Aplikasi Singular Value Decomposition (SVD) untuk Memodelkan Fitur Noise Cancellation dan Mode Transparansi pada Earphone Modern

Maheswara Bayu Kaindra - 13523015
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523015@itb.ac.id, [2]kaindramaheswara11@gmail.com

***Abstract*—Many modern earphones now include noise cancellation and transparency mode features, which need accurate sound processing. This research aims to study and apply the Singular Value Decomposition (SVD) method to model these features effectively using Python modules. The approach begins by representing the sound signal as a matrix that contains both the primary signal and noise components. The SVD method is used to separate the primary components from noise, enabling noise removal in the noise-cancelling feature and the adjustment of environmental sound level in transparency mode.**

***Keywords*— noise cancellation, Python, Singular Value Decomposition, transparency mode.**

I. Pendahuluan

Dalam beberapa tahun terakhir, teknologi audio berkembang pesat, salah satunya teknologi pada *earphone* modern. Fitur yang menjadi standar bagi *earphone* modern adalah *noise cancellation* dan mode transparansi. Kedua fitur tersebut memungkinkan pengguna untuk melemahkan atau menguatkan suara lingkungan sekitar tanpa mengorbankan kualitas audio utama. Kedua fitur ini membutuhkan pemrosesan suara yang presisi untuk menghasilkan kustomisasi audio yang akurat.

Pemrosesan suara dalam kedua fitur ini melibatkan pemisahan komponen utama suara dari *noise* atau gangguan suara eksternal lainnya. Untuk melakukan hal tersebut, digunakan metode Singular Value Decomposition (SVD), sebuah teknik dekomposisi matriks untuk memisahkan matriks menjadi dua buah matriks ortogonal dan sebuah matriks diagonal yang memuat nilai singular. SVD memungkinkan pemrosesan representasi suara dalam bentuk matriks yang mencakup sinyal suara dan *noise*. Dengan metode ini, komponen *noise* dapat dihilangkan atau diperjelas, sehingga suara lingkungan dapat disaring atau diperkuat sesuai kebutuhan pengguna.

Penelitian ini bertujuan untuk mempelajari dan menerapkan metode SVD dalam pemodelan fitur *noise cancellation* dan mode transparansi dan memodelkannya dengan simulasi berbasis Python. Dengan demikian, penelitian ini diharapkan dapat memperluas pemahaman baik teoritis maupun praktikal mengenai penerapan SVD dalam dunia nyata, khususnya pada perkembangan teknologi audio.

Makalah ini akan berfokus pada pembahasan teoritis mengenai SVD dan bagaimana pengimpelementasian SVD untuk memodelkan fitur *noise cancellation* dan mode transparansi pada *earphone* modern menggunakan modul-modul Python.

II. Dasar Teori

A. Noise Cancellation

Noise cancellation merupakan fitur pada perangkat audio modern yang memungkinkan pengguna untuk mendengar audio secara lebih murni, tanpa gangguan kebisingan lingkungan sekitar.

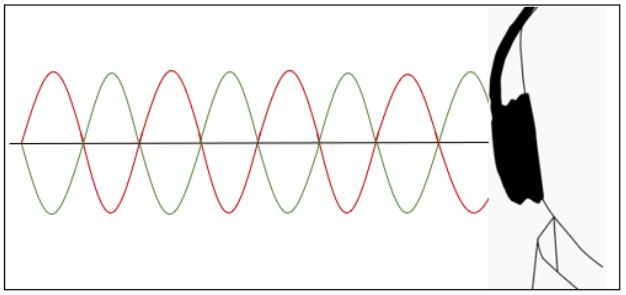

Fig 2.1 Noise Cancelling : Noise vs Anti-Noise
Sumber:
https://www.sony.co.id/en/electronics/support/articles/00203389

Noise cancellation bekerja dengan tiga tahapan utama, yaitu pengumpulan sinyal suara, pemrosesan sinyal, dan pembatalan *noise*. Menurut Sony (2023), pembatalan *noise* dilakukan dengan membuat sinyal suara yang berkebalikan dengan *noise* untuk menetralisir suara dari lingkungan luar.

B. Mode Transparansi

Mode transparansi, atau sering disebut sebagai *transparency mode* merupakan fitur pada perangkat audio modern yang memungkinkan pengguna mendengar kebisingan lingkungan sekitar lebih jelas sambil tetap menikmati audio utama. Fitur ini berguna ketika pengguna harus tetap memperhatikan kondisi lingkungan sekitar ketika mendengarkan audio. Seperti *noise*

cancellation, mode transparansi juga bekerja dengan tiga tahapan utama, yaitu pengumpulan sinyal suara, pemrosesan sinyal, dan mengeluarkan audio utama bersamaan dengan audio noise sesuai pengaturan.

Menurut Sound Core (2024), untuk menjalankan mode transparansi, *earphone* memiliki mikrofon yang selalu menangkap suara lingkungan sekitar. Suara lingkungan sekitar tersebut digabungkan dengan komposisi tertentu dengan audio utama, sehingga pengguna tetap dapat mendengar suara lingkungan sekitar ketika menggunakan *earphone*.

C. Nilai Eigen dan Vektor Eigen

Nilai eigen atau disimbolkan sebagai λ dan vektor (tidak nol) eigen x merupakan pengali suatu matriks persegi A berdimensi $n \times n$ yang menyebabkan

$$Ax = \lambda x \, ()$$

Nilai eigen merupakan nilai representasi atau nilai karakteristik dari matriks A. Vektor eigen dari matriks A juga dapat didefinisikan sebagai suatu matriks kolom yang apabila dikalikan dengan matriks A akan menghasilkan vektor yang merupakan kelipatan vektor itu sendiri dengan nilai eigen.

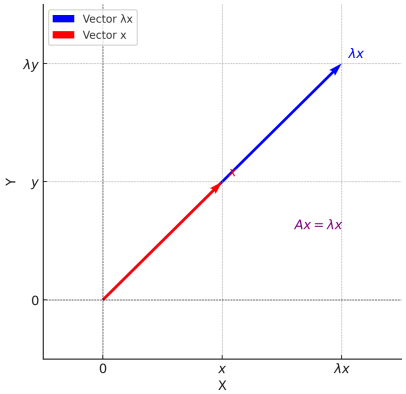

Fig. 2.2 Grafik Perkalian Vektor Eigen Matriks A Diproses dengan Pandas (salah satu modul Python)

Operasi $Ax = \lambda x$ dapat menyusutkan atau meregangkan vektor x dengan arah yang sama jika $\lambda \geq 0$ dan berlawanan arah jika $\lambda < 0$.

Untuk menghitung nilai eigen dan vektor eigen, berikut adalah tahapan yang dapat dilakukan menurut Munir (2024).

$$Ax = \lambda x \, (1)$$
$$IAx = I\lambda x \, (2)$$
$$Ax = I\lambda x \, (3)$$
$$(I\lambda - A)x = 0 \, (4)$$

$x = 0$ merupakan solusi trivial dari $(I\lambda - A)x = 0$. Untuk mendapatkan solusi lain,

$$det(\lambda I - A) = 0 \, (5)$$

Persamaan tersebut disebut persamaan karakteristik matriks A. Nilai-nilai λ yang didapatkan dari persamaan karakteristik disebut akar-akar karakteristik. Substitusi akar-akar karakteristik untuk ke persamaan () untuk mendapatkan vektor eigen x.

D. Singular Value Decomposition (SVD)

Singular Value Decomposition merupakan salah satu metode untuk memfaktorkan matriks berukuran $m \times n$ menjadi matriks U, Σ, dan V sedemikian sehingga

$$A = U\Sigma V^T \, (6)$$

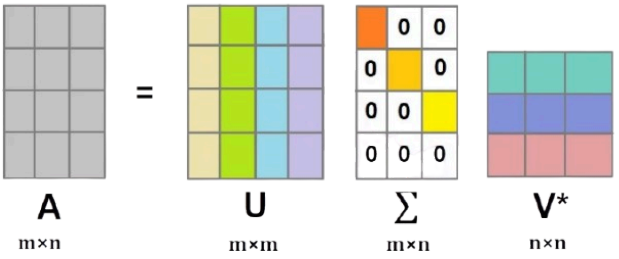

Fig 2. Singular Value Decomposition
Sumber :
https://www.askpython.com/wp-content/uploads/2020/11/SVD-1.jpg.webp

U = matriks orthogonal $m \times m$,
V = matriks orthogonal $n \times n$.
Σ = matriks $m \times n$ dengan nilai-nilai singular matriks A sebagai elemen-elemen diagonal utamanya.

Salah satu cara dekomposisi matriks dengan SVD adalah dengan mengikuti tahapan-tahapan berikut (Munir, 2024).

1. Menentukan vektor-vektor singular kanan yang berkorespondensi dengan nilai-nilai eigen dari matriks A^TA, sebut saja $v_1, v_2, ..., v_n$ dan melakukan normalisasi dengan membagi setiap vektor dengan panjangnya.
2. Menentukan vektor-vektor singular kiri, yaitu $u_1, u_2, ..., u_k$ dengan membagi AV_i dengan nilai singular tidak nol σ_i untuk $1 \leq i \leq k$ dan melakukan normalisasi dengan membagi setiap vektor dengan panjangnya.
3. Membentuk matriks Σ berukuran $m \times n$ dengan nilai-nilai singular tidak nol dengan urutan menurun sebagai elemen-elemen diagonal utamanya.

E. Numpy, dan Scipy

Numpy dan *Scipy* merupakan modul-modul pendukung bahasa pemrograman Python yang digunakan dalam penelitian ini. Menurut situs resmi *Numpy*, *Numpy* merupakan paket pendukung fundamental bahasa pemrograman Python untuk melakukan komputasi ilmiah seperti pengolahan matriks, array, maupun perhitungan statistik. Pada penelitian ini, *Numpy* digunakan sebagai alat utama untuk melakukan perhitungan seperti membentuk, memanipulasi, dan mengelola matriks.

Menurut situs resmi Pypi (2024), *Scipy* (Scientific Python) merupakan pustaka bahasa pemrograman Python yang dirancang untuk mendukung operasi matematika, komputasi ilmiah, dan teknik optimasi. *Scipy* memperluas kemampuan *Numpy* dengan menambahkan modul-modul untuk menyelesaikan masalah kompleks seperti pemrosesan sinyal, integrasi numerik, optimasi,

statistik, dan pemodelan data. Pada Penelitian ini, *Scipy* berperan sebagai pengolah utama file .wav dengan modul *Scipy.io.wavfile*.

III. Pembahasan

A. Pendekatan Implementasi Noise Cancellation

Untuk menggunakan modul Python dalam pemodelan, digunakan pendekatan sederhana untuk memodelkan fitur *noise cancellation* dan mode transparansi, yaitu dengan mengurangi noise pada audio file input dan menyimpan file audio output pada suatu direktori. Dengan ini, audio yang diolah tidak bersifat *live* atau siaran langsung, tetapi bersifat *pre-recorded* dan mengandung kebisingan.

Data file audio bersumber dari salah satu *dataset* tanpa lisensi pada kaggle.com yang digabungkan dengan audio *noise* berupa suara hujan (sumber tercantum).

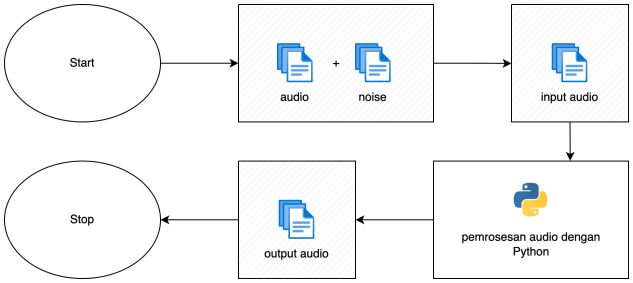

Fig. 3.1 Diagram Pendekatan Noise Cancellation
Dibuat dengan draw.io
Sumber gambar logo Python:
https://w7.pngwing.com/pngs/447/294/png-transparent-python-javascript-logo-clojure-python-logo-blue-angle-text-thumbnail.png

Pemodelan *noise cancellation* tersebut dibagi menjadi beberapa tahapan:

1. Memuat file audio
2. Mengubah sinyal audio yang *stereo* menjadi *mono*
3. Menormalkan (normalisasi) data audio
4. Merekonstruksi ulang data audio dalam bentuk matriks
5. Mengaplikasikan SVD pada matriks
6. Melakukan *filtering* pada noise
7. Merekonstruksi matriks menjadi sinyal audio
8. Menyimpan file audio *output* ke direktori tujuan

B. Pendekatan Implementasi Mode Transparansi

Untuk mengimplementasikan mode transparansi dengan Python, digunakan pendekatan dengan *pre-recorded audio* yang terdiri atas audio input, dan audio noise.

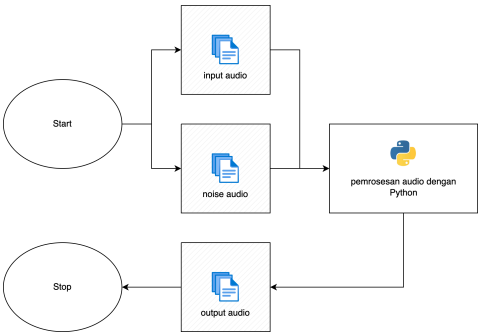

Fig 3.2 Diagram Pendekatan Transparency Mode
Dibuat dengan draw.io
Sumber gambar: draw.io

Layaknya cara kerja mode transparansi pada *earphone* modern, sebagian volume audio noise digabungkan dengan audio utama untuk memungkinkan pengguna mendengar suara lingkungan sekitar ketika mendengar audio utama.

Data file audio bersumber dari salah satu *dataset* tanpa lisensi pada kaggle.com dengan audio *noise* berupa suara hujan (sumber tercantum).

Pemodelan mode transparansi dibagi menjadi tahapan-tahapan utama:

1. Memuat file audio input dan noise
2. Mengubah format audio stereo menjadi mono
3. Menormalkan audio input dan noise
4. Mengatur panjang noise agar sesuai dengan audio input (tidak ada pada mode transparansi sungguhan, namun diperlukan untuk melakukan perhitungan)
5. Merekonstruksi sinyal audio input dan noise menjadi matriks
6. Melakukan SVD pada kedua matriks dan menyimpan hasilnya pada variabel lain (disebut matriks utama)
7. Menyamakan dimensi U dan V^T untuk memungkinkan perhitungan (menghindari *error*)
8. Menyetarakan dimensi setiap komponen SVD noise sesuai matriks utama
9. Merekonstruksi kembali noise yang sudah diatur dimensinya menjadi matriks utuh
10. Menjumlahkan matriks utama dengan matriks noise terekonstruksi
11. Mengembalikan audio yang sudah tergabung ke skala awal (denormalisasi)
12. Menyimpan file audio output ke direktori tujuan

E. Pendekatan dan Implementasi Singular Value Decomposition

Singular Value Decomposition (SVD) menggunakan pendekatan Munir (2024). Implementasi SVD juga tereferensi dari salah satu modul *Scipy* yaitu *svd*. Secara umum, implementasi SVD dibagi menjadi tahapan-tahapan berikut (tahapan detail tercantum pada Dasar Teori):

1. Menghitung A^TA dan nilai eigen untuk mendapatkan V (vektor singular kanan)
2. Mengurutkan nilai eigen dan vektor eigen dari besar ke kecil

3. Menghitung nilai singular
4. Menghitung dan menormalkan matriks *U*
5. Memperluas *U* dan *V* apabila diperlukan (sebagai tambahan fitur berdasarkan referensi salah satu modul Scipy, yaitu *scipy.linalg.svd*).
6. Mentranspose *V* untuk menghasilkan V^T.

Implementasi SVD dengan menggunakan Python terdapat pada file *singular_value_decomposition.py*, dengan rangkuman kode:

Menghitung A^TA, nilai-nilai eigen, dan matriks V

```
#  Hitung  A^T  A  dan  nilai  eigen  untuk
mendapatkan V
    ATA = np.dot(A.T, A)
    eigen_values_V, V = np.linalg.eigh(ATA)
```

Mengurutkan nilai dan vektor eigen dari besar ke kecil

```
# Urutkan nilai eigen dan vektor eigen dari
besar ke kecil
    sorted_indices =
np.argsort(eigen_values_V)[::-1]
    eigen_values_V =
eigen_values_V[sorted_indices]
    V = V[:, sorted_indices]
```

Menghitung nilai-nilai singular

```
# Hitung nilai singular (akar nilai eigen)
    singular_values =
np.sqrt(eigen_values_V)
```

Menghitung dan menormalkan matriks U

```
# Hitung U menggunakan A V / Sigma
    U = np.zeros((A.shape[0],
len(singular_values)))
    for i in range(len(singular_values)):
        if singular_values[i] > 1e-10:
# Hindari pembagian dengan nol
            U[:, i] = np.dot(A, V[:, i]) /
singular_values[i]
# Normalisasi U
    U = U / np.linalg.norm(U, axis=0)
```

Memperluas U dan V (tidak digunakan, namun tereferensi dari Scipy)

```
# Memperluas U dan V jika
full_matrices=True
    if full_matrices:
        m, n = A.shape
        if m > n:
            extra_cols = np.eye(m)[:, n:]
            U = np.hstack((U, extra_cols))
        elif n > m:
            extra_cols = np.eye(n)[m:, :]
            V = np.hstack((V, extra_cols.T))
```

Mentranspose V untuk menghasilkan V^T

```
# Transpose V untuk menghasilkan V^T
    Vt = V.T
```

Mengembalikan U, Σ, V^T

```
return U, singular_values, Vt
```

C. Implementasi Noise Cancellation

Fitur *noise cancellation* diimplementasi dengan bahasa pemrograman Python dengan bantuan modul-modul tambahan seperti *Numpy* dan *Scipy*. Sesuai dengan pendekatan implementasinya, berikut merupakan kutipan kode implementasi *noise cancellation* yang terdapat pada file *noise_cancellation.py*.

Memuat file audio

```
# Memuat file audio
    if not os.path.exists(input_file):
        print("File not found")
        return
    rate, data = wav.read(input_file)
```

Mengubah sinyal audio stereo menjadi mono

```
# Mengonversi audio stereo menjadi mono
    if data.ndim > 1:
        data = data.sum(axis=1)
```

Menormalkan (normalisasi) data audio

```
# Normalisasi sinyal audio
    data = data.astype(np.float64)
    data /= np.max(np.abs(data))
```

Merekonstruksi ulang data audio dalam bentuk matriks

```
# Membentuk sinyal audio dalam bentuk
matriks (agar bisa dilakukan SVD)
    chunk_size = 1024
    num_chunks = len(data) // chunk_size
    audio_matrix =
np.reshape(data[:num_chunks * chunk_size],
(num_chunks, chunk_size))
    print("Audio Matrix: \n", audio_matrix)
    print("\n")
```

Mengaplikasikan SVD pada matriks

```
# Melakukan SVD pada matriks audio
    u, sigma, v_transposed =
svd(audio_matrix, full_matrices=False)
```

Melakukan filtering pada noise

```
# Mengeluarkan noise dari sigma
    anti_sigma = np.where(sigma < threshold,
sigma, 0)
    filtered_sigma = sigma - anti_sigma
filtered_sigma)
```

Merekonstruksi matriks menjadi sinyal audio

```
# Rekonstruksi audio
    filtered_audio = np.dot(u,
np.dot(np.diag(filtered_sigma),
v_transposed)).flatten()
    # Mengembalikan audio ke rentang aslinya
    filtered_audio = (filtered_audio *
32767).astype(np.int16)
```

Menyimpan file audio output ke direktori tujuan

```
# Menyimpan audio hasil noise cancellation
    wav.write(output_file, rate,
filtered_audio)
    print("Noise Cancellation Done")
```

D. Implementasi Mode Transparansi

Mode transparansi diimplementasi dengan bahasa pemrograman Python dengan bantuan *Numpy* dan *Scipy*.

Sesuai pendekatan yang sudah dibahas, berikut merupakan kutipan kode implementasi mode transparansi yang tercantum pada file *transparency_mode.py*.

Memuat file audio input dan noise

```
# Memuat file audio input dan noise
    main_rate, main_data =
wav.read(main_audio)
    noise_rate, noise_data =
wav.read(noise_audio)
```

Mengubah sinyal audio stereo menjadi mono

```
# Mengubah format audio stereo menjadi mono
    if main_data.ndim > 1:
        main_data = main_data.mean(axis=1)
    if noise_data.ndim > 1:
        noise_data = noise_data.mean(axis=1)
```

Menormalkan audio input dan noise

```
# Menormalkan audio input dan noise
    main_data = main_data.astype(np.float64)
/ np.max(np.abs(main_data))
    noise_data =
noise_data.astype(np.float64) /
np.max(np.abs(noise_data))
```

Mengatur panjang noise agar sesuai dengan panjang audio input

```
# Mengatur panjang noise agar sesuai dengan
panjang audio input
    if len(noise_data) > len(main_data):
        noise_data =
noise_data[:len(main_data)]
    elif len(noise_data) < len(main_data):
        noise_data = np.pad(noise_data, (0,
len(main_data) - len(noise_data)),
mode='constant', constant_values=0)
```

Pengaturan panjang noise ini sangat penting untuk menyesuaikan panjang noise dengan panjang audio input. Noise tidak akan melebihi panjang audio input, namun noise akan digunakan seluruhnya apabila ia sama panjang atau lebih pendek dari audio input.

Merekonstruksi audio input dan noise dalam bentuk matriks

```
chunk_size = 1024
    num_chunks = min(len(main_data),
len(noise_data)) // chunk_size
    main_matrix =
np.reshape(main_data[:num_chunks *
chunk_size], (num_chunks, chunk_size))
    noise_matrix =
np.reshape(noise_data[:num_chunks *
chunk_size], (num_chunks, chunk_size))
```

Melakukan SVD pada matriks audio input dan noise

```
# Melakukan SVD pada matriks audio input
dan noise
    main_U, main_Sigma, main_Vt =
svd(main_matrix)
    noise_U, noise_Sigma, noise_Vt =
svd(noise_matrix)
```

Menyamakan dimensi Σ pada matriks utama dengan Σ pada matriks noise

```
# Menyamakan dimensi main_Sigma dan
noise_Sigma
    Sigma_noise_matrix =
np.zeros((noise_U.shape[1],
noise_Vt.shape[0]))
    np.fill_diagonal(Sigma_noise_matrix,
noise_Sigma[:min(len(noise_Sigma),
Sigma_noise_matrix.shape[0])])
```

Matriks Σ merupakan utama yang digunakan untuk operasi-operasi setiap fungsi pada penelitian ini. Untuk memastikan bahwa operasi-operasi berikutnya dapat dilakukan, harus dipastikan bahwa Σ pada matriks utama dan Σ pada matriks noise memiliki dimensi yang sama.

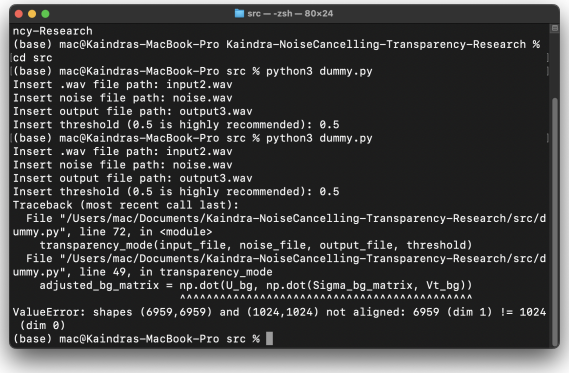

Fig 3.3 Error Karena Perbedaan Dimensi Matriks

Menyesuaikan volume noise

```
# Menyesuaikan volume noise
    Sigma_noise_matrix *= noise_volume
```

Menyamakan dimensi matriks U dan V^T noise

```
# Menyamakan dimensi noise_U dan noise_Vt
    noise_U = noise_U[:, :main_U.shape[1]]
    noise_Vt = noise_Vt[:main_Vt.shape[0],
:]
```

Merekonstruksi noise yang sudah sesuai

```
# Merekonstruksi noise yang disesuaikan
    adjusted_noise_matrix = np.dot(noise_U,
np.dot(Sigma_noise_matrix, noise_Vt)
```

Menggabungkan audio input dan noise

```
# Menggabungkan audio input dan noise
    combined_matrix = main_matrix +
adjusted_noise_matrix
```

Merekonstruksi audio gabungan

```
# Merekonstruksi audio gabungan
    combined_audio =
combined_matrix.flatten()
    combined_audio = (combined_audio *
32767).astype(np.int16)
```

Menyimpan audio output pada direktori

```
# Menyimpan audio output pada direktori
yang ditentukan
    wav.write(output_path, main_rate,
combined_audio)
```

E. Eksekusi Program

Pada *repository*, sudah disediakan file-file audio uji coba untuk mencoba fitur *noise cancelling* dan transparansi mode. Untuk mengeksekusi program, terdapat beberapa tahapan utama:

1. Mengunduh keperluan modul untuk Python

2. Berpindah direktori ke *src*
3. Menjalankan kode program sesuai keinginan, contohnya *python3 transparency_mode.py*

Ketika program dieksekusi, pengguna diminta untuk memasukkan nama file input, file noise (pada mode transparansi), file output, dan skala.

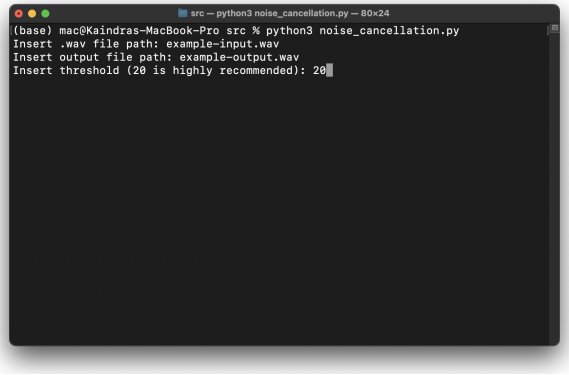

Fig 3.4 Program Meminta Masukan Pengguna

Apabila tidak terjadi masalah saat eksekusi, program akan memberikan pemberitahuan bahwa proses telah selesai.

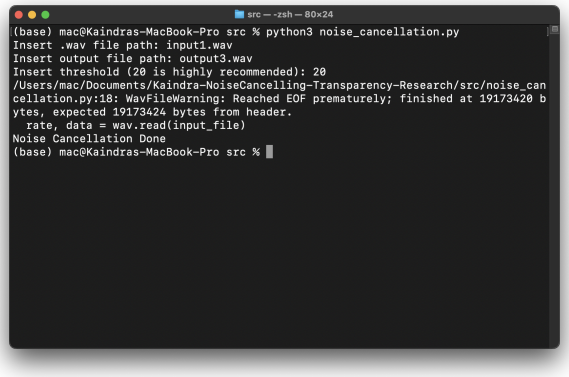

Fig 3.5 Pemberitahuan Proses Selesai

File output akan disimpan pada direktori *output-files* secara otomatis.

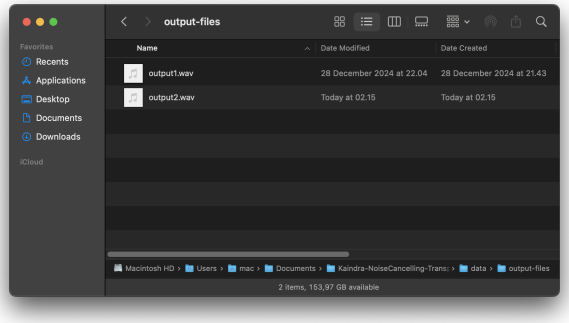

Fig 3.6 File Tersimpan pada Folder output-files

Tahapan dan detail lengkap lain mengenai eksekusi program tercantum dalam *repository*.

F. Visualisasi

Untuk memvisualisasikan data audio dan memberi gambaran mengenai bagaimana proses *noise cancellation* dan mode transparansi mengubah struktur file audio, digunakan format visualisasi *waveform* karena berhubungan dengan bentuk audio keseluruhan.

Berikut merupakan *waveform* dari file input *input1.wav* dan *output1.wav*, yaitu file input untuk melakukan uji coba *noise cancellation* dan file output hasil pemrosesan *input1.wav*.

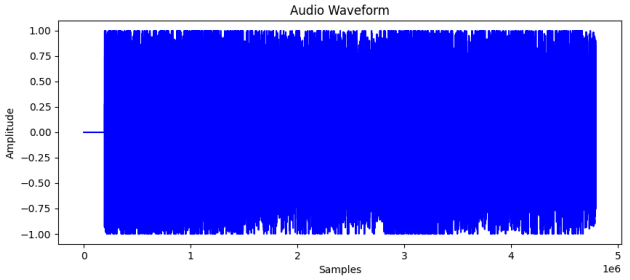

Fig 3.7 Waveform input1.wav
Dibuat dengan matplotlib

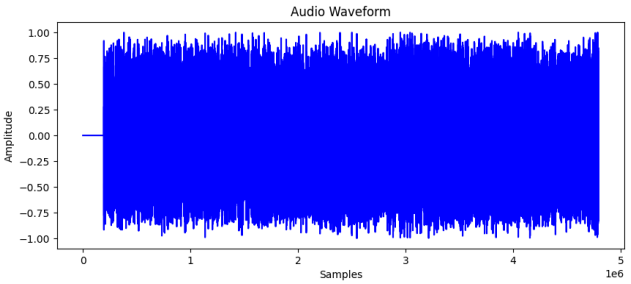

Fig 3.8 Waveform output1.wav
Dibuat dengan matplotlib

Dapat dilihat bahwa terjadi reduksi signifikan pada intensitas audio keseluruhan setelah dilakukannya *noise cancellation* (sesuai hipotesis bahwa *noise cancellation* dapat diimplementasikan dengan SVD).

Berikut merupakan *waveform* dari *input2.wav* dan *output2.wav* yang merupakan file-file uji coba mode transparansi.

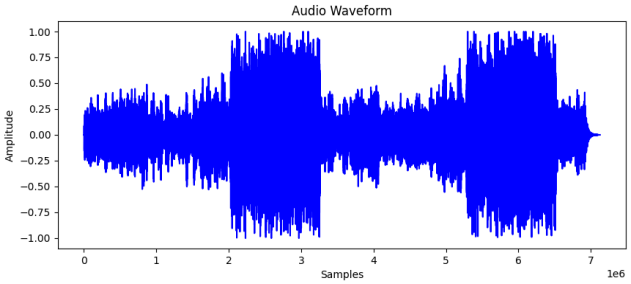

Fig 3.9 Waveform input2.wav
Dibuat dengan matplotlib

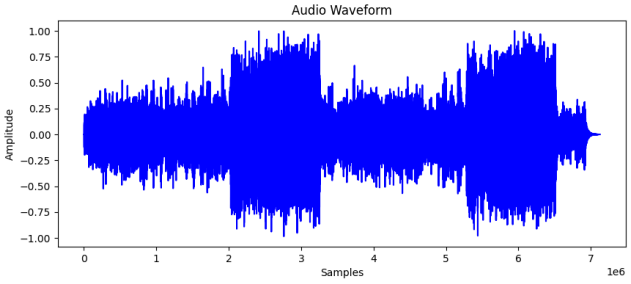

Fig 3.10 Waveform output2.wav
Dibuat dengan matplotlib

Dapat dilihat terjadi penebalan pada intensitas audio *output2.wav*, yang berarti terjadi penambahan audio noise

pada audio utama (sesuai hipotesis, yaitu SVD dapat digunakan untuk mengimplementasikan mode transparansi).

IV. Kesimpulan

Penelitian ini berhasil memodelkan fitur *noise cancellation* dan mode transparansi pada *earphone* modern dengan menerapkan metode Singular Value Decomposition. Meskipun tidak sempurna, metode ini terbukti efektif dalam memodelkan fitur-fitur tersebut.

Pada *noise cancellation*, SVD digunakan untuk menghilangkan *noise* dari sinyal audio utama tanpa mengorbankan audio utama. SVD juga dapat digunakan untuk memodelkan mode transparansi untuk mengatur tingkat transparansi suara *noise* pada audio utama.

Oleh karena itu, dapat disimpulkan bahwa metode SVD tidak hanya relevan secara teoritis, tetapi juga dapat digunakan secara praktis di dunia nyata, khususnya teknologi pengolahan audio.

V. Lampiran

Seluruh *source code*, data, dokumentasi, dan *test file* yang digunakan dalam penelitian ini dapat diakses pada tautan ini atau dengan mengunjungi https://github.com/MaheswaraKaindra/Kaindra-NoiseCancelling-Transparency-Research.git

Video penjelasan umum penelitian ini dapat dikases pada tautan ini atau dengan mengunjungi https://youtu.be/sIPETN1XPWM

VI. Kata Penutup

Penulis mengucapkan terima kasih kepada pihak-pihak yang telah membantu penulis selama melaksanakan perkuliahan Aljabar Linear dan Geometri tahun 2024/2025, pihak-pihak yang membantu penulis memahami konsep Algoritma dan Struktur Data, dan pihak-pihak yang memberi saran kepada penulis mengenai penulisan makalah ini.

Penulis menyadari bahwa makalah ini masih jauh dari sempurna, terutama pada pengimplementasian *noise cancellation*. Oleh karena itu, *repository* bersifat terbuka bagi audiens di seluruh dunia untuk ruang penyempurnaan. Penulis berharap penelitian ini dapat memberi semangat baru bagi mahasiswa dan pelajar dengan membuktikan bahwa konsep teoritis dapat digunakan secara praktikal dalam kehidupan sehari-hari.

Referensi

[1] Munir, Rinaldi. (2024). *Nilai Eigen dan Vektor Eigen (Bagian 1)*. Diakses 27 Desember 2024.
[2] Munir, Rinaldi. (2024). *Nilai Eigen dan Vektor Eigen (Bagian 2)*. Diakses 27 Desember 2024.
[3] Munir, Rinaldi. (2024). *Singular Value Decomposition (Bagian 1)*. Diakses 27 Desember 2024.
[4] Munir, Rinaldi. (2024). *Singular Value Decomposition (Bagian 2)*. Diakses 27 Desember 2024
[5] NumPy. *Numpy Documentation*. Diakses 31 Desember 2024.
[6] PyPi. (2024). *scipy 1.14.1*. Diakses 31 Desember 2024.
[7] Sound Core. (2024). *What is transparency Mode on Earbuds?*. Diakses 27 Desember 2024.
[8] Sony. (2023). *What is Noise-Cancelation and what can I expect?*. Diakses 27 Desember 2024.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 27 Desember 2024

Maheswara Bayu Kaindra - 13523015