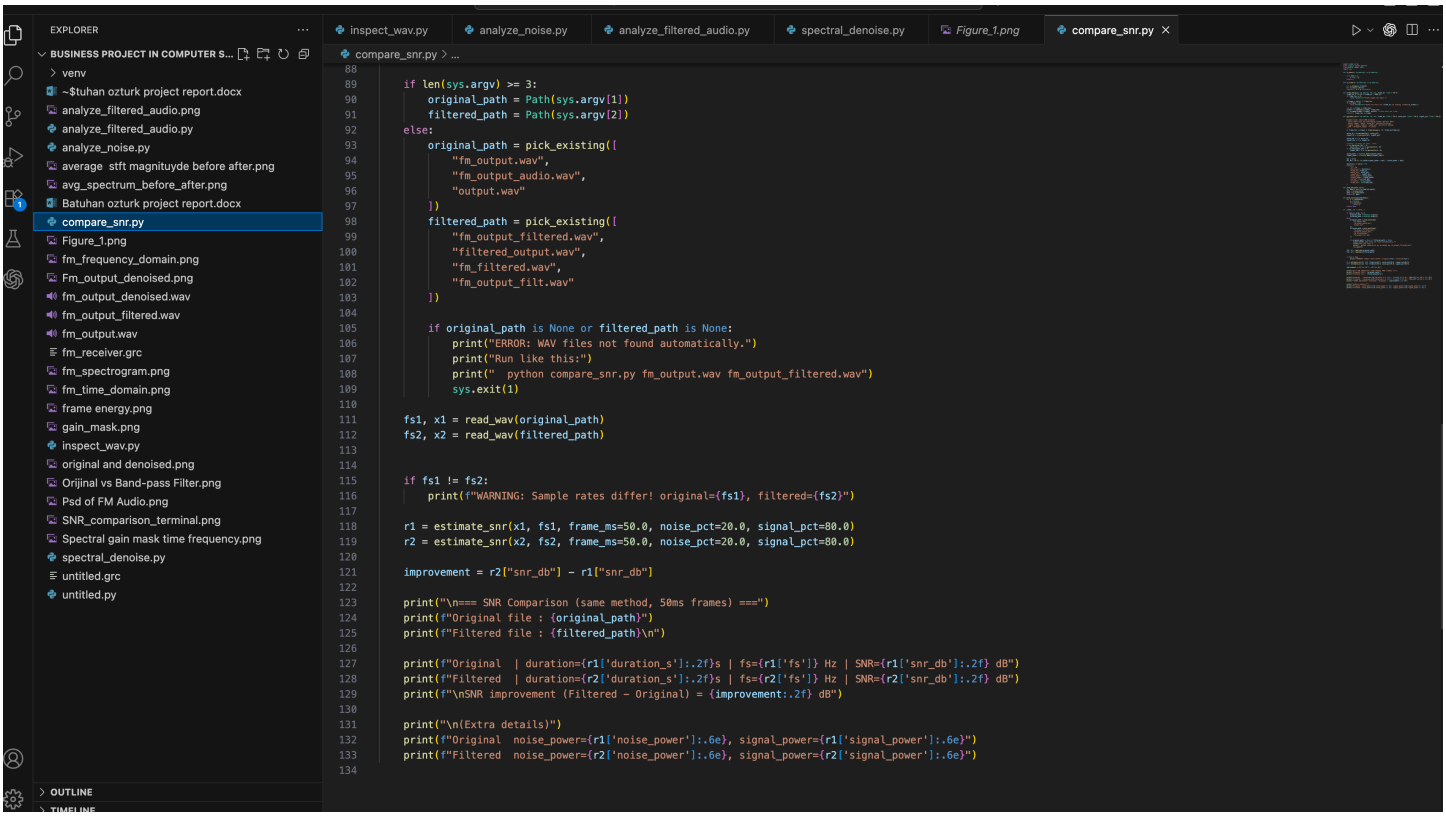
Interpretation

The SNR values shown in this assessment were determined with the help of a custom Python script (compare_snr.py).This script streamlines the evaluation of different audio outputs produced in the course of the project, starting with the original FM signal and the processed (filtered or denoised) versions.



The script loads the audio files,scales their volumes to the same level and applies the same frame based SNR estimation technique to all signals.By using the same frame length and calculation method,a fair and consistent comparison is guaranteed.The script generates numerical SNR values as well as SNR improvement in decibels, thus facilitating direct evaluation of noise reduction performance.


EXPLORER
BUSINESS PROJECT IN COMPUTER S...
venv
~$tuhan ozturk project report.docx
analyze_filtered_audio.png
analyze_filtered_audio.py
analyze_noise.py
average stft magnituyde before after.png
avg_spectrum_before_after.png
Batuhan ozturk project report.docx
compare_snr.py
Figure_1.png
fm_frequency_domain.png
Fm_output_denoised.png
fm_output_denoised.wav
fm_output_filtered.wav
fm_output.wav
fm_receiver.grc
fm_spectrogram.png
fm_time_domain.png
frame energy.png
gain_mask.png
inspect_wav.py
original and denoised.png
Orijinal vs Band-pass Filter.png
Psd of FM Audio.png
SNR_comparison_terminal.png
Spectral gain mask time frequency.png
spectral_denoise.py
untitled.grc
untitled.py
OUTLINE
TIMELINE
inspect_wav.py
analyze_noise.py
analyze_filtered_audio.py
spectral_denoise.py
Figure_1.png
compare_snr.py
compare_snr.py > ...
def estimate_snr(x: np.ndarray, fs: int, frame_ms: float = 50.0, noise_pct: float = 20.0, signal_pct: float = 80.0):
noise_idx = E <= np.percentile(E, 30)
if np.sum(signal_idx) < 5:
signal_idx = E >= np.percentile(E, 70)
noise_power = float(np.mean(E[noise_idx]))
signal_power = float(np.mean(E[signal_idx]))
eps = 1e-12
snr_db = 10.0 * np.log10((signal_power + eps) / (noise_power + eps))
duration_s = len(x) / fs
return {
"fs": fs,
"duration_s": duration_s,
"frame_ms": frame_ms,
"noise_pct": noise_pct,
"signal_pct": signal_pct,
"noise_power": noise_power,
"signal_power": signal_power,
"snr_db": float(snr_db),
"n_frames": int(n_frames),
"frame_len": int(frame_len),
}
def read_wav(path: Path):
fs, data = wavfile.read(str(path))
data = to_mono(data)
data = to_float(data)
return fs, data
def pick_existing(candidates):
for c in candidates:
p = Path(c)
if p.exists():
return p
return None
if __name__ == "__main__":
if len(sys.argv) >= 3:
original_path = Path(sys.argv[1])
filtered_path = Path(sys.argv[2])
else:
original_path = pick_existing([
"fm_output.wav",
"fm_output_audio.wav",
"output.wav"
])
Visual Studio Code


EXPLORER
BUSINESS PROJECT IN COMPUTER S...
venv
~$tuhan ozturk project report.docx
analyze_filtered_audio.png
analyze_filtered_audio.py
analyze_noise.py
average stft magnituyde before after.png
avg_spectrum_before_after.png
Batuhan ozturk project report.docx
compare_snr.py
Figure_1.png
fm_frequency_domain.png
Fm_output_denoised.png
fm_output_denoised.wav
fm_output_filtered.wav
fm_output.wav
fm_receiver.grc
fm_spectrogram.png
fm_time_domain.png
frame energy.png
gain_mask.png
inspect_wav.py
original and denoised.png
Orijinal vs Band-pass Filter.png
Psd of FM Audio.png
SNR_comparison_terminal.png
Spectral gain mask time frequency.png
spectral_denoise.py
untitled.grc
untitled.py
OUTLINE
TIMELINE
inspect_wav.py
analyze_noise.py
analyze_filtered_audio.py
spectral_denoise.py
Figure_1.png
compare_snr.py
compare_snr.py > ...
if len(sys.argv) >= 3:
original_path = Path(sys.argv[1])
filtered_path = Path(sys.argv[2])
else:
original_path = pick_existing([
"fm_output.wav",
"fm_output_audio.wav",
"output.wav"
])
filtered_path = pick_existing([
"fm_output_filtered.wav",
"filtered_output.wav",
"fm_filtered.wav",
"fm_output_filt.wav"
])
if original_path is None or filtered_path is None:
print("ERROR: WAV files not found automatically.")
print("Run like this:")
print(" python compare_snr.py fm_output.wav fm_output_filtered.wav")
sys.exit(1)
fs1, x1 = read_wav(original_path)
fs2, x2 = read_wav(filtered_path)
if fs1 != fs2:
print(f"WARNING: Sample rates differ! original={fs1}, filtered={fs2}")
r1 = estimate_snr(x1, fs1, frame_ms=50.0, noise_pct=20.0, signal_pct=80.0)
r2 = estimate_snr(x2, fs2, frame_ms=50.0, noise_pct=20.0, signal_pct=80.0)
improvement = r2["snr_db"] - r1["snr_db"]
print("\n=== SNR Comparison (same method, 50ms frames) ===")
print(f"Original file : {original_path}")
print(f"Filtered file : {filtered_path}\n")
print(f"Original | duration={r1['duration_s']:.2f}s | fs={r1['fs']} Hz | SNR={r1['snr_db']:.2f} dB")
print(f"Filtered | duration={r2['duration_s']:.2f}s | fs={r2['fs']} Hz | SNR={r2['snr_db']:.2f} dB")
print(f"\nSNR improvement (Filtered - Original) = {improvement:.2f} dB")
print("\n(Extra details)")
print(f"Original noise_power={r1['noise_power']:.6e}, signal_power={r1['signal_power']:.6e}")
print(f"Filtered noise_power={r2['noise_power']:.6e}, signal_power={r2['signal_power']:.6e}")

```
(venv) batuhanozturk@Batuhans-MacBook-Pro BUSINESS PROJECT IN COMPUTER SCEINCE % python compare_snr.py fm_output.wav fm_output_denoised.wav


=== SNR Comparison (same method, 50ms frames) ===
Original file : fm_output.wav
Filtered file : fm_output_denoised.wav

Original  | duration=171.18s | fs=48000 Hz | SNR=2.38 dB
Filtered  | duration=171.12s | fs=48000 Hz | SNR=9.11 dB

SNR improvement (Filtered - Original) = 6.73 dB

(Extra details)
Original  noise_power=5.844001e-02, signal_power=1.011260e-01
Filtered  noise_power=3.734510e-03, signal_power=3.043608e-02
(venv) batuhanozturk@Batuhans-MacBook-Pro BUSINESS PROJECT IN COMPUTER SCEINCE %
```

7-BUSINESS CASE (IOT & SMART ENERGY)

The focus of this project goes beyond just signal processing; it is aimed at solving a practical business problem that is quite common in IoT and Smart Energy systems, namely ensuring reliable data transmission even in conditions that are noisy and unpredictable.

In smart energy applications like remote smart metering and distributed energy monitoring, wireless communication links often suffer from interference, fading due to distance, and environmental noise. The resulting poor signal quality may corrupt measurements, cause data loss, or lead to wrong operational decisions.

The SDR-based solution we are proposing is not only low-cost and flexible but also a better alternative to conventional proprietary receivers. Signal quality can be assessed, enhanced, and adjusted without any hardware change through software-defined processing and Python-based analysis. Thus, energy companies save on both deployment and maintenance costs.

The SNR analysis results are significant from the business point of view. The SNR improvement measured after the application of the denoising technique indicates that it is possible to retrieve good data even in the presence of noise, thus allowing more efficient data acquisition, better network design, and decision-making based on data regarding the location of the antennas, the power of the transmission, and the scalability of the system.

To sum it up, the project demonstrates the impact of Computer Science activities such as control by software, algorithmic signal analysis, and data-driven evaluation on the economy at once for IoT and Smart Energy systems.

8- CONCLUSION AND FUTURE WORK

This project showcased a simple and affordable system for wireless data analysis utilizing an SDR, GNU Radio, and Python for signal processing. The whole system went through development, implementation, and testing starting from the reception of raw FM signals and at each stage the focus was on signal quality in the noisy environment.

With the results obtained, it is demonstrated that the application of software signal processing can lead to a drastic increase in the quality of the data, however, this is not the case with hardware. Although basic band-pass filtering results in minor improvements, the use of spectral-denoising methods has produced a significant and quantifiable increase in the signal-to-noise ratio. The assessment corroborates that the deployment of sophisticated DSP algorithms is a prerequisite for communicating over low-SNR channels that are often the case in many applications where such technology is used.

To a large extent this study is a proof of concept that Computer Science principles like algorithmic processing, data-driven evaluation and software reconfigurability, could be directly used to resolve issues in IoT and Smart Energy. The possibility of measuring, analyzing and improving the quality of a signal with the help of software grants companies both technical and financial advantages.

The work could comprise of the existing system being extended to cover different modulation schemes and actual IoT communication protocols as its future aspect. Moreover, it is proposed that the noise-reduction and machine-learning-based signal-classification techniques be the subject of further investigation. Lastly, testing the system on actual IoT sensor transmissions would not only prove its practical relevance but also enhance it.

9- REFERENCES

- Ettus Research. (2023) *Software Defined Radio (SDR) Basics*. https://www.ettus.com.
- GNU Radio Project. (2024) *GNU Radio Documentation*. https://www.gnuradio.org/doc/
- Lyons, R.G. (2011) *Understanding Digital Signal Processing*. 3rd edn. Boston: Pearson Education.
- Proakis, J.G. and Manolakis, D.G. (2007) *Digital Signal Processing: Principles, Algorithms, and Applications*. 4th edn. Upper Saddle River: Pearson Prentice Hall.
- Sklar, B. (2001) *Digital Communications: Fundamentals and Applications*. 2nd edn. Upper Saddle River: Prentice Hall.
- Welch, P. (1967) 'The use of fast Fourier transform for the estimation of power spectra', *IEEE Transactions on Audio and Electroacoustics*, 15(2), pp. 70–73.