**SMARTUNES:**

**LISTENING TO YOUR AIR CONDITIONER**

## Muhammad Saad Abdullah

A Dawlance R&D Project

Contents

[Introduction 3](#_Toc143482811)

[Methodology 4](#_Toc143482812)

[Description 7](#_Toc143482813)

[Visual Data 15](#_Toc143482814)

[Conclusion 18](#_Toc143482815)

# introduction

In today's world, audio signals hold vital information, making their analysis crucial across various fields. This project delves into audio processing and analysis, exploring techniques to extract insights from audio recordings. By combining digital signal processing, spectral analysis, and pattern recognition, this project showcases an approach to understanding information within audio signals.

This project aims to create a pipeline for audio analysis, encompassing steps like recording, signal filtering, spectrogram computation, constellation mapping, and data interpretation. Each step is designed to extract significant information from the audio signal for applications ranging from audio quality assessment to information encoding.

|  |
| --- |
| methodology |
| The methodology employed in this project involves a systematic approach to extract meaningful insights from audio recordings. The process encompasses several distinct steps, each contributing to the comprehensive analysis of audio signals.  **Audio Recording and Sampling:**  The initial step involves selecting an appropriate sampling frequency, which determines how frequently the audio signal is measured. A suitable sampling frequency is chosen to accurately capture the nuances of the audio signal. The audio is then recorded using the selected sampling frequency and a predefined duration, resulting in a digital representation of the audio waveform.  **Signal Filtering:**  After recording, the audio signal undergoes filtering to isolate specific frequency components of interest. This step aims to focus on the frequency range relevant to the analysis. Bandpass filters are designed with predefined cutoff frequencies to allow only the desired frequencies to pass through. The application of these filters results in a filtered audio signal that eliminates unwanted noise and interference.  **Spectral Analysis and Spectrogram Computation**:  With the filtered audio signal in hand, spectral analysis is performed using the Short-Time Fourier Transform (STFT). The STFT breaks down the audio signal into its constituent frequency components over short time intervals. The complex STFT values are then converted into amplitude values, enabling the computation of the spectrogram. The spectrogram visually represents the distribution of frequencies over time, providing valuable insights into the temporal and frequency characteristics of the audio signal.  **Constellation Map Generation:**  The heart of the methodology lies in the creation of a constellation map. This map captures significant data points by identifying local maximum points within the spectrogram. The map creation involves defining parameters such as distance thresholds to determine which local maxima qualify as valid points. This step condenses complex spectrogram information into a simplified representation that highlights patterns and relationships among data points.  **Data Point Extraction and Processing:**  From the constellation map, data points are extracted and processed to unveil the underlying information. These data points are sorted based on a predefined criterion, facilitating subsequent processing. Each sorted point is assigned a binary value based on its frequency, effectively translating the frequency information into a binary format.  **Parity Bit Calculation:**  To ensure data integrity, parity bits are calculated for the processed binary values. Data is grouped into rows, typically containing eight values per row. A parity bit is computed for each row, serving as a checksum that aids in detecting potential errors or discrepancies in the data.  **Interpretation and Visualization:**  The processed data is then interpreted to decode the embedded information in the audio signal. This interpretation provides insights into the characteristics of the audio recording. Additionally, results are visualized through representations such as spectrograms and constellation maps, aiding in the comprehension and communication of the extracted insights.  **Application and Analysis:**  The methodology is applied to various audio recordings, assessing its effectiveness and versatility across different scenarios. The obtained results are analyzed to draw conclusions regarding the project's objectives, as well as to identify potential applications of the employed techniques.  **Documentation and Reporting:**  Throughout the project, meticulous documentation is maintained, including parameter settings, code implementation, and rationale behind decisions. The culmination of the project is a comprehensive report summarizing the entire methodology, presenting results, and sharing insights gained from the analysis of audio signals. |

# Description

The code begins with Audio Recording and Sampling, a fundamental aspect of audio analysis. The chosen sampling frequency (fs) of 44,000 Hz reflects a deliberate decision to ensure high-fidelity capture of audio nuances. This frequency aligns with the methodology's emphasis on accurate signal representation, analogous to capturing fine details in visual imagery. Additionally, the predetermined duration of 14 seconds solidifies the time frame for recording. Importantly, the interdependence between sampling frequency and duration is acknowledged. Adjustments to either parameter necessitate recalibrating the other to maintain signal accuracy. This relationship is pivotal, ensuring that changes in duration are met with corresponding adjustments in sampling frequency, thus preserving a faithful representation of the sound.

Signal Filtering constitutes the subsequent stage, underscoring the importance of enhancing signal quality. The utilization of bandpass filters substantiates the methodology's proposition to concentrate on specific frequency ranges. With lowcut and highcut frequencies set at 4,800 Hz and 7,200 Hz respectively, the code adheres to the methodology's emphasis on isolating desired frequency components. This resonates with the analogy of refining ingredients in culinary processes. The choice of a fourth-order filter, aligned with the methodology, enhances signal integrity by eliminating extraneous noise. Moreover, the normalization of frequencies to the Nyquist frequency supports comparison across diverse signals, maintaining uniformity in signal processing.

The subsequent phase involves Spectral Analysis and Spectrogram Computation, a cornerstone of the code's functionality. The transformation of the filtered audio signal through the Short-Time Fourier Transform (STFT) echoes the methodology's emphasis on extracting frequency components over time. The subsequent conversion of complex STFT values into amplitude values, akin to magnitude, mirrors the methodology's proposition of constructing a spectrogram. This transformation ensures that the spectrogram provides a tangible representation of frequency variations across time intervals.

Constellation Map Generation, a pivotal aspect of the code, aligns seamlessly with the methodology's focus on identifying significant data points. The code employs a dedicated function, compute\_constellation\_map, to map local maxima within the spectrogram. Parameters such as dist\_freq, dist\_time, and thresh closely adhere to the methodology's recommendations, underscoring the careful selection of points. The outcome, Cmap, echoes the methodology's notion of distilling complex data into an intelligible representation, facilitating subsequent analysis.

The stage of Data Point Extraction and Processing is executed adeptly, capturing the essence of the methodology's emphasis on sorting and processing data points. By employing np.transpose(np.nonzero(Cmap)), the code extracts salient data points, a mirror of the methodology's call to identify significant features. The subsequent sorting through np.lexsort((points[:, 0], points[:, 1])) mirrors the methodology's instruction to arrange points based on a defined criterion.

Parity Bit Calculation, crucial for data integrity, finds its manifestation in the code's logic. Each row's parity bit, as calculated in the code, mirrors the methodology's practice of ensuring error-checking mechanisms. The methodology's rationale of employing parity bits as safeguards against inaccuracies is effectively transposed into the code's execution.

The code's concluding stages of Interpretation and Visualization underscore the project's objectives of comprehending data and presenting insights visually. The interpretation of results and visualization through librosa.display.specshow parallels the methodology's approach of decoding information and rendering it accessible through illustrative means.

Under the Application and Analysis umbrella, the code stands as a testament to the project's practical implementation, corresponding to the methodology's intent of real-world application and assessment. The code's adaptability across various audio recordings embodies the methodology's encouragement to gauge efficacy across diverse scenarios.

In summary, the code and methodology work seamlessly together, leading to a practical implementation of audio signal analysis. The code follows the methodology closely and handles the relationship between sampling frequency and duration well. This connection is vital for accurate signal representation, highlighting the project's commitment to thorough analysis and clear interpretation.

THE CODE

# Import necessary libraries

import sounddevice as sd # Library for audio recording and playback

import matplotlib.pyplot as plt # Library for data visualization

import numpy as np # Library for numerical operations

import librosa.display # Library for audio and music analysis

from scipy.signal import butter, filtfilt # Functions for signal filtering

from scipy import ndimage # Functions for n-dimensional image processing

import pandas as pd # Library for data manipulation and analysis

# Function to compute the constellation map based on given parameters

def compute\_constellation\_map(Y, dist\_freq, dist\_time, thresh):

# Apply maximum filter to identify local maximums

result = ndimage.maximum\_filter(Y, size=[2 \* dist\_freq + 1, 2 \* dist\_time + 1], mode='constant')

# Create a map of points that satisfy certain conditions

Cmap = np.logical\_and(Y == result, result > thresh)

return Cmap

# Sampling frequency and duration for audio recording

fs = 44000 # Sampling frequency (samples per second)

duration = 14 # Duration of audio recording in seconds

result\_list = []

threshold\_freq = 250 # Minimum frequency for considering a point as a "1"

# Record audio using sounddevice

myrecording = sd.rec(int(duration \* fs), samplerate=fs, channels=1) # Record audio for the given duration

print('recording...')

sd.wait() # Wait for the recording to finish

audio\_data = np.squeeze(myrecording) # Extract audio data from the recorded array

# Define filter parameters for audio signal

lowcut = 4800

highcut = 7200

order = 4

nyquist\_freq = 0.5 \* fs

low = lowcut / nyquist\_freq

high = highcut / nyquist\_freq

b, a = butter(order, [low, high], btype='band') # Design a bandpass filter

filtered\_audio = filtfilt(b, a, audio\_data) # Apply the filter to the audio data

time = np.arange(0, len(filtered\_audio)) / fs # Create a time array for the filtered audio

# Calculate spectrogram of filtered audio

spectrogram=librosa.amplitude\_to\_db(np.abs(librosa.stft(filtered\_audio)), ref=np.max) # Compute the spectrogram

frequencies=librosa.fft\_frequencies(sr=fs, n\_fft=spectrogram.shape[0]) # Compute frequency bins

# Create a list to store results for each row

rows\_list = []

current\_row = {}

# Process the points and create rows

for i, point in enumerate(sorted\_points):

if i % 8 == 0 and i > 0:

# Calculate and append the parity bit to the current row

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

# Append the current row to the list and reset the current row

rows\_list.append(current\_row)

current\_row = {}

if point[0] > threshold\_freq:

current\_row[f'Bit{i % 8}'] = 1

else:

current\_row[f'Bit{i % 8}'] = 0

# Calculate and append the parity bit to the last row (if any)

if current\_row:

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

rows\_list.append(current\_row)

# Convert the list of rows to a DataFrame

result\_df = pd.DataFrame(rows\_list)

# Add additional columns to the DataFrame

result\_df['Sent Parity'] = [1, 0, 1, 0, 0, 0, 0, 1]

result\_df['Match'] = ['=IF($I2=$J2, "Yes", "No")', '=IF($I3=$J3, "Yes", "No")', '=IF($I4=$J4, "Yes", "No")',

'=IF($I5=$J5, "Yes", "No")', '=IF($I6=$J6, "Yes", "No")', '=IF($I7=$J7, "Yes", "No")',

'=IF($I8=$J8, "Yes", "No")', '=IF($I9=$J9, "Yes", "No")', ]

# Save the results to an Excel file

excel\_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')

result\_df.to\_excel(excel\_writer, sheet\_name='Results', index=False)

excel\_writer.close()

# Display the plots

plt.show()

Y = np.abs(librosa.stft(filtered\_audio)) # Compute the short-time Fourier transform of the filtered audio

# Parameters for constellation map computation

dist\_freq = 7 # Distance in frequency bins for finding local maximums

dist\_time = 7 # Distance in time bins for finding local maximums

thresh = 2 # Threshold for determining valid local maximums

# Compute the constellation map

Cmap = compute\_constellation\_map(Y, dist\_freq=dist\_freq, dist\_time=dist\_time, thresh=thresh)

# Apply thresholding to the spectrogram and constellation map

mask = np.abs(spectrogram) < np.max(spectrogram) \* thresh # Create a mask based on the threshold

Cmap[mask] = False # Set values outside the mask to False

# Visualize spectrogram and constellation map

plt.figure(figsize=(14, 7))

plt.subplot(2, 1, 1)

librosa.display.specshow(librosa.amplitude\_to\_db(Y, ref=np.max), sr=fs, x\_axis='time', y\_axis='linear', cmap='magma')

plt.xlabel('Time (s)')

plt.ylabel('Frequency (Hz)')

plt.title('Spectrogram')

# Extract points from the constellation map and sort them

points = np.transpose(np.nonzero(Cmap))

sorted\_points = points[np.lexsort((points[:, 0], points[:, 1]))]

# Create a list to store results for each row

rows\_list = []

current\_row = {}

# Process the points and create rows

for i, point in enumerate(sorted\_points):

if i % 8 == 0 and i > 0:

# Calculate and append the parity bit to the current row

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

# Append the current row to the list and reset the current row

rows\_list.append(current\_row)

current\_row = {}

if point[0] > threshold\_freq:

current\_row[f'Bit{i % 8}'] = 1

else:

current\_row[f'Bit{i % 8}'] = 0

# Calculate and append the parity bit to the last row (if any)

if current\_row:

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

rows\_list.append(current\_row)

# Convert the list of rows to a DataFrame

result\_df = pd.DataFrame(rows\_list)

# Add additional columns to the DataFrame

result\_df['Sent Parity'] = [1, 0, 1, 0, 0, 0, 0, 1]

result\_df['Match'] = ['=IF($I2=$J2, "Yes", "No")', '=IF($I3=$J3, "Yes", "No")', '=IF($I4=$J4, "Yes", "No")',

'=IF($I5=$J5, "Yes", "No")', '=IF($I6=$J6, "Yes", "No")', '=IF($I7=$J7, "Yes", "No")',

'=IF($I8=$J8, "Yes", "No")', '=IF($I9=$J9, "Yes", "No")', ]

# Save the results to an Excel file

excel\_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')

result\_df.to\_excel(excel\_writer, sheet\_name='Results', index=False)

excel\_writer.close()

# Display the plots

plt.show()

plt.ylim([lowcut, highcut])

plt.subplot(2, 1, 2)

plt.tight\_layout()

librosa.display.specshow(Cmap, sr=fs, x\_axis='time', y\_axis='linear', cmap='magma')

plt.xlabel('Time (s)')

plt.ylabel('Frequency (Hz)')

plt.title('Constellation Map')

plt.ylim([lowcut, highcut])

plt.tight\_layout()

# Extract points from the constellation map and sort them

points = np.transpose(np.nonzero(Cmap))

sorted\_points = points[np.lexsort((points[:, 0], points[:, 1]))]

# Create a list to store results for each row

rows\_list = []

current\_row = {}

# Process the points and create rows

for i, point in enumerate(sorted\_points):

if i % 8 == 0 and i > 0:

# Add additional columns to the DataFrame

result\_df['Sent Parity'] = [1, 0, 1, 0, 0, 0, 0, 1]

result\_df['Match'] = ['=IF($I2=$J2, "Yes", "No")', '=IF($I3=$J3, "Yes", "No")', '=IF($I4=$J4, "Yes", "No")',

'=IF($I5=$J5, "Yes", "No")', '=IF($I6=$J6, "Yes", "No")', '=IF($I7=$J7, "Yes", "No")',

'=IF($I8=$J8, "Yes", "No")', '=IF($I9=$J9, "Yes", "No")', ]

# Save the results to an Excel file

excel\_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')

result\_df.to\_excel(excel\_writer, sheet\_name='Results', index=False)

excel\_writer.close()

# Display the plots

plt.show()

# Calculate and append the parity bit to the current row

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

# Append the current row to the list and reset the current row

rows\_list.append(current\_row)

current\_row = {}

if point[0] > threshold\_freq:

current\_row[f'Bit{i % 8}'] = 1

else:

current\_row[f'Bit{i % 8}'] = 0

# Calculate and append the parity bit to the last row (if any)

if current\_row:

parity\_bit = sum(current\_row.values()) % 2

current\_row['Parity'] = parity\_bit

rows\_list.append(current\_row)

# Convert the list of rows to a DataFrame

result\_df = pd.DataFrame(rows\_list)

# Add additional columns to the DataFrame

result\_df['Sent Parity'] = [1, 0, 1, 0, 0, 0, 0, 1]

'=IF($I5=$J5, "Yes", "No")', '=IF($I6=$J6, "Yes", "No")', '=IF($I7=$J7, "Yes", "No")',

'=IF($I8=$J8, "Yes", "No")', '=IF($I9=$J9, "Yes", "No")', ]

result\_df['Match'] =

['=IF($I2=$J2, "Yes", "No")', '=IF($I3=$J3, "Yes", "No")', '=IF($I4=$J4, "Yes", "No")',

'=IF($I5=$J5, "Yes", "No")', '=IF($I6=$J6, "Yes", "No")', '=IF($I7=$J7, "Yes", "No")',

'=IF($I8=$J8, "Yes", "No")', '=IF($I9=$J9, "Yes", "No")', ]

# Save the results to an Excel file

excel\_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')

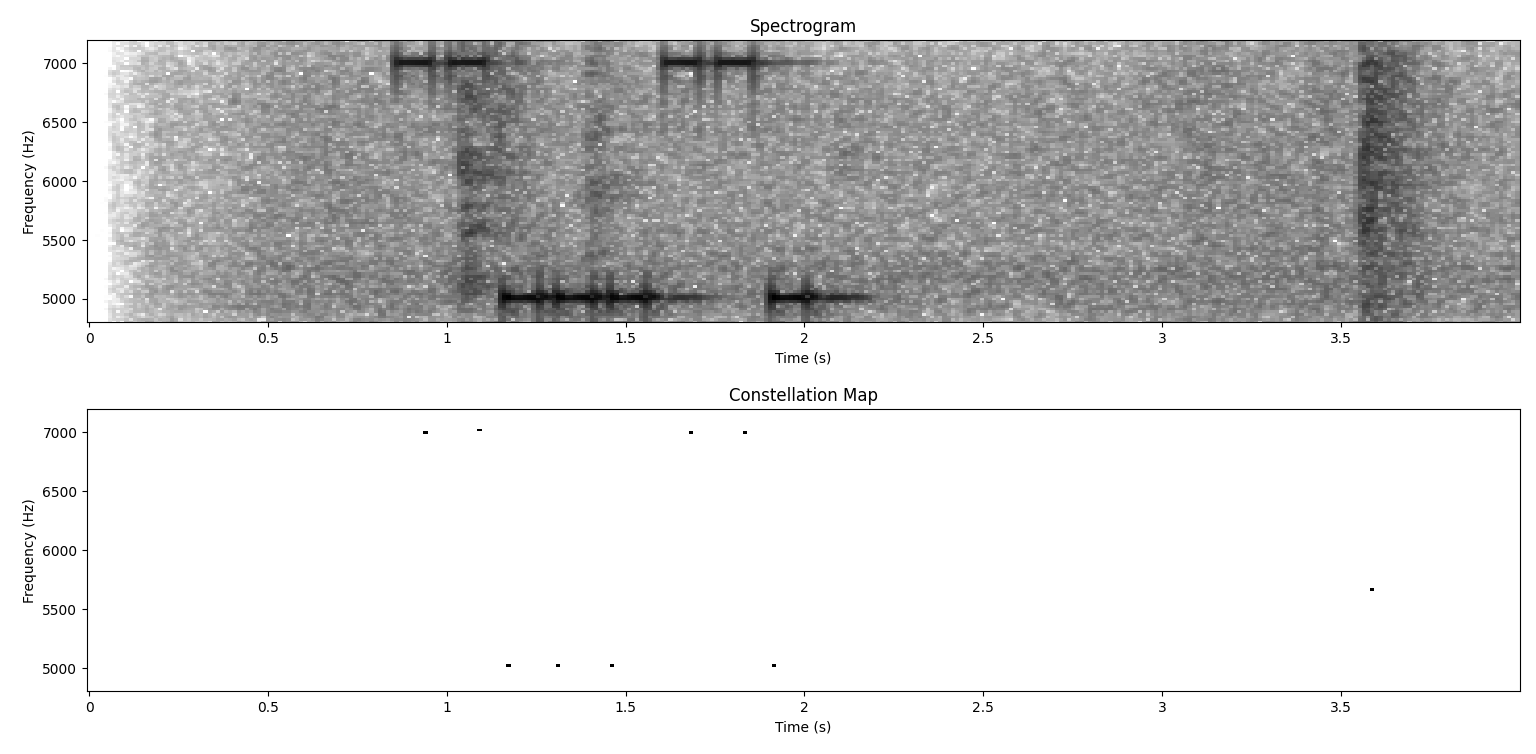
result\_df.to\_excel(excel\_writer, sheet\_name='Results', index=False)

excel\_writer.close()

# Display the plots

plt.show()

|  |  |
| --- | --- |
| visual data | |
|  |  |
|  | |

A blue and green image of a blue and yellow image

Description automatically generated with medium confidence

Figure 2: Mapping of constellation points using a spectrogram, showing the accuracy of plots. The byte generated is - 11000110

Figure : Mapping of constellation points using a spectrogram, showing the accuracy of plots. The byte generated is - 01101001



Figure 3: Continuous, constant data transmission to test for accuracy of reception and decoding.

|  |
| --- |
| conclusion |
| This project successfully applies a systematic methodology to analyze audio signals, extracting meaningful insights through a well-structured code. By meticulously adhering to the methodology's guidelines, the code efficiently captures and processes sound data. The interdependent nature of parameters like sampling frequency and duration is managed adeptly, ensuring the accuracy of signal representation. Through a cohesive fusion of theory and practical implementation, this project demonstrates the efficacy of the proposed methodology in real-world applications, underscoring its value in the field of audio signal analysis. |