PENERAPAN METODE MEL-FREQUENCY CEPSTRAL COEFFICIENTS DAN K-MEANS CLUSTERING UNTUK PENGENALAN PEMBICARA

LAPORAN TUGAS AKHIR

Diajukan sebagai syarat untuk menyelesaikan
Program Studi Strata-1 Departemen Teknik Informatika

Disusun oleh:

DANIEL CHRISTIAN T.

1111010

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
School of Telematics

DEPARTEMEN TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2015

INSTITUT TEKNOLOGI HARAPAN BANGSA
School of Telematics

DEPARTEMEN TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA

LEMBAR PENGESAHAN

PENERAPAN METODE MEL-FREQUENCY CEPSTRAL COEFFICIENTS DAN K-MEANS CLUSTERING UNTUK PENGENALAN PEMBICARA

Disusun oleh:

Nama: Daniel Christian Tirtabudi

NIM : 1111010

Telah Diperiksa dan Disetujui
Sebagai Tugas Akhir Departemen Teknik Informatika
Institut Teknologi Harapan Bangsa

Bandung, Agustus 2015

Disetujui,	Diketahui,
Dosen Pembimbing I	Dosen Pembimbing II
Elisafina Siswanto, S.T., M.T.	**Oetomo Sudjana, S.T., M.T.**
NIK. 111033	**NIK. 111009**

PERNYATAAN HASIL KARYA PRIBADI

Saya yang bertanda tangan di bawah ini:

Nama : Daniel Christian Tirtabudi

NIM : 1111010

Dengan ini menyatakan bahwa laporan Tugas Akhir dengan judul: **"PENERAPAN METODE MEL-FREQUENCY CEPSTRAL COEFFICIENTS DAN K-MEANS CLUSTERING UNTUK PENGENALAN PEMBICARA"** adalah hasil pekerjaan saya dan seluruh ide, pendapat atau materi dari sumber lain telah dikutip dengan cara penulisan referensi yang sesuai.

Pernyataan ini saya buat dengan sebenar-benarnya dan jika pernyataan ini tidak sesuai dengan kenyataan maka saya bersedia menanggung sanksi yang akan dikenakan pada saya.

Bandung, Agustus 2015

Yang membuat pernyataan,

Daniel Christian Tirtabudi

ABSTRAK

Suara manusia dapat dimanfaatkan dalam berbagai bidang pekerjaan atau bidang keamanan. Sistem pengenalan pembicara dapat diimplementasikan dalam bidang keamanan, seperti untuk proses pembuatan kata sandi atau untuk mengenali suara pelaku kejahatan. Sistem ini berbasis pada *input* berupa *file audio* dengan data ucapan yang tidak bergantung pada teks atau *text-independent*. Pada penelitian ini, sistem pengenalan pembicara dibuat untuk dapat mengenali suara pembicara dengan menggunakan Mel-Frequency Cepstral Coefficients dan K-Means Clustering. Mel-Frequency Cepstral Coefficients digunakan untuk melakukan ekstraksi fitur dari data suara sehingga dihasilkan fitur-fitur yang mewakili pembicara tersebut. K-Means Clustering digunakan untuk melakukan proses *features matching* untuk mengenali pembicara berdasarkan fitur-fitur yang dihasilkan pada proses ekstraksi fitur. Berdasarkan hasil pengujian yang telah dilakukan, sistem pengenalan pembicara ini dapat mengenali pembicara dengan akurasi hingga 80%.

Kata Kunci: Mel-Frequency Cepstral Coefficients, K-Means Clustering, *Feature Matching, Text-Independent*.

ABSTRACT

Human voice can be used in various field of work or security field. Speaker recognition system can be implemented in security field, such as for password making process or to recognize the voice of perpetrators. This system based on the input of an audio file with speech data that doesn't rely on text or text-independent. In this paper, speaker recognition system was created to recognize the speaker's voice by using Mel-Frequency Cepstral Coefficients and K-Means Clustering. Mel-Frequency Cepstral Coefficients is used to perform feature extraction of voice data so that resulting that represent the speaker. K-Means Clustering is used to perform feature matching process to recognize the speaker based on features that produced in the process of feature extraction. Based on the empirical examination, this speaker recognition system can recognize the speaker with accuracy of 80%.

Keywords: Mel-Frequency Cepstral Coefficients, K-Means Clustering, Feature Matching, Text-Independent.

PEDOMAN PENGGUNAAN LAPORAN TUGAS AKHIR

Laporan Tugas Akhir yang tidak dipublikasikan terdaftar dan tersedia di perpustakaan Institut Teknologi Harapan Bangsa, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada pengarang dan pembimbing Tugas Akhir. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan seizin pengarang dan pembimbing Tugas Akhir disesuaikan dengan kebiasaan ilmiah untuk menyebutkan sumbernya.

Tidak diperkenankan untuk memperbanyak atau menerbitkan sebagian atau seluruh laporan tugas akhir tanpa seizin dari pengarang dan pembimbing tugas akhir yang bersangkutan.

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas berkat, karunia, dan rahmat-Nya, karena telah memberikan kekuatan dan hikmah-Nya dalam penyusunan dan perampungan Tugas Akhir ini sehingga dapat diselesaikan sebagai salah satu syarat untuk mencapai gelar sarjana strata satu pada program Teknik Informatika di Institut Teknologi Harapan Bangsa.

Laporan Tugas Akhir ini tidak akan terwujud tanpa dukungan dan bantuan dari berbagai pihak yang telah diberikan sejak awal hingga akhir. Oleh karena itu, pada kesempatan ini penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Tuhan Yang Maha Esa, atas segala penyertaan-Nya, kasih-Nya, dan mukjizat-Nya yang senantiasa nyata dalam hidup penulis. Terima kasih untuk segala sesuatu yang telah dipercayakan kepada penulis hingga saat ini, biar segala Puji dan Hormat hanya bagi nama-Mu, sekarang dan selamanya.
2. Ibu Elisafina Siswanto, S.T., M.T. selaku pembimbing 1 dalam Tugas Akhir ini. Terima kasih untuk waktu dan kesabarannya dalam membimbing penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini. Terima kasih juga untuk ilmu yang telah Ibu berikan baik selama kuliah maupun saat membimbing Tugas Akhir saya.
3. Bapak Oetomo Sudjana, S.T., M.T. selaku pembimbing 2 dalam Tugas Akhir ini. Terima kasih untuk waktu dan kesabarannya dalam membimbing penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini. Terima kasih juga untuk ilmu dan saran yang Bapak berikan untuk mendukung penulis dalam menyelesaikan Tugas Akhir ini.
4. Ibu Ir. Inge Martina, M.T. selaku Kepala Departemen Teknik Informatika ITHB dan penguji I dalam Tugas Akhir. Terima kasih atas dukungan, semangat, ilmu-ilmu, dan masukan yang telah diberikan kepada penulis dalam menyelesaikan Laporan Tugas Akhir ini.

5. Bapak Maclaurin Marulam Pandapotan Hutagalung, BE., M.Sc., Ph.D selaku penguji II dalam Tugas Akhir. Terima kasih atas dukungan, semangat, ilmu-ilmu, dan masukan yang telah diberikan kepada penulis dalam menyelesaikan Laporan Tugas Akhir ini.
6. Seluruh dosen dan staff Departemen Teknik Informatika ITHB yang telah membantu dalam menyelesaikan Laporan Tugas Akhir ini.
7. Segenap jajaran staf dan karyawan ITHB yang turut membantu kelancaran dalam menyelesaikan Laporan Tugas Akhir ini.
8. Kedua orang tua tercinta yang dalam kesibukannya selalu menyediakan waktu untuk memberikan doa dan dukungan yang tak habis-habisnya kepada penulis untuk menyelesaikan Laporan Tugas Akhir ini. Terima kasih untuk nasihat, masukan, perhatian dan kasih sayang yang diberikan hingga saat ini. Terima kasih juga untuk teguran yang diberikan sehingga penulis dapat terus belajar memperbaiki diri, terima kasih untuk teladan dan disiplin yang telah diberikan. Semoga ini semua dapat membuat kedua orang tua penulis bangga untuk segala usaha dan jerih lelahnya setiap waktu.
9. Teman-teman seperjuangan dari Teknik Informatika ITHB angkatan 2011, terima kasih untuk perhatian, doa, dan semangat yang selalu dilontarkan setiap saat. Terima kasih untuk waktu dan pengalaman berharga bersama kalian, empat tahun ini adalah masa yang berharga. Sukses untuk kalian semua.
10. Semua pihak yang telah membantu dan memberikan semangat dan doanya bagi penulis yang tidak dapat disebutkan satu per satu.

Laporan tugas akhir ini tidaklah sempurna. Oleh karena itu, kritik dan saran untuk membangun kesempurnaan tugas akhir ini sangat diharapkan. Semoga tugas akhir ini dapat membantu pihak-pihak yang membutuhkannya.

Bandung, Agustus 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

Pada bab ini akan menjelaskan mengenai latar belakang masalah, rumusan masalah, tujuan penelitian, kontribusi penelitian, batasan masalah, metodologi penulisan, dan sistematika penulisan.

1.1 Latar Belakang Masalah

Digital Signal Processing merupakan bidang yang berkembang dengan pesat. Saat ini sudah banyak penelitian yang berkaitan dengan menerima dan memproses suara manusia dalam bentuk *digital*. Dengan demikian suara manusia dapat dimanfaatkan dalam berbagai teknologi untuk mempermudah pekerjaan atau dapat juga dimanfaatkan dalam bidang keamanan.

Suara dihasilkan dari getaran pita suara ketika manusia berbicara atau mengeluarkan suara. Setiap orang memiliki karakter komposisi getaran suara yang berbeda-beda yang membuat mereka memiliki karakter suara yang berbeda-beda. Manusia dapat membedakan manusia lain hanya dengan mendengar suara mereka.

Oleh karena itu, penulis melakukan penelitian ini dengan tujuan untuk membuat sebuah sistem yang dapat mengidentifikasi pembicara berdasarkan input berupa *file audio* yang kemudian akan diproses untuk diidentifikasi pembicara dari *file audio* tersebut.

Dalam melakukan ekstraksi ciri, ada beberapa metode yang dapat dipakai, diantaranya MFCC, LPCC, LPC, GFCC. Berdasarkan penelitian yang dilakukan oleh Li Zhu dan Qing Yang yang menerapkan metode LPCC dan mendapatkan tingkat akurasi sekitar 88 % - 96 % menggunakan metode Vector Quantization (VQ) sebagai metode untuk mengklasifikasikan data suara [ZHU12]. Metode LPCC dapat mengekstrasi fitur dengan cukup baik pada file suara dengan frekuensi *sampling* rendah (dibawah 1000 Hz). Berdasarkan penelitian yang telah dilakukan oleh Utpal Bhattacharjee dengan melakukan percobaan melakukan ekstraksi ciri pada file suara, metode LPCC dapat menghasilkan akurasi sebesar 94,23%, namun metode ini masih kurang baik dalam menangani *noise* sehingga

akurasi yang dihasilkan akan menurun. Metode ekstraksi fitur MFCC dapat memberikan akurasi yang lebih baik terhadap pemrosesan suara yang mengandung *noise* [BHA13]. Algoritma MFCC yang digunakan dalam ekstrasi ciri suara menunjukkan akurasi yang cukup baik dalam mendeteksi pembicara [ALF12].

K-Means Clustering digunakan sebagai *machine learning* untuk mengelompokkan suatu objek yang akan diidentifikasi. Dengan menggunakan metode ini, ruang untuk penyimpanan tidak terlalu besar karena informasi suara yang diperoleh akan dikompres [ZHU12] dan perhitungannya cepat walaupun dengan jumlah variabel yang banyak [SAR14]. K-Means Clustering juga memiliki kelebihan yaitu karena sangat sederhana dan kuat, sangat efisien, dan dapat digunakan untuk berbagai jenis data [WUJ12]. Sedangkan Mel-Frequency Cepstral Coefficients (MFCC) merupakan metode ekstraksi fitur yang dapat memodelkan pendengaran manusia. Metode MFCC ini digunakan karena dapat diimplementasikan dengan baik pada *file* suara dengan frekuensi *sampling* tinggi (diatas 1000 Hz) atau kualitas audio yang sangat baik. MFCC juga cukup baik dalam mengkarakterisasi suara manusia sehingga dapat memberikan hasil yang baik untuk mengidentifikasi pembicara [ALF12].

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka peneliti merumuskan masalah bagaimana sistem dapat melakukan identifikasi pembicara dengan tepat sebagai berikut:

1. Bagaimana membuat sebuah sistem yang dapat mendeteksi pembicara dengan menerapkan metode Mel-Frequency Cepstral Coefficients untuk ekstraksi fitur?
2. Bagaimana menerapkan *machine learning* K-Means Clustering?
3. Bagaimana akurasi yang dihasilkan oleh sistem dengan menerapkan metode ekstraksi fitur dan *machine learning* K-Means Clustering?

1.3 Tujuan Penelitian

Tujuan dilakukannya penelitian ini adalah untuk membuat sebuah sistem yang dapat mengidentifikasi pembicara melalui *file audio* yang diinput. Sistem pendeteksi pembicara ini dapat dimanfaatkan dalam berbagai bidang seperti keamanan untuk mengenali suara pelaku kejahatan atau dapat dimanfaatkan dalam pembuatan *password* untuk meningkatkan keamanan.

1.4 Kontribusi Penelitian

Penelitian ini mengembangkan penelitian yang dilakukan oleh Erwin Hadinata dengan judul "*Speech Recognition Menggunakan Metoda Fast Fourier Transform*" yang melakukan penelitian tentang pendeteksian kata dengan menggunakan Fast Fourier Transform. Pada penelitian ini penulis akan membuat suatu sistem yang dapat mengenali suara pembicara dengan metode ekstraksi ciri yang berbasis Fast Fourier Transform, yaitu Mel-Frequency Cepstral Coefficients dan juga menerapkan *machine learning* K-Means Clustering untuk mengenali pembicara.

1.5 Batasan Masalah

Dari rumusan masalah di atas, penulis membatasi permasalahan yang akan dibahas yang mencakup:

1. Pendeteksian pembicara dilakukan dengan memproses *file audio* dengan format WAV.
2. Hanya menangani *file audio* yang memiliki satu *channel* (*mono*).
3. Data *training* dan *testing* direkam dengan alat perekam yang sama.

1.6 Metodologi Penelitian

Untuk mendapatkan data yang dibuthkan dalam penelitian, maka penulis mengumpulkan sumber data dengan metode berikut:

1. Studi Pustaka

   Penulis mengumpulkan data yang digunakan sebagai bahan untuk menunjang penelitian melalui artikel atau bahan bacaan lainnya yang berhubungan dengan pokok pemasalahan.

2. Analisis dan perancangan

   Menganalisis contoh kasus dengan menerapkan metode-metode yang ada.

3. Implementasi sistem

   Sistem diimplementasikan dalam bentuk perangkat lunak yang menggunakan bahasa pemrograman Java.

4. Pengujian dan analisis sistem

   Pada tahap ini akan dilakukan analisis apakah sistem pengidentifikasi pembicara ini dapat berjalan sesuai tujuan atau tidak, dan jika terdapat kesalahan pada sistem akan dilakukan perbaikan.

5. Dokumentasi sistem

   Pembuatan laporan Tugas Akhir lengkap dengan analisis yang didapatkan.

1.7 Sistematika Penelitian

Laporan Tugas Akhir ini disusun dengan sistematika penulisan seperti yang dideskripsikan di bawah ini :

1. BAB I PENDAHULUAN

   Bab ini menjelaskan tentang latar belakang pemilihan topik, rumusan masalah, batasan masalah, tujuan penelitian, metodologi penelitian, dan sistematika penulisan.

2. BAB II LANDASAN TEORI

   Bab ini menjelaskan tentang landasan teori yang digunakan sebagai acuan dalam penulisan skripsi, yaitu mencakup tahapan-tahapan dalam perancangan aplikasi dan pengolahan *file audio*.

3. BAB III ANALISIS DAN PERANCANGAN

   Pada bab ini diuraikan mengenai analisis sistem yang akan dikembangkan. Kemudian dari hasil analisa tersebut akan diketahui masalah yang dihadapi pada sistem yang sedang berjalan.

4. BAB IV IMPLEMENTASI DAN PENGUJIAN

   Pada bab ini dijelaskan mengenai penelitian dan pengujian yang dilakukan seperti pengujian *silence removal*, *frame blocking*, *threshold*, *data training* dan *testing* yang digunakan.

5. BAB V KESIMPULAN DAN SARAN

   Bab ini berisi kesimpulan mengenai hasil dari penelitian yang telah dilakukan dan saran yang dibutuhkan untuk pengembangan penelitian lebih lanjut.

BAB II
LANDASAN TEORI

Pada bab ini akan dibahas mengenai teori-teori yang digunakan sebagai pendukung penelitian yang penulis lakukan. Teori-teori yang dibahas adalah mengenai suara dan algoritma yang digunakan dalam sistem untuk menentukan jenis suara manusia.

2.1 Sistem Pendengaran Manusia

Bagian mekanik pendengaran terdiri dari tiga bagian, *external ear*, *middle ear*, dan *inner ear*. *External ear* merupakan kombinasi kartilago di auricula dan meatus akustik eksternal (saluran telinga). *Middle ear* meliputi membran timpani (gendang telinga) dan segala sesuatu di sisi kanannya. Gerakan membran timpani yang diseabkan oleh gelombang suara melalui dan diperkuat oleh *external ear*. Getaran dikirimkan oleh membran timpani ke maleus, dari maleus ke inkus, lalu ke tulang sanggurdi. Lalu getaran dikirimkan ke telinga bagian dalam melalui ovalis fenestra koklea. Gerakan tulang sanggurdi menyebabkan gelombang tekanan pada cairan telinga bagian dalam yang kemudian mengeksitasi ribuan rambut (silia) di dalam spiral koklea (tympani scala). Silia tersambung ke saraf pendengaran dan mengirimkan sinyal gerakan ke otak untuk kognisi [BEI11].

Bentuk spiral timpani scala memberikan kemampuan kognitif semi-logaritmik suara yang penting dalam pengembangan model *speaker* dan fitur. Titik akhir itu disebut helicotrema. Mulai di ovalis fenestra koklea, *audio high-pitch* direalisasikan. Sebagai suara perjalanan menuju helicotrema, komponen bernada tinggi ditekan dan hanya komponen bernada rendah bertahan seperti itu, dekat dengan helicotrema, hanya nada terendah yang direalisasikan [BEI11].

2.2 File Audio WAV

Format file audio WAV dikembangkan bersama-sama oleh Microsoft dan IBM. WAV memiliki *header* yang dimulai dengan RIFF (*Resource Interchange File Format*) dan dilanjutkan dengan informasi ukuran audio mentah, *codec* yang

digunakan dan juga beberapa informasi lain yang ditambahkan oleh pemiliknya [BEI11].

2.3 Discrete Fourier Transform

Discrete Fourier Transform (DFT) adalah metode yang umum digunakan di bidang pemrosesan sinyal digital. DFT memungkinkan kita untuk menganalisa, memanipulasi, dan mensintesis sinyal analog. DFT adalah prosedur matematis yang digunakan untuk menentukan konten harmonik atau konten frekuensi dari urutan sinyal diskrit. Urutan sinyal diskrit merupakan suatu set nilai yang diperoleh melalui sampel periodik dari sinyal kontinyu dalam domain waktu [LYO11]. DFT diefinisikan melalui persamaan di bawah ini [LYO11]:

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}$$

(2.1)

$x(n)$ merupakan urutan nilai diskrit sampel domain waktu dari suatu sinyal. e adalah basis logaritma natural, dan $j = \sqrt{-1}$ [LYO11].

Euler's relationship, $e^{-j\,\omega} = \cos(\omega) - j\sin(\omega)$, setara dengan [LYO11]:

$$X(m) = \sum_{n=0}^{N-1} x(n)[\cos\left(\frac{2\pi nm}{N}\right) - jsin\left(\frac{2\pi nm}{N}\right)]$$

(2.2)

Yang berarti $\omega = 2\pi nm/N$.

Keterangan:

$X(m)$ = komponen *output* DFT ke-m, seperti, $X(0)$, $X(1)$, $X(2)$, dst

m = indeks *output* DFT dalam domain frekuensi, m = 0, 1, 2, 3, ... , N-1

$x(n)$ = urutan *input* sampel, $x(0)$, $x(1)$, $x(2)$, $x(3)$, dst

n = indeks *input* sampel dalam domain waktu, n = 0, 1, 2, 3 ... , N-1

$j = \sqrt{-1}$, dan

N = jumlah urutan input sampel dan jumlah titik frekuensi dalam *output* DFT

Dengan input N nilai sampel domain waktu, DFT menentukan konten spektral dari input pada N titik frekuensi. Nilai N merupakan parameter penting karena menentukan input sampel yang dibutuhkan, resolusi hasil frekuensi domain, dan jumlah pemrosesan waktu yang dibutuhkan untuk menghitung N-titik DFT. Nilai frekuensi dari tiap sinusoid yang berbeda tergantung pada frekuensi sampling dan jumlah sampel N. Frekuensi analisis didefinisikan melalui persamaan berikut [LYO11]:

$$f_{analysis}(m) = \frac{mf_s}{N} \tag{2.3}$$

$X(m)$ dari persamaan di atas menghasilkan nilai *magnitude* yang merupakan hasil dari akar dari penjumlahan bilangan real dan bilangan imajiner yang dikuadratkan. Persamaan berikut menunjukkan cara untuk mendapatkan nilai *magnitude* [LYO11]:

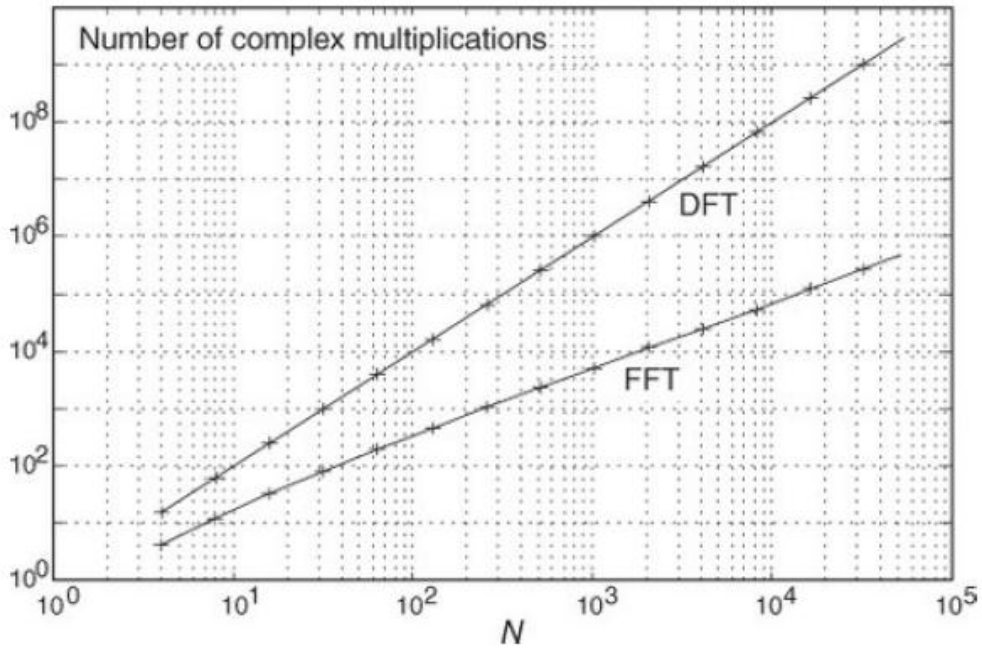
$$X_{mag}(m) = |X(m)| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2} \tag{2.4}$$

Sedangkan *power spectrum* didefinisikan sebagai berikut [LYO11]:

$$X_{PS}(m) = X_{mag}(m)^2 = X_{real}(m)^2 + X_{imag}(m)^2 \tag{2.5}$$

2.4 Fast Fourier Transform

Operasi Discrete Fourier Transform membutuhkan operasi O(N^2). Fast Fourier Transform mengurangi operasi perhitungannya menjadi O(N $\log_2(N)$) [BEI11]. Dengan menggunakan FFT, jumlah operasi aritmatika yang diperlukan menjadi berkurang. Misalnya untuk 8-point DFT kita memerlukan N^2 atau 64 kali perkalian kompleks. Sedangkan untuk FFT hanya memerlukan kira-kira (N/2) $\log_2$ N. Perbandingannya dapat dilihat pada gambar di bawah ini [LYO11].



Gambar 2.1 Perbandingan operasi matematika yang diperlukan FFT dan DFT

Salah satu algoritma FFT yang populer adalah algoritma FFT radix-2 yang membutuhkan input sampel sebanyak $N = 2^k$. Dengan demikian total interval suatu kumpulan data sampel adalah N/f_s detik. Dan resolusi frekuensinya adalah f_s/N Hz. N tersebut adalah jumlah seluruh sampel dari suatu sinyal atau *file audio*. Karena tidak semua sinyal yang telah di sampling memiliki sampel sebanyak 2^k, maka sinyal tersebut dapat ditambahkan sampel bernilai nol sampai jumlah sampel = 2^k [LYO11] . Untuk menentukan nilai *magnitude* dari dengan FFT dapat digunakan persamaan 2.4:

$$X_{mag}(m) = |X(m)| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2}$$

(2.6)

FFT memisahkan urutan data input *x*(*n*) menjadi dua bagian, elemen indeks ganjil dan elemen indeks genap, didefinisikan melalui persamaan berikut:

$$X(m) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1)W_{N/2}^{nm}$$

(2.7)

untuk *m* = 0 sampai *m* = *N*/2-1, dan

$$X\left(m + \frac{N}{2}\right) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n)W_{N/2}^{nm} - W_N^m \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1)W_{N/2}^{nm}$$

(2.8)

untuk *m* = *N*/2 sampai *m* = *N*, dimana:

$$W_{N/2} = e^{-j2\pi/(\frac{N}{2})}$$

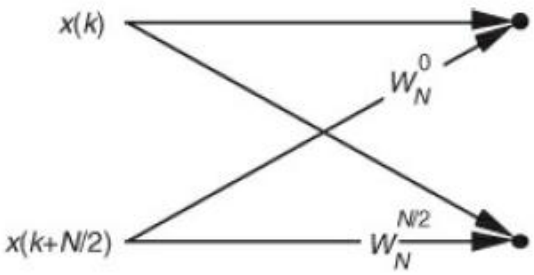
(2.9)

$$W_N = e^{-j2\pi/N}$$

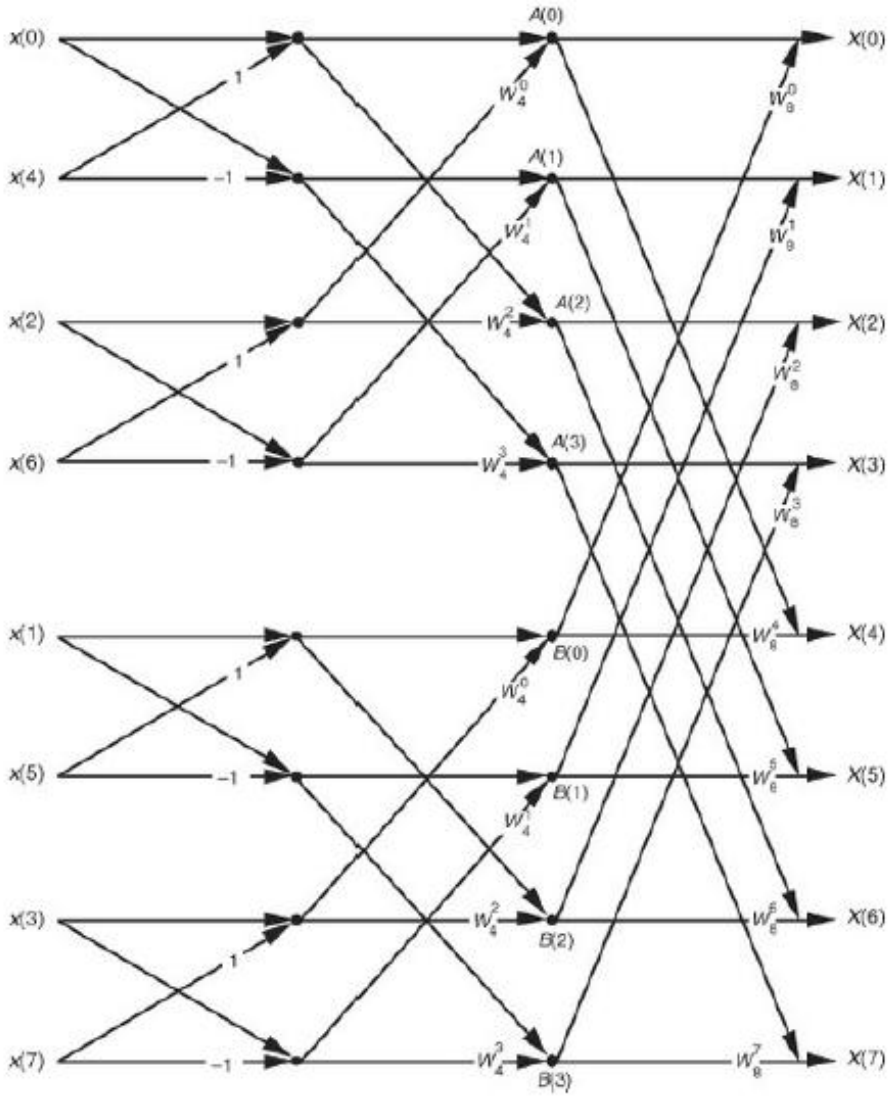
(2.10)

m dalam rentang 0 sampai *N*/2-1 karena masing-masing dari dua *N*/2-point DFT di sisi kanan pada persamaan 2.7 periodik dalam *m* dengan periode *N*/2 [LYO11].

Tahap dalam FFT ditentukan berdasarkan jumlah *input* sampel *N* dengan $Log_2(N)$. Pada tahap pertama Fast Fourier Transform membagi *N*-point sampel menjadi *N*/2-point sampel dan seterusnya sampai *N* = 2-point atau *single* 2-point sampel [LYO11].



Gambar 2.2 *Single* 2-*point* DFT



Gambar 2.3 Implementasi FFT *Full Decimation-In-time* 8-titik DFT

2.5 *Frame Blocking*

Pada tahap *frame blocking* ini sinyal audio dibagi menjadi beberapa *frame* dengan mengambil jumlah sampel sebanyak *N* untuk setiap *frame*. Ini dilakukan karena sinyal *audio* mengalami perubahan dalam jangka waktu tertentu. Biasanya *frame blocking* dilakukan dengan lebar 20-30 ms [BEI11].

2.6 Hamming *Window*

Proses *frame blocking* dapat menyebabkan kebocoran spektral karena diskoninuitas dari sinyal yang disebabkan pada proses *frame blocking*. Sehingga

untuk meminimalisir diskontinuitas yang terjadi pada sinyal, maka diperlukan proses *windowing*. Konsep dari *windowing* adalah melancipkan ujung sinyal menjadi nol pada bagian awal dan akhir *frame* dengan mengalikan tiap *frame* dari sinyal dengan *window* seperti berikut:

$$y_1(n) = x_1(n)w(n), \qquad 0 \leq n \leq N-1$$

(2.11)

Keterangan:

$y_1(n)$ = Nilai sampel yang telah di *windowing*

$x_1(n)$ = Nilai sampel awal sebelum di *windowing*

$w(n)$ = Nilai Hamming *window*

n = Indeks sampel dalam suatu *frame*

N = Jumlah sampel tiap *frame*

dimana N adalah total sampel sinyal *file audio*, $y_l(n)$ adalah sinyal yang sudah dikalikan dengan *window*, $x_l(n)$ adalah sinyal *frame* ke-*n*, dan $w(n)$ adalah fungsi *window*. *Window* yang digunakan adalah window Hamming dengan rumus sebagai berikut [LYO11]:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right), \qquad 0 \leq n \leq N-1$$

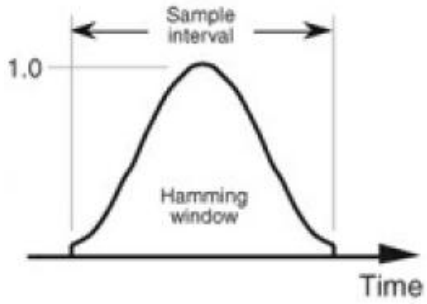
(2.12)

Keterangan:

$w(n)$ = Nilai Hamming *window*

n = Indeks sampel dalam suatu *frame*

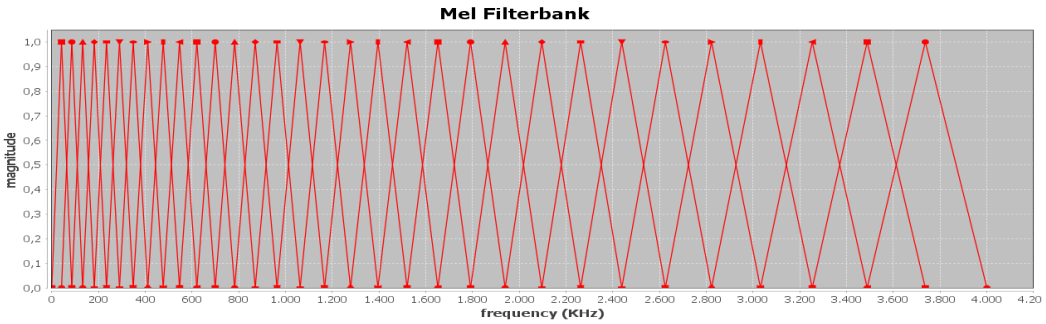
N = Jumlah sampel tiap *frame*



Gambar 2.4 Hamming Window

2.7 Mel Frequency Wrapping

Pada proses Mel-Frequency Wrapping ini sinyal suara yang telah diubah ke dalam bentuk *frequency-domain* akan di saring dengan sebuah *mel filterbank*. Mel filterbank akan menyaring sinyal sebanyak *N* sampel. *Mel filterbank* terdiri dari rangkaian *Triangular Window* yang saling *overlapping* [ALF12].



Gambar 2.5 *Mel Filterbank*

$$B(f) = 1125 \ln(1 + \frac{f}{700})$$

(2.13)

Keterangan:

$B(f)$ = Nilai mel, hasil konversi dari nilai frekuensi

f = Nilai frekuensi.

$$B^{-1}(b) = 700(\exp\left(\frac{b}{1125}\right) - 1)$$

(2.14)

Keterangan:

$B^{-1}(b)$ = Nilai frekuensi, hasil konversi nilai dari nilai mel

b = Nilai mel

Untuk mebuat filterbank seperti gambar 2.6, batas bawah f_l dan batas atas f_h frekuensi harus ditentukan terlebih dahulu. Batas atas frekuensi maksimal adalah setengah dari frekuensi sampling dai *file audio*. Setelah itu menentukan nilai mel menggunakan persamaan 2.13, nilai mel didefinisikan sebagai *mel*(i) [HUA01].

Setelah itu menentukan berapa banyak *M filter* yang ingin dibuat. Pada gambar 2.6 filter berjumlah 32 buah. *Filter* ini adalah *triangular filter* yang saling *overlapping*. Setiap *filter* berjarak linear sehingga untuk menentukan titik-titik untuk membuat *filter* digunakan persamaan berikut ini [HUA01]:

Nilai mel yang didapat dirubah ke dalam Hertz (Hz), $B^{-1}(b)$, dengan persamaan 2.14. Namun untuk dapat membuat *filterbank* nilai frekuensi $B^{-1}(b)$ harus dikonversikan ke dalam nilai FFT bin terdekat dengan menggunakan persamaan 2.15 di bawah ini [HUA01]:

$$f[m] = \left(\frac{N}{F_s}\right) B^{-1}\left(B(f_l) + m\frac{B(f_h) - B(f_l)}{M+1}\right)$$

(2.15)

Keterangan:

$f[m]$ = Nilai FFT bin

N = Jumlah FFT tiap *frame*

F_s = Frekuensi sampling

f_h = Frekuensi batas atas

f_l = Frekuensi batas bawah

M = Jumlah *filter*

Sedangkan filterbank didefinisikan melalui persamaan berikut ini [HUA01]:

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \dfrac{k - f[m-1]}{f[m] - f[m-1]} & f[m-1] \le k \le f[m] \\ \dfrac{f[m+1] - k}{f[m+1] - f[m]} & f[m] \le k \le f[m+1] \\ 0 & k > f[m+1] \end{cases}$$

(2.16)

Keterangan:

$H_m[k]$ = Nilai *filterbank*

m = Indeks *filter*

k = Indeks input sampel FFT (k = 0, 1, ... , N - 1)

$f[m]$ = Nilai FFT bin ke-m

Proses pemfilteran sinyal dilakukan dengan persamaan 2.17 sehingga dihasilkan *log energy* pada tiap *filter* [HUA01].

$$S[m-1] = \ln\left[\sum_{k=0}^{N-1} |X_a[k]|^2 H_m[k]\right], \qquad 1 \le m \le M$$

(2.17)

Keterangan:

$S[m - 1]$ = Nilai *log energy*

$X_a[k]$ = Nilai *magnitude* (Hz)

$H_m[k]$ = Nilai *filterbank*

N = Jumlah nilai FFT tiap *frame*

m = Indeks *filter*

k = Indeks *input* sampel FFT (k = 0, 1, ... , N - 1)

2.8 Cepstrum

Koefisien amplitudo dari spektrum akan dihasilkan dengan menggunakan Discrete Cosine Transform (DCT). C_0 merupakan nilai rata-rata dalam dB yang dapat digunakan untuk estimasi energi yang berasal dari *filterbank*. Koefisien DCT adalah nilai amplitudo dari spektrum yang dihasilkan [ALF12]. Biasanya 13 koefisien cepstrum pertama saja yang digunakan. Persamaan DCT didefinisikan sebagai berikut [HUA01]:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\frac{\pi n\left(m+\frac{1}{2}\right)}{M}\right) \qquad 0 \leq n < M$$

(2.18)

Keterangan:

$c[n]$ = Nilai cepstrum ke-n

$S[m]$ = Nilai *log energy*

M = Jumlah *filter*

m = Indeks *filter*

n = Indeks koefisien DCT

2.9 K-Means Clustering

K-Means Clustering merupakan metode yang digunakan untuk analisis *cluster* pada *data mining* [SAR14]. Algoritma ini memungkinkan untuk memproses sampel dalam jumlah yang sangat besar pada komputer *digital* karena sangat sederhana dan kuat, sangat efisien, dan dapat digunakan untuk berbagai jenis data [WUJ12].

K-Means Clustering merupakan algoritma *partitional clustering* sederhana yang berusaha untuk menemukan K *non-overlapping cluster*. *Cluster* tersebut memiliki *centroid* yang merupakan objek yang menjadi pusat *cluster* tersebut. Berikut proses K-Means Clustering [WUJ12]:

1. *User* menentukan *K centroid* awal, dimana *K* merupakan jumlah *cluster*.

2. Mengelompokkan setiap objek kepada *cluster* dengan *centroid* terdekat. Jarak dari *cluster* dan objek diukur dengan menggunakan Euclidean Distance [BAR05].

$$d = \sqrt{\sum_{i=1}^{P}(V_{1i} + V_{2i})^2}$$

(2.19)

Keterangan:

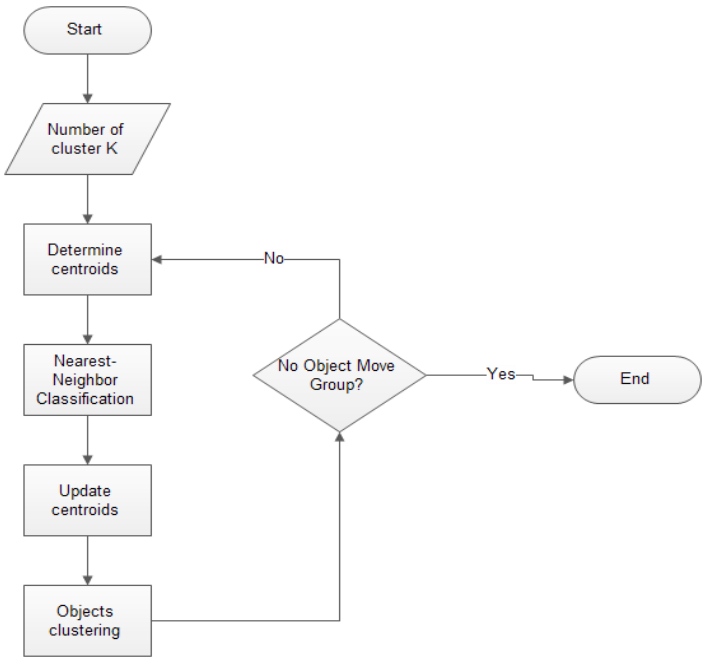
d = jarak

P = dimensi

V_{1i} = nilai ke-i objek 1

V_{2i} = nilai ke-i objek 2

3. Menentukan kembali *centroid* setiap *cluster* berdasarkan objek-objek yang ditetapkan terhadap *centroid* dari *cluster* tersebut.
4. Ulangi langkah 2 dan 3 sampai tidak ada objek yang berpindah *cluster*.



Gambar 2.6 Alur Kerja K-Means Clustering

2.10 Common Math

Commons Math adalah *library* yang berisi komponen-komponen matematika dan statistika. Library ini digunakan untuk mengatasi masalah-masalah perhitungan umum yang tersedia dalam bahasa pemrograman Java. Dengan menggunakan *library* ini, kita dapat menghitung nilai Fast Fourier Transform dengan memanfaatkan kelas FastFourierTransformer.

Kelas FastFourierTransformer merupakan kelas yang berfungsi untuk mengimplementasikan Fast Fourier Transform untuk transformasi dataset riil dan kompleks 1 dimensi. Ada beberapa varian dari Discrete Fourier Transform yang ditentukan oleh parameter DftNormalization dengan berbagai konvensi normalisasi. Panjang *input* implementasi dari Fast Fourier Transform ini membutuhkan panjang data sebanyak bilangan *power of two*.

Tabel 2.1 *Constructor* dan *Method* Kelas FastFourierTransformer

FastFourierTransformer *Class Constructors*		
Constructor	**Deskripsi**	
FastFourierTransformer(DftNormalization normalization)	Untuk menciptakan turunan kelas objek baru dengan berbagai variasi konvensi nomalisasi.	
FastFourierTransformer *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
Complex[]	transform(double[] f, TransformType type)	Menghitung nilai kompleks dengan (forward, inverse) transform dari dataset bilangan riil.

Tabel 2.2 Konstanta *Enum* Kelas DftNormaliztion

DftNormalization *Class Enum Constants*	
Enum Constant	**Deskripsi**
STANDARD	Konvensi normalisasi yang digunakan untuk normalisasi standard. Kovensi normalisasi didefinisikan sebagai berikut: • Forward transform: $Y_n = \sum_{k=0}^{N-1} X_k \exp(-\frac{2\pi ink}{N})$ • Inverse transform: $X_k = N^{-1} \sum_{n=0}^{N-1} Y_n \exp\left(\frac{2\pi ink}{N}\right)$ dimana N adalah jumlah data sampel.

Tabel 2.3 *Method* Kelas Complex

Complex *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
double	getImaginary()	Mengakses nilai imajiner dari bilangan kompleks.
double	getReal()	Mengakses nilai riil dari bilangan kompleks.

2.11 JFreeChart

JFreeChart merupakan *Java library* yang dapat menggambarkan berbagai macam bentuk grafik, seperti diagram, histogram, pie chart, grafik koordinat cartesius, dll.

Tabel 2.4 *Constructor* dan *Method* Kelas JFreeChart

JFreeChart *Class Constructors*		
Constructor	**Deskripsi**	
JFreeChart(Plot plot)	Membuat grafik berdasarkan *plot* yang diberikan.	
JFreeChart Class Methods		
Modifier* dan *type	***Method***	**Deskripsi**
XYPlot	getXYPlot()	Untuk mengatur jarak axis.

Tabel 2.5 *Constructor* dan *Method* Kelas XYSeries

XYSeries Class *Constructors*		
Constructor	**Deskripsi**	
XYSeries(java.lang.Comparable key)	Untuk membuat kelas objek XYSeries, seri baru yang masih kosong. Nilai yang ditambahkan ke dalam seri akan secara otomatis diurutkan dari nilai terkecil ke nilai yang terbesar.	
XYSeries Class *Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
void	add(double x, double y)	Menambahkan nilai x dan y ke dalam seri.

Tabel 2.6 *Constuctor* dan *Method* Kelas XYSeriesCollection

XYSeriesCollection *Class Constructors*		
Constructor	**Deskripsi**	
XYSeriesCollection()	Membuat dataset kosong.	
XYSeriesCollection *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
void	addSeries(XYSeries series)	Menambahkan seri ke dalam *collection*.

Tabel 2.7 Constructor dan Method Kelas *XYPlotConstructor* dan Method Kelas XYPlot

XYPlot *Class Constructors*		
Constructor	**Deskripsi**	
XYPlot(XYDataset dataset, ValueAxis domainAxis, ValueAxis rangeAxis, XYItemRenderer renderer)	Membuat dataset kosong.	
XYPlot *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
ValueAxis	getDomainAxis()	Untuk mendapatkan domain axis.

Tabel 2.8 *Constructor* dan *Method* Kelas ValueAxis

ValueAxis *Class Constructors*		
Constructor	**Deskripsi**	
XYPlot(XYDataset dataset, ValueAxis domainAxis, ValueAxis rangeAxis, XYItemRenderer renderer)	Membuat dataset kosong.	
ValueAxis *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
void	setRange(double lower, double upper)	Untuk mengatur jarak axis.

Tabel 2.9 *Method* Kelas ChartFactory

ChartFactory *Class Methods*		
Modifier* dan *type	***Method***	**Deskripsi**
static JFreeChart	createXYLineChart(java.lang.String title, java.lang.String xAxisLabel, java.lang.String yAxisLabel, XYDataset dataset)	Untuk membuat grafik berupa garis berdasarkan XYDataset dengan pengaturan bawaan.

Tabel 2.10 *Constructor* Kelas ChartPanel

ChartPanel *Class Constructors*	
Constructor	**Deskripsi**
ChartPanel(JFreeChart chart)	Membuat panel yang menampilkan suatu grafik.

2.12 Tinjauan Studi

Pada penelitian ini penulis melakukan sebuah studi terhadap penelitian yang dilakukan oleh Alfredo Maesa, Fabio Garzia, Michele Scarpiniti, dan Roberto Cusani pada penelitian tentang "*Text Independtdent Automatic Speaker Recognition System Using Mel-Frequency Cepstrum Coefficient and Gaussian Mixture Models*". Pada penelitian yang dilakukan tersebut berupaya untuk membuat suatu sistem yang dapat mengidentifikasi siapa pembicara dari suatu *file audio*. Sistem tersebut menggunakan MFCC (Mel Frequency Cepstral Coefficients) untuk ekstraksi fitur dan GMM (Gaussian Mixture Model).

Pertama *file audio* yang diinput akan di proses terlebih dahulu pada tahap *pre-processing*. Pada proses *pre-processing noise* dari data suara di hilangkan dengan algoritma *spectral substraction*. Kemudian *file* tersebut akan dicari frekuensi fundamentalnya lalu diproses dengan Mel Frequency Cepstral Coefficients. Parameter yang digunakan adalah *frame length* kurang dari 20 ms dengan 50 % *overlapping*. Sementara koefisien Discrete Cosine Transform yang digunakan adalah 20 koefisien Discrete Cosine Transform.

File suara yang digunakan adalah *database* suara yang diperoleh dari *Voxforge.org*. Pada penelitian tersebut digunakan 450 jenis ucapan dari tiap pembicara. Tiap pembicara digunakan dua jenis ucapan untuk fase *training* dan *testing*.

Dari hasil penelitian yang dilakukan, hasilnya Mel Frequency Cepstral Coefficients menunjukkan akurasi yang cukup baik. Mel Frequency Cepstral Coefficients dan Gaussian Mixture Model dapat meraih akurasi hingga 97,98 % dalam sistem identifikasi pembicara tersebut.

Kemudian dalam penelitian yang dilakukan oleh Vicky Zilvan dan Furqon Hensan Muttaqien dengan judul "*Identifikasi Pembicara Menggunakan Algoritme VFI5 dengan MFCC sebagai Pengektraksi Ciri*", Mel Frequency Cepstral Coefficients digunakan sebagai pengekstraksi fitur dan untuk mengklasifikasikan data, digunakan algoritma Voting Feature Intervals 5.

Pada proses Mel Frequency Cepstral Coefficients ada beberapa tahap yang dilakukan untuk mendapatkan fitur dari data suara, seperti *frame blocking*,

windowing, Fast Fourier Transform, Mel Frequency Wrapping, dan Discrete Cosine Transform. Parameter yang digunakan dalam tahap ekstraksi fitur ini antara lain, *sampling rate* 16000 Hz, *frame length* 30 ms, *overlapping* 0,5, dan 13 koefisien DCT. Data suara yang digunakan terdiri dari 10 pembicara yang berbeda, 5 laki-laki dan 5 perempuan. Data suara tersebut bersifat *text dependent*, karena itu semua pebicara mengucapkan kata yang sama. Kata yang diucapkan adalah "komputer".

Pertama, data suara yang telah direkam diproses dengan Mel Frequency Cepstral Coefficients untuk didapatkan fiturnya, kemudian dilakukan *training* dengan algoritma VFI5.

Mel Frequency Cepstral Coefficients menunjukkan akurasi yang baik dengan akurasi tertinggi mencapai 97%. Dengan menggunakan 11 data latih, akurasi yang didapat sudah cukup baik yaitu sebesar 94,5% sedangkan dengan 38 data latih akurasi yang didapat adalah 97%.

BAB III
ANALISIS DAN PERANCANGAN

Pada bab ini akan dibahas apa saja yang diperlukan untuk membangun sistem yang dapat mengidentifikasi pembicara dengan mendeteksi suatu *file audio*. Adapun dalam membangun sistem ini, diperlukan suatu metode untuk melakukan ekstraksi fitur yang dimana fitur-fitur tersebut membawa karakter yang khusus dari pembicara tersebut. Dan untuk dapat mengenali pembicara, diperlukan *machine learning* yang akan melakukan identifikasi terhadap fitur-fitur yang telah diekstrak.

3.1 Analisis Masalah

Speaker recognition merupakan sebuah sistem yang dapat mengidentifikasi pembicara melalui input berupa *file audio*. Input yang digunakan untuk sistem ini adalah *file audio* WAV. Agar sistem dapat mengidentifikasi pembicara, sistem perlu dibuat dengan metode yang dapat memodelkan pendengaran manusia untuk diterapkan pada sistem.

Dari *file audio* tersebut, sistem akan melakukan ekstraksi fitur untuk mendapatkan *acoustic vector*. *Acoustic vector* ini merupakan parameter yang akan digunakan untuk mengidentifikasi pembicara. Dalam tahap ini diperlukan metode yang dapat memodelkan pendengaran manusia sehingga sistem dapat memodelkan suatu nilai yang mendekati pendengaran manusia.

Untuk itu, *machine learning* K-Means Clustering diterapkan pada sistem ini untuk mengklasifikasi dan mengidentifikasi pembicara. Dengan memanfaatkan nilai-nilai *acoustic vector* sistem akan melakukan perbandingan untuk mengenali pembicara tersebut.

3.2 Analisis Dataset

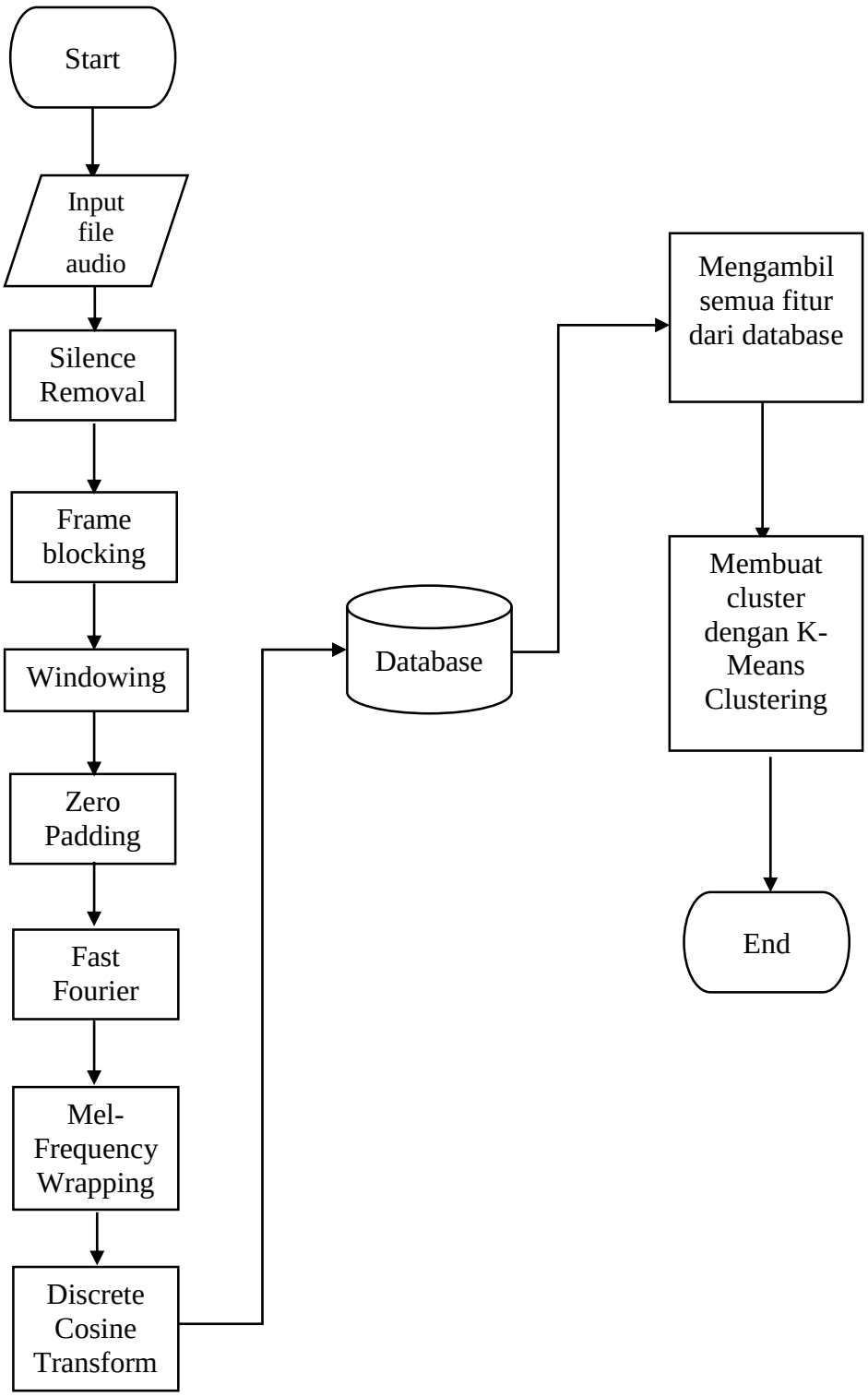
Data yang akan digunakan untuk sistem ini adalah data suara berupa *file audio* WAV karena *file audio* WAV merupakan *file raw* dan belum terkompres. Setiap *file audio* berisi suara dari satu orang pembicara dalam durasi 3 detik. Pada

penelitian ini, penulis menggunakan *file audio* dari 5 pembicara. Masing-masing pembicara direkam sebanyak 18 kali. Dari 15 *file audio* tersebut akan digunakan sebanyak 15 *file audio* digunakan untuk data *training* dan 3 *file* suara untuk data *testing*. Setiap pembicara akan mengucapkan angka 0 sampai 9 secara acak. Perekaman dilakukan di dalam suatu ruangan tertutup namun masih terdapat *noise*. *File audio* yang digunakan memiliki frekuensi sampling 8000 HZ. Frekuensi sampling dipilih karena suara manusia normal adalah antara 500 Hz – 2000 Hz. Dan juga pendengaran manusia yang lebih sensitif pada rentang frekuensi 2000 Hz – 4000 Hz [MAR01]. Agar memenuhi *Nyquist Criterion*, maka frekuensi sampling yang dipakai adalah 8000 Hz [LYO11].

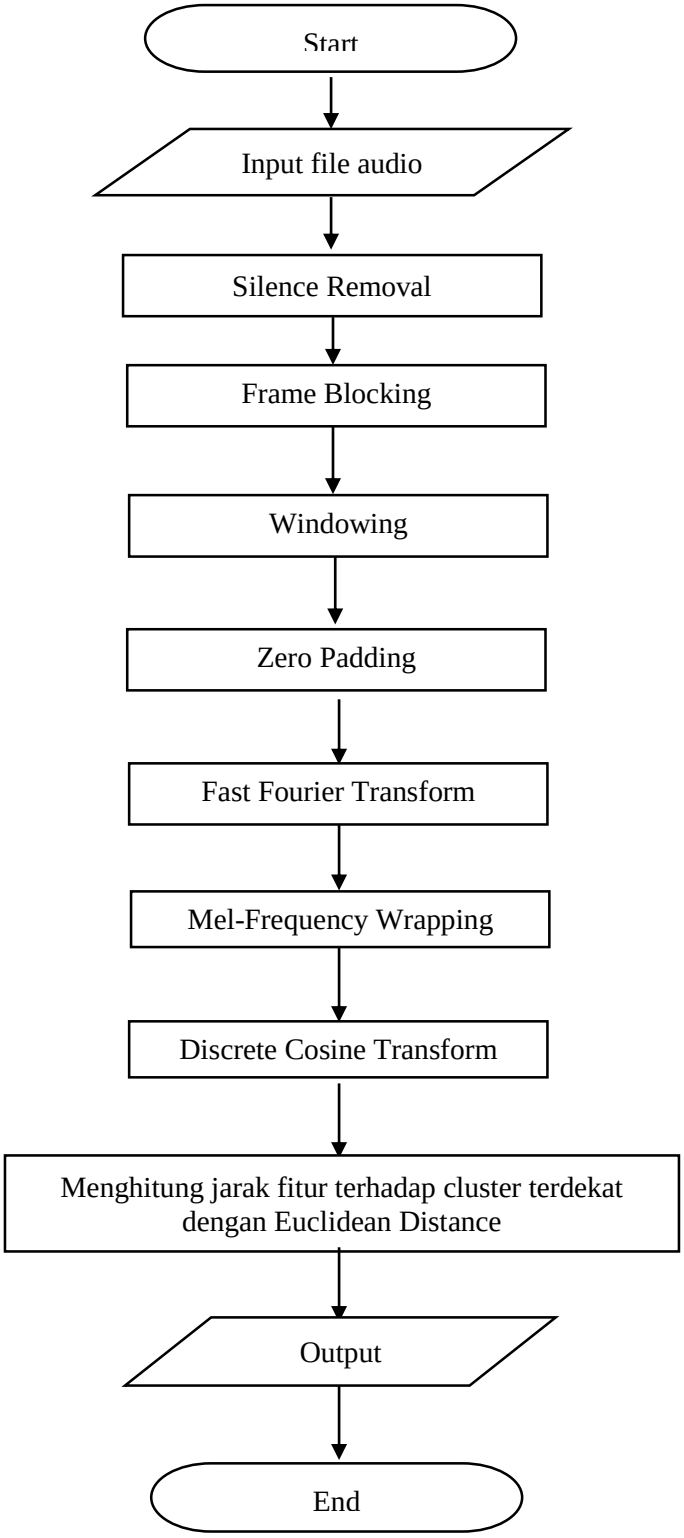
3.3 Perancangan Alur Kerja Sistem

Gambar 3.1 merupakan gambaran secara umum dari alur kerja sistem yang akan dibuat. Pertama, *user* harus menginput *file audio* yang akan diproses. Setelah itu baru kemudian dilakukan ekstraksi fitur terhadap *file audio*. Fitur-fitur yang telah diekstrak akan dipakai sebagai parameter untuk *machine learning* dalam proses pemodelan pembicara pada tahap *training*.

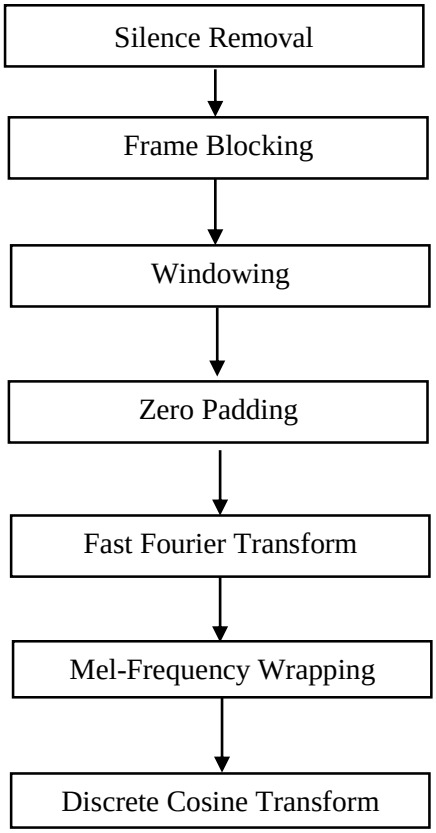
Setelah melakukan *training* terhadap data, baru dilakukan *testing* untuk mengidentifikasi pembicara dengan membandingkan fitur *data training* dan *data testing*. Sistem akan menampilkan output berupa ID pembicara yang paling cocok dengan *data testing*.



Gambar 3.1 *Flowchart* Proses *Training* Sistem



Gambar 3.2 *Flowchart* Proses *Testing*



Gambar 3.3 *Flowchart* Proses Ekstraksi Fitur

Gambar 3.2 adalah *flowchart* untuk melakukan ekstraksi fitur MFCC [ZIL11]. Pada tahap *frame blocking* ini sinyal akan dibagi menjadi beberapa *frame*. Tiap *frame* tersebut akan di-*filter* dengan *window* Hamming pada proses *windowing*. Sinyal yang telah di *windowing* akan diproses dengan Fast Fourier Transform untuk diubah ke dalam domain frekuensi. Setelah itu sinyal dalam domain frekuensi di-*filter* dengan mel filterbank untuk dihitung nilai cepstrumnya.

3.3.1 Ekstraksi Fitur

Setelah melakukan *frame blocking* dan *windowing*, maka tahap selanjutnya adalah menerapkan metode Fast Fourier Transform untuk mengubah sinyal ke dalam domain frekuensi. Untuk memodelkan pendengaran manusia, maka perlu untuk menerapkan *mel filterbank* terhadap sinyal untuk melakukan

filter terhadap sinyal *input* sehingga akan dihasilkan Mel-Frequency Cepstral Coefficients.

Untuk parameter dari Mel-Frequency Ceptral Coefficients penulis menggunakan *frame length* 30 ms, *overlapping* sebesar 50%, *window* Hamming, dan jumlah *cepstrum* sebanyak 13 [ZIL11]. Untuk jumlah *filter* penulis menggunakan 32 buah *filter* karena pada penelitian yang dilakukan oleh Vibha Triwari dengan menggunakan 32 buah *filter* menghasilkan akurasi yang lebih baik [TIW10].

3.3.1.1 *Silence Removal*

Pada tahap ini, nilai sampel dari *file audio* dari -5 sampai 5 akan dihilangkan dan tidak akan di proses karena nilai amplitudo sampel tersebut sangat kecil. Sehingga jumlah sampel *file audio* tersebut akan berkurang dari 24000. Karena sampel yang nilai amplitudonya -5 sampai 5 berjumlah 1680 buah, jumlah sampel *file audio* tersebut akan menjadi 22320 buah.

3.3.1.2 *Frame Blocking*

Pada tahap awal, sinyal pada domain waktu akan dibagi menjadi beberapa bagian dalam rentang waktu tertentu. Dalam hal ini total sampel pada setiap *frame* dilambangkan dengan *N*. Dengan frekuensi sampling 8000 Hz memiliki total sampel sebanyak 22320 sampel setelah dilakukan *silence removal*. Frekuensi sampling 8000 Hz dipilih Jika dilakukan *frame blocking* dengan panjang setiap *frame* adalah 30 ms, maka dilakukan perhitungan sebagai berikut:

$$
\begin{aligned}
N &= (30 / 1000) * 8000 \\
&= 0.03 * 8000 \\
&= 240
\end{aligned}
$$

Dari hasil hitungan di atas, dihasilkan bahwa dalam satu *frame* terdapat 240 buah sampel. Banyak sampel dalam satu *frame* harus bilangan *power of two* (2^n). Karena 240 bukan merupakan bilangan *power of two*, maka akan dicari

bilangan *power of two* yang terdekat. *Power of two* terdekat akan dipilih antara *power of two* selanjutnya, 256, dan *power of two* sebelumnya, 128. Karena 240 lebih dekat dengan 256 dibandingkan dengan 128, maka akan didapatkan 256 sampel dalam satu *frame*.

Dan jika pada *file audio* tersebut memiliki total sampel sebanyak 22320 sampel dan *overlapping* sebesar 50%, maka untuk dapat mengetahui total *frame* yang dimiliki, dilakukan perhitungan sebagai berikut:

$$\text{totalFrame} = \text{round}(\text{jumlah sampel sinyal} / \tfrac{1}{2} * \text{sampelPerFrame})$$
$$= \text{round}(22320 / \tfrac{1}{2} * 256)$$
$$= \text{round}(22320 / 128)$$
$$= 174$$

Dari perhitungan di atas dihasilkan bahwa total *frame* yang dimiliki oleh *file audio* dengan melakukan *frame blocking* setiap 30 ms dan *overlapping* sebesar 50 % adalah sebanyak 174 buah *frame*.

3.3.1.3 *Windowing*

Pada proses ini sinyal pada tiap *frame* dikalikan dengan sinyal yang telah diproses dengan *window* Hamming. Proses tersebut didefinisikan pada persamaan 2.11 dan untuk mendapatkan nilai dari *window* Hamming didefinisikan pada persamaan 2.12. Berikut ini adalah nilai-nilai sampel dari *frame* ke-1:

Tabel 3.1 Nilai Sampel

Frame	Sampel ke-	Nilai Sampel
1	1	1,0
1	2	-2,0
1	3	2,0
1	4	-2,0
1	5	2.0
1	6	-2,0

Frame	Sampel ke-	Nilai Sampel
1	7	2,0
1	8	-2,0
k1	9	2,0
1	10	-3,0
1	11	4,0
1	12	-4,0
...	...	...
1	249	3,0
1	250	-4,0
1	251	4,0
1	252	-3,0
1	253	2.0
1	254	-1.0
1	255	1,0
1	256	-1,0

Untuk mendapatkan nilai *window* Hamming, maka perhitungan akan dilakukan dengan persamaan 2.12 seperti berikut ini:

n = 0 (diasumsikan bahwa setiap index dimulai dari angka 0)

$w[0] = 0.54 – 0.46 \cos (2 * 0 * \pi / 256)$

$= 0,08$

.

.

n = 255

$w[255] = 0.54 – 0.46 \cos (2 * 255 * \pi / 256)$

$= 0,08$

Perhitungan diatas menunjukkan nilai *window* Hamming yang didapatkan untuk *N* sampel pada suatu *frame*. Setelah itu, untuk melakukan *windowing* pada sinyal, nilai dari *window* Hamming yang telah didapat dikalikan dengan nilai amplitudo pada sampel ke-*n*:

$n = 0$

$y_1(0) = x_1(0) * w(0)$

$= 1{,}0 * 0{,}08$

$= 0{,}08$

.

.

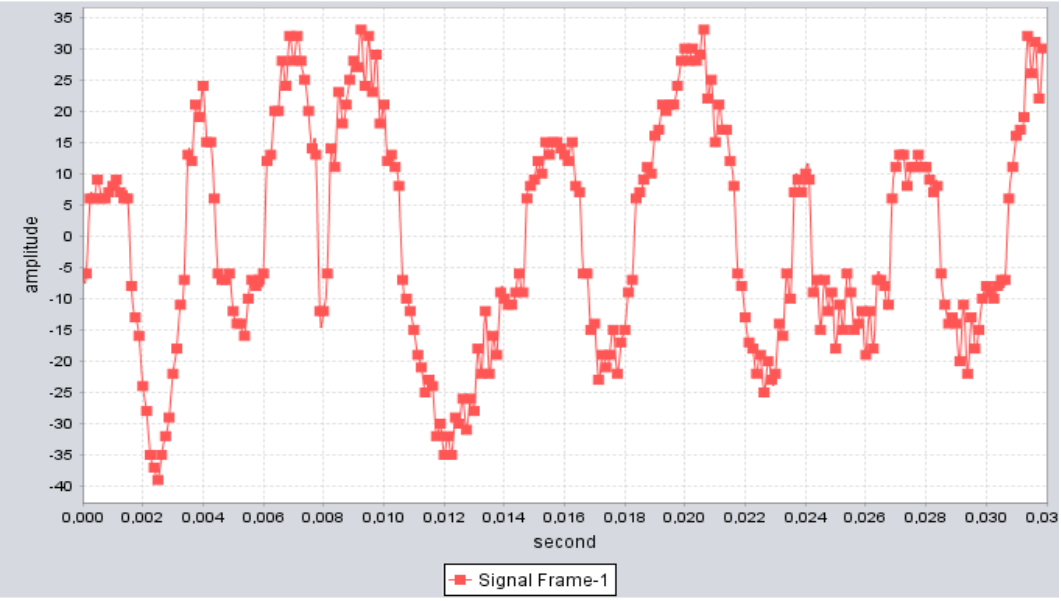
$n = 255$

$y_1(255) = x_1(255) * w(255)$

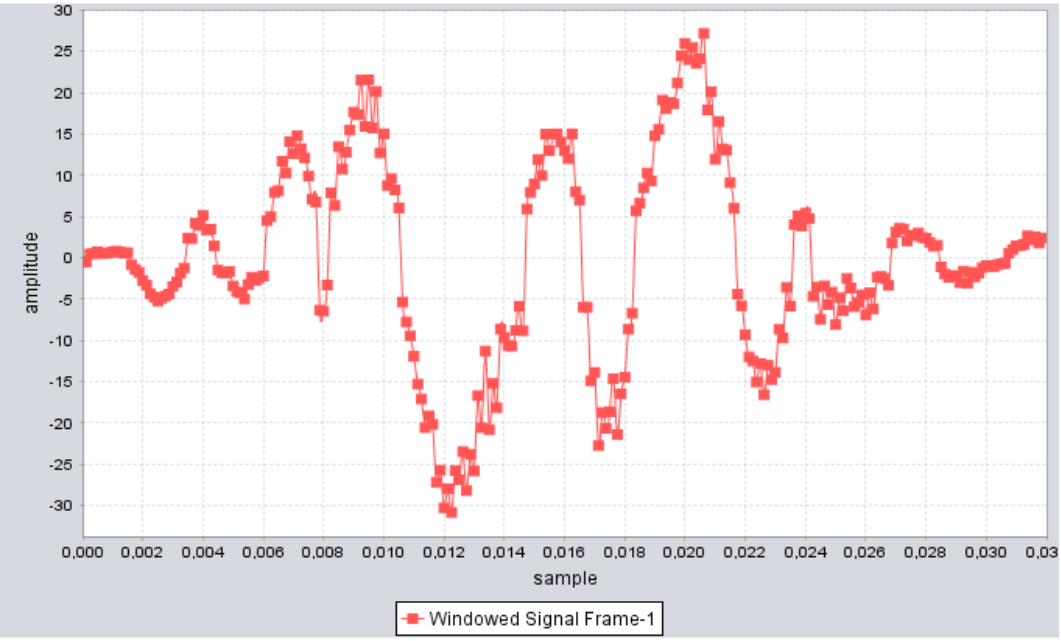
$= -1{,}0 * 0{,}08$

$= -0{,}08$

Dengan demikian hasil *windowing* dari sinyal pada sampel ke-0 pada *frame* ke-1 adalah 0,08, dan seterusnya. Nilai hasil *windowing* ini yang akan diproses untuk tahap Fast Fourier Transform.



Gambar 3.4 *Sebelum Windowing*



Gambar 3.5 *Setelah Windowing*

3.3.1.4 *Zero Padding*

Pada *frame* terakhir biasanya akan kurang jumlah sampelnya dari jumlah sampel tiap *frame* yang telah ditetapkan sehingga jika pada *frame* terakhir jumlah sampelnya masih kurang dari 256, maka akan dilakukan proses *zero padding* dengan menambahkan nilai 0 sampai jumlah sampel pada *frame* terakhir tersebut jumlah sampelnya adalah 256.

3.3.1.5 Fast Fourier Transform

Dalam proses Fast Fourier Transform, sinyal dalam domain waktu akan diubah ke dalam domain frekuensi. Nilai-nilai frekuensi yang terdapat dalam sinyal dari *file audio* nantinya akan digunakan dalam tahap *filtering* untuk mendapatkan koefisien-koefisien yang merupakan *acoustic vector*.

Fast Fourier Transform memiliki beberapa tahap yang tergantung pada jumlah input sampel. Jumlah input sampel disini didefinisikan sebagai panjang *frame* atau jumlah sampel pada setiap *frame* yang sudah ditentukan pada tahap *frame blocking*. Tahap dalam Fast Fourier Transform ditentukan dengan $\text{Log}_2(N)$.

Jika panjang *frame* N = 256, maka akan ada 8 tahap dalam Fast Fourier Transform, seperti yang diperlihatkan pada gambar 2.4.

Dari hasil *windowing* di atas sudah didapatkan nilai sampel $y_i(n)$ pada *frame* ke-1. Nilai $y_i(n)$ adalah sebagai berikut:

Tabel 3.2 Nilai Sampel Yang Sudah Di *Windowing*

Frame	**Sampel ke-**	**Nilai Sampel Yang Sudah Di *Windowing***
1	1	-0,48
1	2	-0,48
1	3	0,48
1	4	0,49
1	5	0,74
1	6	0,50
1	7	0,51
1	8	0,61
1	9	0,71
1	10	0,82
1	11	0,66
1	12	0,58
1	13	0,60
1	14	-0,83
1	15	-1,39
...	...	
1	242	-1,00
1	243	-1,07
1	244	-0,83
1	245	-0,70
1	246	-0,68
1	247	0,56

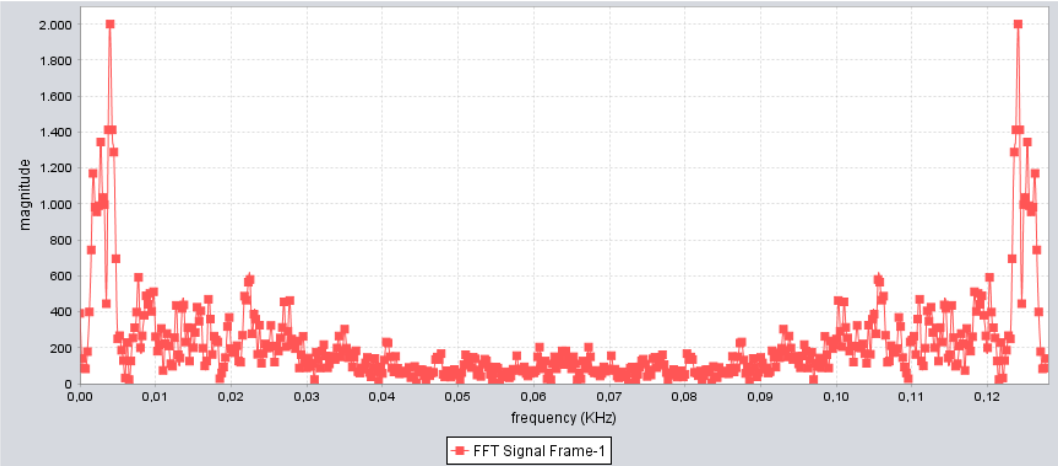
Frame	Sampel ke-	Nilai Sampel Yang Sudah Di *Windowing*
1	248	1,00
1	249	1,42
1	250	1,48
1	251	1,61
1	252	2,67
1	253	2,14
1	254	2,52
1	255	1,77
1	256	2,40

Nilai-nilai tersebut adalah nilai *amplitude* sinyal dalam domain waktu. Setelah melalui proses windowing, nilai-nilai di atas akan diproses dengan metode Fast Fourier Tranform Algoritma Radix-2. Hasil dari Fast Fourier Transform adalah nilai *magnitude* dalam domain frekuensi sebagai berikut:

Tabel 3.3 Nilai *Magnitude*

Frame	Sampel ke-	*Magnitude*
1	1	59,05
1	2	137,31
1	3	724,73
1	4	955,90
1	5	254,35
1	6	796,77
1	7	989,30
1	8	699,93
1	9	423,24
1	10	302,65

Frame	Sampel ke-	*Magnitude*
1	11	214,38
1	12	488,63
1	13	149,70
1	14	274,72
...	...	...
1	243	145,86
1	244	274,72
1	245	149,70
1	246	488,63
1	247	214,38
1	248	302,65
1	249	423,24
1	250	699,93
1	251	989,30
1	252	796,77
1	253	254,35
1	254	955,90
1	255	724,73
1	256	137,31



Gambar 3.6 Nilai sampel dalam domain frekuensi

Fast Fourier Transform ini diterapkan pada setiap sampel dari setiap *frame* yang telah ditentukan pada tahap *frame blocking* dan telah dilakukan *windowing* pada setiap sampel tersebut. Setelah menghitung nilai *magnitude* dari semua sampel *input*, tahap selanjutnya adalah melakukan *filter* terhadap sinyal dalam domain frekuensi tersebut dengan *mel filterbank* untuk melakukan ekstraksi fitur.

3.3.1.6 Mel-Frequency Wrapping

Setelah mendapatkan sinyal dalam domain frekuensi, tahap selanjutnya adalah melakukan *filter* pada sinyal untuk setiap *frame*. *Filter* yang digunakan adalah *mel filterbank*. *Mel filterbank* yang terdiri dari *Triangular Window* sebanyak *M*.

Untuk membuat *mel filterbank*, ditentukan terlebih dahulu batas atas dan batas bawah frekuensi. Batas bawah adalah 0 Hz dan batas atas adalah 4000 Hz. Setelah itu frekuensi batas atas dan bawah diubah ke dalam nilai mel dengan persamaan 2.13.

$$M(0) = 1125 \ln (1 + (f / 700))$$
$$= 1125 \ln (1 + (0 / 700))$$
$$= 0$$

$$M(4000) = 1125 \ln (1 + (f / 700))$$
$$= 1125 \ln (1 + (4000 / 700))$$
$$= 2142,27$$

Dengan nilai *M* adalah 32 untuk mempermudah perhitungan, maka artinya *mel filterbank* memiliki *Triangular Window* sebanyak 32 buah, diperlukan 34 titik. Setelah mengetahui nilai mel untuk frekuensi batas bawah dan frekuensi batas atas, nilai mel pada 34 titik antara titik awal dan titik akhir berjarak secara linear antara 0 dan 2142,27 dihitung dengan persamaan 2.13 sehingga akan didapatkan hasil seperti di bawah ini:

Tabel 3.4 Nilai Mel

Mel ke-	Nilai Mel
1	0
2	64,92
3	129,83
4	194,75
5	259,67
6	324,59
7	389,5
8	454,42
9	519,34
10	584,25
11	649,17
12	714,09
13	779,01
14	843,92
15	908,84
16	973,76

Mel ke-	Nilai Mel
17	1038,67
18	1103,59
19	1168,51
20	1233,43
21	1298,34
22	1363,26
23	1428,18
24	1493,1
25	1558,01
26	1622,93
27	1687,85
28	1752,76
29	1817,68
30	1882,6
31	1947,52
32	2012,43
33	2077,35
34	2142,27

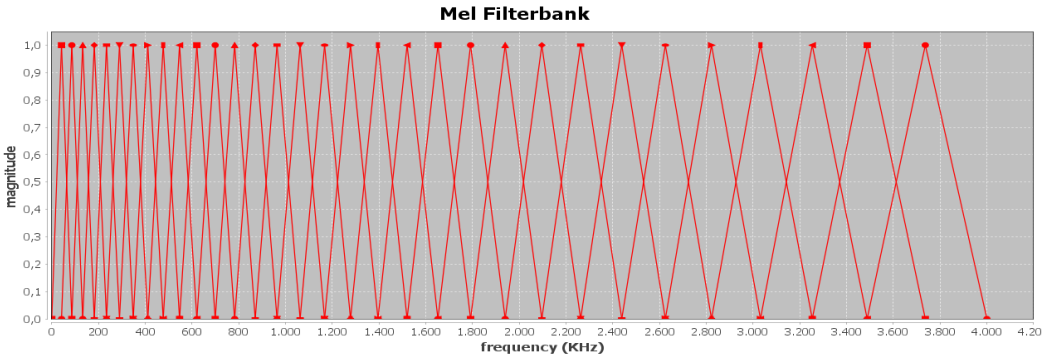
Setelah itu nilai mel dikonversikan ke dalam Hertz (Hz) dengan persamaan 2.14 sehingga akan didapatkan hasil sebagai berikut:

Tabel 3.5 Nilai Frekuensi

No	Frekuensi
1	0
2	41,58
3	85,63
4	132,3
5	181,74

No	Frekuensi
6	234,12
7	289,6
8	348,39
9	410,66
10	476,64
11	546,53
12	620,58
13	699,02
14	782,13
15	870,17
16	963,44
17	1062,25
18	1166,93
19	1277,83
20	1395,32
21	1519,78
22	1651,64
23	1791,33
24	1939,32
25	2096,1
26	2262,19
27	2438,15
28	2624,56
29	2822,04
30	3031,26
31	3252,9
32	3487,71
33	3736,47
34	4000

Gambar di bawah ini menunjukkan ada 32 buah *Triangular Window* yang membentuk *mel filterbank* yang ditentukan berdasarkan nilai-nilai frekuensi di atas.



Gambar 3.7 *Mel Filterbank*

Titik batas bawah yang bernilai 0 Hz dan titik batas atas yang bernilai 4000 Hz. Nilai-nilai frekuensi di atas harus dikonversikan ke dalam nilai FFT bin yang terdekat dengan menggunakan persamaan 2.15 yang kemudian akan digunakan untuk menghitung nilai-nilai *filterbank* dengan persamaan 2.16. Berikut nilai-nilai FFT bin yang dihasilkan:

Tabel 3.6 Nilai FFT Bin

m	$f[m]$
0	0
1	0
2	1
3	2
4	2
5	3
6	4
7	5
8	6
9	7

m	$f[m]$
10	8
11	9
12	11
13	12
14	13
15	15
16	16
17	18
18	20
19	22
20	24
21	26
22	28
23	31
24	33
25	36
26	39
27	41
28	45
29	48
30	52
31	55
32	59
33	63

Filterbank pada gambar 3.4 didefinisikan dengan persamaan 2.16 dengan jumlah *filter* sebanyak 32 buah, sehingga setelah *filterbank* didefinisikan, sinyal input akan di-*filter* dengan mengalikan nilai filterbank dengan nilai *magnitude*

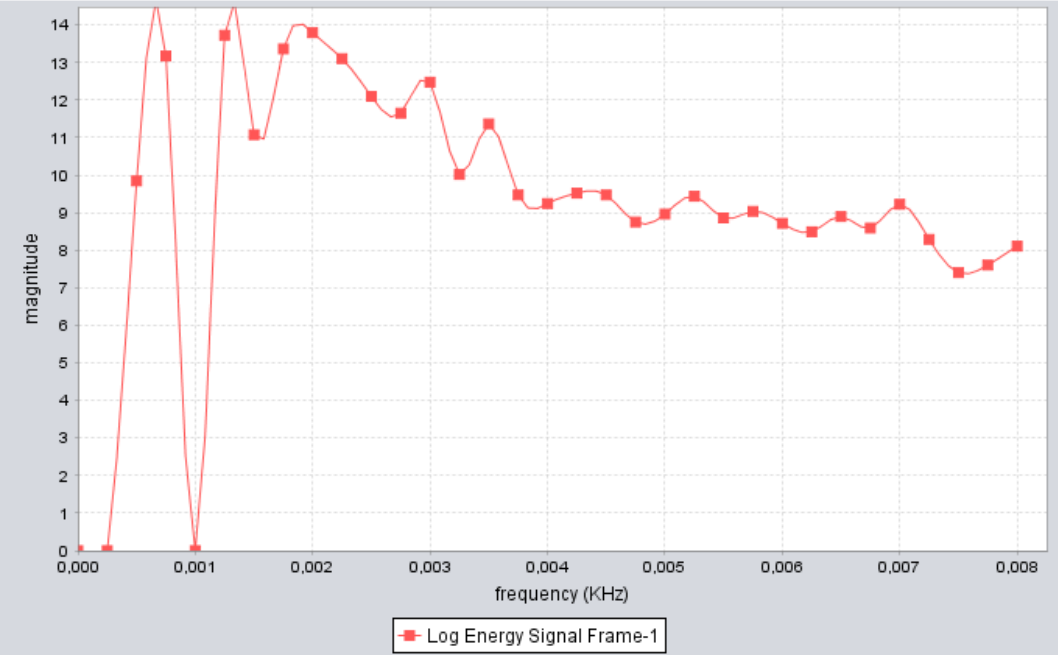
dari sinyal untuk setiap *frame*. Jumlah total nilai-nilai setiap *filter* pada *filterbank* yang telah dihitung adalah sebanyak jumlah sampel tiap *frame*, maka jumlah totalnya adalah sebanyak 32 * 174.

Proses pem-*filter*-an sinyal didefinisikan melalui persamaan 2.17 untuk mendapatkan nilai *log-energy* untuk masing-masing *filter*. Hasil dari pem-*filter*-an sinyal ini adalah nilai *log-energy* sebanyak jumlah *filter* yang ada.

Tabel 3.7 Nilai *Log Energy*

Frame	*Log energy* ke-	Nilai *log energy*
1	1	0,0
1	2	9,84
1	3	13,17
1	4	0,0
1	5	13,73
1	6	11,08
1	7	13,36
1	8	13,79
1	9	13,10
1	10	12,10
1	11	11,65
1	12	12,48
1	13	10,02
1	14	11,36
1	15	9,47
1	16	9,25
1	17	9,52
1	18	9,47
1	19	8,75
1	20	8,96
1	21	9,44

Frame	*Log energy* ke-	Nilai *log energy*
1	22	8,86
1	23	9,03
1	24	8,72
1	25	8,49
1	26	8,90
1	27	8,59
1	28	9,22
1	29	8,29
1	30	7,40
1	31	7,61
1	32	8,11



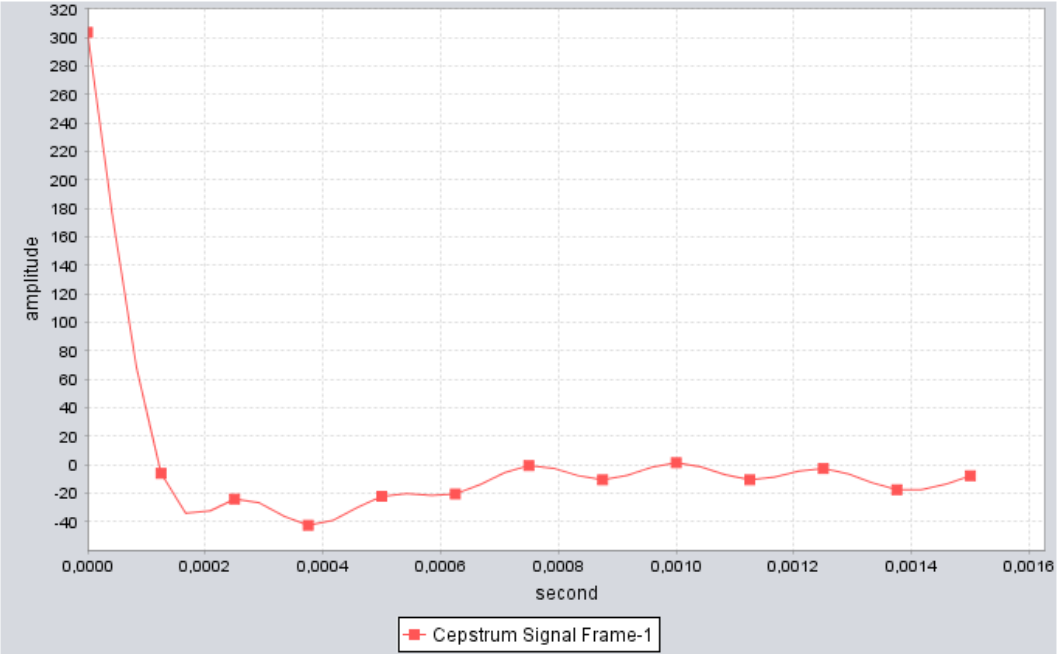
Gambar 3.8 Nilai sampel setelah di *filter* dengan *Mel Filterbank*

3.3.1.7 Discrete Cosine Transform

Untuk mendapatkan *acoustic vector* atau Mel-Frequency Cepstral Coefficients, nilai *log-energy* dihitung dalam persamaan 2.18. Pada perhitungan DCT ini, koefisien ke-2 sampai ke-13 pertama saja yang akan disimpan sebagai *acoustic vector*. Untuk *frame* ke-1 dapat diketahui nilai DCT-nya sebagai berikut:

Tabel 3.8 Nilai Discrete Cosine Transform

Frame	DCT ke-	Nilai DCT
1	2	-6,00
1	3	-24,06
1	4	-42,48
1	5	-22,10
1	6	-20,47
1	7	-0,53
1	8	-10,50
1	9	1,45
1	10	-10,50
1	11	-2,60
1	12	-17,50
1	13	-7,73



Gambar 3.9 Nilai sampel hasil Discrete Cosine Transform

3.4 Analisis *Feature Matching*

Dalam melakukan *feature matching* ada dua tahap yang harus dilakukan, yaitu tahap *training* dan tahap *testing*. Pada tahap *training* sistem akan melakukan proses *training* terhadap *machine learning* dari sistem agar dapat melakukan perbandingan terhadap *input*. Sedangkan pada tahap *testing*, sistem akan dilakukan perbandingan antara *data input* terhadap database untuk dicari data yang paling cocok dengan *data input*.

Tabel 3.9 Tabel Objek

Pembicara ke-*i*	***Frame* ke-*j***	**Suara ke-*k***	***Value* (*Value*$_n$) Suara ke-*j* *Pembicara ke-i***
Pembicara 1 (Anshori)	*Frame* ke-1	Suara ke-1	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-1	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-1	$Value_1, Value_2, \dots, Value_{12}$
	...	...	...
	Frame ke-1	Suara ke-15	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-15	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-15	$Value_1, Value_2, \dots, Value_{12}$
Pembicara 2 (David C)	*Frame* ke-1	Suara ke-16	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-16	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-16	$Value_1, Value_2, \dots, Value_{12}$
	...	...	...
	Frame ke-1	Suara ke-30	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-30	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-30	$Value_1, Value_2, \dots, Value_{12}$
Pembicara 3 (Diana)	*Frame* ke-1	Suara ke-31	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-31	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-31	$Value_1, Value_2, \dots, Value_{12}$
	...	...	...
	Frame ke-1	Suara ke-45	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-45	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-45	$Value_1, Value_2, \dots, Value_{12}$
Pembicara 4 (Larissa)	*Frame* ke-1	Suara ke-46	$Value_1, Value_2, \dots, Value_{12}$
	...	Suara ke-46	$Value_1, Value_2, \dots, Value_{12}$
	Frame ke-174	Suara ke-46	$Value_1, Value_2, \dots, Value_{12}$
	...	...	...
	Frame ke-1	Suara ke-60	$Value_1, Value_2, \dots, Value_{12}$

Pembicara ke-*i*	***Frame* ke-*j***	**Suara ke-*k***	***Value* (*Value*$_n$) Suara ke-*j* *Pembicara ke-i***
	...	Suara ke-60	$Value_1, Value_2, ..., Value_{12}$
	Frame ke-174	Suara ke-60	$Value_1, Value_2, ..., Value_{12}$
Pembicara 5 (Yabes)	*Frame* ke-1	Suara ke-61	$Value_1, Value_2, ..., Value_{12}$
	...	Suara ke-61	$Value_1, Value_2, ..., Value_{12}$
	Frame ke-174	Suara ke-61	$Value_1, Value_2, ..., Value_{12}$
	...	...	...
	Frame ke-1	Suara ke-75	$Value_1, Value_2, ..., Value_{12}$
	...	Suara ke-75	$Value_1, Value_2, ..., Value_{12}$
	Frame ke-174	Suara ke-75	$Value_1, Value_2, ..., Value_{12}$

Objek dari data yang akan di *cluster* dimodelkan dengan tabel 3.1. Setiap pembicara memiliki beberapa *file* suara, dan setiap *file* suara memiliki nilai koefisien Discrete Cosine Transform hasil dari ekstraksi fitur.

Pada K-Means Clustering ini akan diproses tiap *frame* dari tiap data suara, sehingga memungkinkan untuk *frame* dari data suara yang sama akan berakhir pada *cluster* yang berbeda.

3.5 Analisis Proses *Training*

Dalam proses *learning* ini sistem akan melakukan *training* sebagai acuan untuk proses pengenalan pembicara. *Machine learning* yang digunakan dalam sistem ini adalah algoritma K-Means Clustering. Proses yang dilakukan pada tahap *training* digambarkan pada *flowchart* gambar 2.7.

Pertama, jumlah *cluster* ditentukan terlebih dahulu. Jumlah *cluster* ditentukan berdasarkan jumlah pembicara. Jika total pembicara ada 5 orang seperti di dalam tabel 3.9, maka jumlah *cluster* adalah 5 buah.

Langkah selanjutnya adalah menentukan *centroid* sesuai dengan jumlah *cluster* yang telah ditetapkan. *Centroid* yang dipilih adalah *centroid* yang

mewakili setiap pembicara. Setiap pembicara memiliki sejumlah *file* suara dan dari *file* suara tersebut dipilih salah satu secara acak sebagai *centroid* awal.

Setelah mengetahui *centroid*-nya, selanjutnya adalah melakukan proses *clustering* objek-objek atau dalam hal ini adalah *acoustic vector* yang didapat dari proses ekstraksi fitur. Setelah melakukan *clustering*, *centroid* untuk setiap *cluster* akan ditentukan kembali. Proses ini akan dilakukan secara berulang sampai *cluster* yang ada stabil atau dengan kata lain tidak ada objek yang berpindah. Perhitungan dalam *flowchart* pada gambar 2.7 akan dibahas secara rinci pada subbab berikut.

3.5.1 *Determine Centroids*

Sebelum menentukan *centroid*, jumlah *cluster* yang ingin dibuat ditentukan terlebih dahulu. Jika jumlah *cluster* yang ditentukan adalah 5, maka *centroid* awal dipilih 5 buah *centroid*. Jumlah *cluster* ditentukan berdasarkan jumlah pembicara. Jika total pembicara ada 5 orang seperti di dalam tabel 3.9, maka jumlah *cluster* adalah 5 buah.

Centroid awal dipilih secara acak, satu dari *file* suara setiap pembicara. *Centroid* yang dipilih adalah *centroid* yang mewakili setiap pembicara. *Centroid* awal yang dipilih ditunjukkan pada tabel di bawah ini.

Tabel 3.10 *Centroid* Awal

No	*Centroid* Awal
1	-43.15; -33.53; -35.04; -10.14; -24.85; -0.65; -10.28; 13.83; -7.66; 12.89; -10.82; 1.61
2	-42.16; -29.0; -52.18; -16.67; -29.68; 4.11; -18.05; 7.91; -15.5; 1.4; -17.29; -0.19
3	-50.47; -59.22; -76.67; -21.52; -22.8; 13.9; -4.47; 21.04; -40.79; -11.81; -33.42; 2.37
4	-38.09; -57.94; -70.28; -13.93; -27.18; 13.34; -6.38; 25.41; -24.2; -13.81; -49.21; -5.86
5	-49.97; -31.57; -47.98; -17.28; -23.89; 5.2; -12.79; 7.78; -17.08; 3.95; -32.9; -12.82

3.5.2 *Nearest-Neighbor Cassification*

Selanjutnya dihitung jarak antara tiap objek dengan *centroid* dengan menggunakan Euclidean Distance (persamaan 2.19). Jarak frame ke-*x* Suara ke-*x* ke *centroid* 1 dihitung dengan cara berikut:

$$\sqrt{(V1_1 - C1_1)^2 + (V1_2 - C1_2)^2 + \cdots + (V1_{12} - C1_{12})^2}$$

Keterangan:

$V1_n$ = Nilai fitur ke-*n* suara ke-1

$C1_n$ = Nilai fitur ke-*n centroid* ke-1

Sama halnya juga dengan objek lain, perhitungan dilakukan dengan cara seperti di atas. Hasil perhitungan antara objek dan *centroid* ditunjukkan pada tabel 3.11 berkut ini:

Tabel 3.11 Hasil Perhitungan Jarak Suara ke-1 dan *Centroid*

Objek	*Centroid*	Jarak
frame ke-1, Suara ke-1	*Centroid* ke-1	48.40
frame ke-1, Suara ke-1	*Centroid* ke-2	39.32
frame ke-1, Suara ke-1	*Centroid* ke-3	86.67
frame ke-1, Suara ke-1	*Centroid* ke-4	71.36
frame ke-1, Suara ke-1	*Centroid* ke-5	70.52

Dari hasil di atas dapat dilihat jarak antara tiap objek dan *centroid*, selanjutnya adalah melakukan *clustering* dengan mengambil objek dengan jarak yang terdekat dengan *centroid*.

3.5.3 *Objects Clustering*

Pada tahap ini akan akan ditentukan *cluster* dari masing-masing frame tiap *file* suara. Dari tabel 3.11 diketahui bahwa jarak frame ke-1 dari Suara ke-1 dengan *Centroid* ke-2 adalah yang paling dekat sehingga frame ke-1 dari Suara ke-1 akan masuk ke *cluster* 2

Dari hasil perhitungan sebelumnya dihasilkan anggota dari tiap *cluster* pada iterasi pertama adalah sebagai berikut:

Tabel 3.12 Tabel *Cluster* Iterasi ke-1

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	David C	185	47	48	33	20	37
2	*Cluster* 2	Diana	6151	1500	1152	1356	1032	1111
3	*Cluster* 3	Anshori	2082	234	423	825	479	121
4	*Cluster* 4	Yabes	746	97	69	47	459	74
5	*Cluster* 5	Larissa	4064	744	933	449	671	1267

Tabel 3.12 merupakan hasil *objects clustering* pada iterasi pertama. Pada tabel diatas diketahui jumlah *frame* dari *file* suara tiap pembicara yang berada pada *cluster* tersebut. Misalnya pada *cluster* 1 jumlah *frame* dari *file* suara pembicara Anshori adalah 47. Demikian juga untuk jumlah *frame* dari *file* suara pembicara David C pada *cluster* 1 adalah 48, dan seterusnya. Jumlah *frame* tersebut merupakan total dari seluruh *file* suara yang berbeda-beda untuk setiap pembicara.

3.5.4 Update Centroids

Setelah mengetahui anggota *cluster*, *centroid* diperbaharui karena anggota *cluster* telah berubah. Perhitungan untuk menentukan *centroid* dilakukan sebagai berikut:

Centroid ke-1 = $\left(\frac{\sum_{n=1}^{N} V_1(n)}{N} \text{ , ... , } \frac{\sum_{n=1}^{N} V_{12}(n)}{N}\right)$

Keterangan:

$V_i(n)$ = nilai koefisien ke-*i* dari anggota *cluster* ke-*n*

N = jumlah anggota *cluster*

n = indeks anggota *cluster*

Perhitungan di atas adalah perhitungan *centroid* baru pada cluster ke-1 karena *centroid* ke-1 mewakili *cluster* ke-1. *Centroid* yang baru ini dihitung

berdasarkan nilai yang dimiliki dari anggota *cluster* yang ingin dihitung *centroid*-nya. *Cluster* 1 yang anggotanya berjumlah 185 *frame* dari tiap *file* suara, maka *centroid* baru dihitung berdasarkan nilai-nilai yang dimiliki oleh masing-masing *frame* suara. Begitu juga dengan *cluster* lainnya. Hingga pada saat tidak ada lagi anggota *cluster* yang berpindah, maka iterasi dinyatakan telah selesai. Dan hasil akhirnya adalah seperti pada Tabel 3.12.

Tabel 3.13 Hasil akhir K-Means Clustering

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	David C	3982	822	907	754	503	996
2	*Cluster* 2	Diana	3329	621	437	857	892	522
3	*Cluster* 3	Anshori	1905	357	546	529	261	212
4	*Cluster* 4	Yabes	2536	786	718	91	74	867
5	*Cluster* 5	Larissa	1476	36	17	479	931	13

BAB IV
IMPLEMENTASI & PENGUJIAN

Pada bab ini dibahas mengenai analisis kebutuhan dari sistem yang akan dibangun dan bagaimana perancangan dari sistem yang akan dibangun.

4.1 Lingkungan Perangkat Pengembangan

Pada *subbab* ini akan dibahas mengenai lingkungan perangkat keras dan lunak serta implementasi yang dilakukan dalam penelitian ini.

Spesifikasi perangkat keras digunakan dalam pengembangan aplikasi ini antara lain:

1. Notebook dengan *processor* Intel(R) Core(TM) i5-2450M CPU @ 2,5GHz
2. *Hard disk* 500 GB
3. *RAM* 4 GB
4. *Keyboard* dan *mouse*

Perangkat lunak yang digunakan dalam pengembangan aplikasi ini antara lain:

1. Sistem operasi Microsoft Windows 7 64 bit
2. NetBeans IDE 8.0.2
3. Java Development Kit 1.8.0 update 31
4. Java Runtime Environment 1.8.0 update 31
5. XAMPP 3.2.1

Adapun batasan dalam perangkat lunak yang dikembangkan adalah:

1. Pembatasan dalam pembuatan *database* dimana untuk setiap *file* suara dibatasi dengan pengucapan 3 angka antara 0-9 dalam durasi waktu 3 detik.

Aplikasi yang dibuat hanya dapat memproses *file audio* dengan *format* WAV, frekuensi sampling 8000 Hz, *channel mono,* dan *audio format* 16 bit.

4.2 Implementasi Perangkat Lunak

Perangkat lunak yang dikembangkan pada penelitian ini mengimplementasikan kelas-kelas dan *method-method* yang digunakan untuk membangun sistem pengidentifikasi suara. Berikut ini adalah daftar *method* dan variabel global dari tiap kelas:

Tabel 4.1 Daftar Atribut Global dan *Method* Kelas

SpeakerIdentification Class	
Atribut	**Deskripsi**
ArrayList<String> speakerNames	Untuk menyimpan nama-nama pembicara yang ada diambil dari *database*.
boolean silenceRemoval	Untuk menentukan status silence removal. Jika melakukan silence removal, maka nilainya adalah true, dan sebaliknya.
boolean doFrameBlocking	Untuk menentukan status frame blocking. Jika melakukan frame blocking, maka nilainya true, dan sebaliknya.
boolean doWindowing	Untuk menentukan status windowing. Jika melakukan windowing, maka nilainya true, dan sebaliknya.
DatabaseAccess dao	Objek yang digunakan untuk mengakses *database*.
MathHelper mathHelper	Untuk melakukan operasi matematika tertentu, seperti Euclidean Distance dan mencari nilai *power of two*.
AudioFileProcessor audioFileProcessor	Untuk memproses *file audio* yang diinput. Memproses nilai-nilai bit dalam *file audio* menjadi nilai-nilai sampel dalam 16 bit.
double[][] removedSilenceSignal	Untuk menyimpan nilai-nilai sampel yang sudah dihilangkan *silence*-nya.
ArrayList<double[][]> signals	Untuk menyimpan nilai-nilai sampel dari *file audio* ke dalam *list*.

ArrayList<HashMap<Integer, double[][]>> framedSignalsList	Untuk menyimpan nilai-nilai sampel *file audio* yang sudah di frame blocking ke dalam suatu *list*.
ArrayList<HashMap<Integer, double[][]>> windowedSignalsList	Untuk menyimpan nilai sampel *file audio* yang sudah di windowing ke dalam suatu *list*.
ArrayList<HashMap<Integer, double[][]>> fftSignalsList	Untuk menyimpan nilai *magnitude file audio* yang didapat setelah proses Fast Fourier Transform ke dalam suatu *list*.
ArrayList<HashMap<Integer, double[][]>> logEnergiesList	Untuk menyimpan nilai *log energy file audio* yang didapat setelah proses Mel-Frequency Wrapping ke dalam suatu *list*.
ArrayLIst<HashMap<Integer, double[][]>> cepstrumsList	Untuk menyimpan nilai *cepstrum file audio* yang didapat setelah proses Discrete Cosine Transform ke dalam suatu *list*.
AudioInputStream audioInputStreamTesting	Objek yang digunakan untuk menyimpan *input* berupa *audio file* pada saat proses *testing*.
AudioInputStream[] audioInputSteamTraining	Objek yang digunakan untuk menyimpan *input* berupa *audio file* pada saat proses *training*. *Input file audio* pada tahap *training* bisa lebih dari satu.
int frameBlockingLength	Untuk menyimpan durasi *frame blocking*.
int overlapping	Untuk menyimpan overlapping untuk proses *frame blocking*.
int samplePerFrame	Untuk menyimpan jumlah sampel dalam satu *frame*.
JFileChooser chooser	Untuk memilih file dari direktori di komputer.
File file	Untuk menyimpan informasi *file* yang dipilih.
String idSave	Untuk menyimpan nama pembicara yang akan disimpan ke *database*.
String idFile	Untuk menyimpan nama *file* suara yang akan disimpan ke *database*.
int sampleRate	Untuk menyimpan frekuensi sample rate dari *file* suara yang diinput.

int channels	Untuk menyimpan jumlah channel yang dimiliki oleh *file* suara yang diinput.
int totalFrame	Untuk menyimpan total *frame* dalam satu *file* suara.
int numFilters	Untuk menyimpan jumlah *filter* yang dipakai pada proses Mel Frequency Wrapping.
int coefficients	Untuk menyimpan jumlah koefisien untuk proses Discrete Cosine Transform.
KMeansClustering kmc	Objek untuk melakukan proses K-Means Clustering.
double[] frequency	Untuk menyimpan nilai-nilai frekuensi hasil konversi dari nilai-nilai mel yang akan digunakan untuk menggambarkan mel filterbank.
double[][] logEnergy	Untuk menyimpan nilai-nilai log energy hasil dari proses Mel Frequency Wrapping pada nilai-nilai *magnitude* dalam domain frekuensi pada *frame* tertentu.
HashMap<Integer, double[][]> logEnergies	Untuk menyimpan nilai-nilai log energy hasil dari proses Mel Frequency Wrapping pada nilai-nilai *magnitude* dalam domain frekuensi pada tiap *frame*.
MelFrequencyWrapping mfw	Objek yang digunakan untuk melakukan proses Mel Frequency Wrapping pada nilai-nilai sampel pada *frame* tertentu.
AcousticVector acousticVector	Objek untuk menyimpan informasi dari nilai-nilai fitur.
ArrayList<Feature> features	Untuk menyimpan hasil ekstraksi fitur tiap *file audio*.
ArrayList<Integer> totalFramePerFile	Untuk menyimpan total *frame* tiap *file audio* setelah proses ekstraksi fitur.
ArrayList<AcousticVector> acousticVectors	Untuk menyimpan AcousticVector ke dalam *list*.
ArrayList<double[]> centroids	Untuk menyimpan nilai-nilai *centroid* untuk proses K-Means Clustering.

Method	Parameter	Deskripsi	Return
initialize	-	Untuk proses inisialisasi atribut-atribut tertentu.	-

displayMelFilterBank	• double[] frequencies • String title • String domainAxis • String rangeAxis	Untuk menampilkan gambar Mel Filterbank.	-
display	• double[] samples • String title • String domainAxis • String rangeAxis • String info • int indexFrame • AudioInputStream audioInputStream • JPanel panel	Untuk menampilkan grafik sinyal *file audio*, seperti *full signal*, *framed signal*, *windowed signal*, hasil Fast Fourier Transform, hasil Mel-Frequency Wrapping, hasil Discrete Cosine Transform.	-
saveAll	-	Untuk menyimpan ke dalam *database* semua fitur pada ArrayList<Feature>	-
MathHelper Class			
Method	***Parameter***	**Deskripsi**	**Return**
isPowerOfTwo	• int n	Untuk memeriksa apakah bilangan n adalah bilangan *power of two* atau bukan.	• boolean (n & (n-1)) == 0
getNextPowerOfTwo	• int n	Untuk mendapatkan	• int n

		bilangan *power of two* selanjutnya yang terdekat.	
getPreviousPowerOfTwo	• int n	Untuk mendapatkan bilangan *power of two* sebelumnya yang terdekat.	• int n
euclideanDistance	• double[] a, double[] b	Untuk menghitung Euclidean Distance antara nilai-nilai pada *array* a dan b.	• double Math.sqrt(sum)
AudioFileProcessor Class			
Atribut		**Deskripsi**	
AudioInputStream[] audioInputStreamTesting		Objek yang digunakan untuk menyimpan *input* berupa *audio file* pada saat proses *testing*.	
AudioInputStream[] audioInputStreamTraining		Objek yang digunakan untuk menyimpan *input* berupa *audio file* pada saat proses *training*. *Input file audio* pada tahap *training* bisa lebih dari satu.	
File[] testingFile		Untuk menyimpan *input file testing* yang dipilih.	
File[] trainingFiles		Untuk menyimpan *input file training* yang dipilih.	
ArrayList<double[][]> signals		Untuk menyimpan nilai-nilai sampel *file audio* yang diinput dan hanya diambil 24000 sampel pertama karena nilai sampel yang didapat dari *file audio* jumlahnya lebih dari 24000.	
double[][] sampleContainer		Untuk menyimpan nilai-nilai sampel *file audio* yang diinput.	
Method	***Parameter***	**Deskripsi**	**Return**

readFileDataTraining	• File[] trainingFiles	Untuk membaca *file* yang di-*input* pada proses *training*.	-
readFileDataTesting	• File testingFiles	Untuk membaca *file* yang di-*input* pada proses *testing*.	-
getSampleRate	• AudioInputStream audioInputStream	Untuk mendapatkan nilai *sample rate* dari *file audio* yang di-*input*.	• float audioInputStream.getFormat().getSampleRate()
getChannelNumbers	• AudioInputStream audioInputStream	Untuk mendapatkan jumlah *channel* dari *file audio* yang di-*input*.	• float audioInputStream.getFormat().getChannels()
getSampleContainerThreeSecond	-	Berfungsi untuk mendapatkan nilai-nilai sampel dalam 3 detik pertama dari *file audio* yang diinput.	• double[][] sampleThreeSecond
getAudioInputStreamTesting	-	Untuk mengakses *file audio* pada proses *testing*.	• AudioInputStream audioInputStreamTesting
getAudioInputStreamTraining	-	Untuk mengakses *file audio* pada proses *training*.	• AudioInputStream[] audioInputStreamTraining
getSignals	-	Untuk mengakses nilai-nilai sampel *file audio*.	• ArrayList<double[][]> signals

createSampleArrayCollection	-	Untuk membaca nilai-nilai sampel dari *file audio*.	-
getSamples	• byte[] eightBitByteArray	Untuk mengubah nilai-nilai byte dari *file audio* menjadi nilai-nilai sampel.	• double[][] samples

FeatureExtractor Class	
Atribut	**Deskripsi**
boolean silenceRemoval	Untuk menentukan status silence removal. Jika melakukan silence removal, maka nilainya adalah true, dan sebaliknya.
boolean doFrameBlocking	Untuk menentukan status frame blocking. Jika melakukan frame blocking, maka nilainya true, dan sebaliknya.
boolean doWindowing	Untuk menentukan status windowing. Jika melakukan windowing, maka nilainya true, dan sebaliknya.
int sampleRate	Untuk menyimpan frekuensi sample rate dari *file* suara yang diinput.
int channels	Untuk menyimpan jumlah channel yang dimiliki oleh *file* suara yang diinput.
double[][] signal	Untuk menyimpan nilai sampel *file audio*.
double[][] removedSilenceSignal	Untuk menyimpan nilai sampel *file audio* yang sudah dihilangkan *silence*-nya.
MathHelper mathHelper	Untuk melakukan operasi matematika tertentu, seperti Euclidean Distance dan mencari nilai *power of two*.
FrameBlocking frameBlocking	Untuk melakukan frame blocking terhadap nilai-nilai sampel *file audio*.
Windowing windowing	Untuk melakukan windowing terhadap nilai-nilai sampel *file audio*.
FastFourierTransform fastFourierTransform	Untuk menghitung nilai *magnitude* dari nilai-nilai sampel *file audio* dengan Fast

	Fourier Transform.
MelFrequencyWrapping melFrequencyWrapping	Untuk melakukan Mel-Frequency Wrapping terhadap nilai-nilai *magnitude* untuk mendapatkan nilai-nilai *log energy* sesuai dengan jumlah *filter*.
DiscreteCosineTransform dct	Untuk menghitung nilai-nilai *cepstrum* dengan memproses nilai-nilai *log energy*.
int sampleLastFrame	Untuk menyimpan jumlah sampel pada *frame* terakhir.
int samplePerFrame	Untuk menyimpan jumlah sampel tiap *frame*.
int totalFrame	Untuk menyimpan jumlah *frame* yang dihasilkan.
double[][] framedSignal	Untuk menyimpan nilai-nilai sampel *frame* tertentu.
HashMap<Integer, double[][]> framedSignals	Untuk menyimpan nilai-nilai sampel seluruh *frame* setelah proses frame blocking
HashMap<Integer, double[][]> windowedSignals	Untuk menyimpan nilai-nilai sampel seluruh *frame* setelah proses windowing dan zero padding.
HashMap<Integer, double[][]> fftSignals	Untuk menyimpan nilai-nilai *magnitude* pada tiap *frame* dalam domain frekuensi setelah di proses dengan Fast Fourier Transform.
int numFilters	Untuk menyimpan jumlah *filter* yang akan dipakai untuk proses Mel Frequency Wrapping.
double[] frequency	Untuk menyimpan nilai-nilai frekuensi hasil konversi dari nilai-nilai mel yang akan digunakan untuk menggambarkan mel filterbank.
HashMap<Integer, double[][]> logEnergies	Untuk menyimpan nilai-nilai log energy hasil dari proses Mel Frequency Wrapping pada nilai-nilai *magnitude* dalam domain frekuensi pada tiap *frame*.
int coefficients	Untuk menyimpan jumlah koefisien untuk proses Discrete Cosine Transform.
HashMap<Integer, double[][] cepstrums	Untuk menyimpan hasil proses Discrete Cosine Transform pada tiap *frame*.

Method	***Parameter***	**Deskripsi**	**Return**

extractingFeatures	-	Untuk melakukan proses ekstraksi fitur.	-
removeSilence	-	Untuk melakukan proses silence removal.	-
doFrameBlocking	-	Untuk melakukan proses frame blocking.	-
doWindowing	-	Untuk melakukan proses windowing.	-
doFFT	-	Untuk melakukan proses Fast Fourier Transform.	-
doDiscreteCosineTransform	-	Untuk melakukan proses Discrete Cosine Transform.	-
getRemovedSilenceSignal	-	Untuk mengakses nilai-nilai sampel yang telah dihilangkan *silence*-nya.	• double[][] removedSilenceSignal
geFramedSignals	-	Untuk mengakses nilai-nilai sampel yang telah di frame blocking.	• HashMap<Integer, double[][]> framedSignals
getWindowedSignals	-	Untuk mengakses nilai-nilai sampel yang telah di windowing.	• HashMap<Integer, double[][]> windowedSignals
getFftSignals	-	Untuk mengakses nilai-nilai	• HashMap<Integer, double[][]> fftSignals

		magnitude setelah proses Fast Fourier Transform.	
getLogEnergies	-	Untuk mengakses nilai-nilai *log energy* setelah proses Mel-Frequency Wrapping.	• HashMap<Integer, double[][]> logEnergies
getCepstrums	-	Untuk mengakses nilai-nilai *cepstrum* setelah proses Discrete Cosine Transform.	• HashMap<Integer, double[][] cepstrums
getNumFilters	-	Untuk mengakses jumlah *filter* yang digunakan pada proses Mel-Frequency Wrapping.	• int numFilters
getCoefficients	-	Untuk mengakses jumlah koefisien yang digunakan pada proses Discrete Cosine Transform.	• int coefficients
getTotalFrame	-	Untuk mengakses jumlah *frame* yang dihasilkan setelah proses ekstraksi fitur.	• int totalFrame
getFrequency	-	Untuk mengakses nilai-nilai frekuensi pada Mel	• double[] frequency

		Filterbank.	
FrameBlocking Class			
Atribut		**Deskripsi**	
int frameBlockingLength		Untuk menyimpan panjang frame blocking.	
int overlapping		Untuk menyimpan overlapping untuk proses *frame blocking*.	
int sampleRate		Untuk menyimpan frekuensi sample rate dari *file* suara yang diinput.	
int channels		Untuk menyimpan jumlah channel yang dimiliki oleh *file* suara yang diinput.	
int sampleLastFrame		Untuk menyimpan jumlah sampel pada *frame* terakhir.	
int samplePerFrame		Untuk menyimpan jumlah sampel tiap *frame*.	
int totalFrame		Untuk menyimpan total *frame* dalam satu *file* suara.	
double[][] signal		Untuk menyimpan nilai-nilai sampel *file audio*.	
double[][] framedSignal		Untuk menyimpan nilai-nilai sampel *frame* tertentu.	
HashMap<Integer, double[][]> framedSignals		Untuk menyimpan nilai-nilai sampel seluruh *frame* setelah proses frame blocking	
MathHelper mathHelper		Untuk melakukan operasi matematika tertentu, seperti Euclidean Distance dan mencari nilai *power of two*.	
Method	***Parameter***	**Deskripsi**	**Return**
doFrameBlocking	-	Untuk melakukan proses frame blocking.	-
geFramedSignals	-	Untuk mengakses nilai-nilai sampel yang telah di frame blocking.	• HashMap<Integer, double[][]> framedSignals
getSamplePerFrame	-	Untuk mengakses jumlah	• int samplePerFrame

		sampel tiap *frame*.	
getSampleLastFrame	-	Untuk mengakses jumlah sampel *frame* terakhir.	• int sampleLastFrame
getTotalFrame	-	Untuk mengakses jumlah *frame* yang dihasilkan pada proses frame blocking.	• int totalFrame
Windowing Class			
Atribut		**Deskripsi**	
HashMap<Integer, double[][]> signals		Untuk menyimpan nilai-nilai sampel yang akan di windowing setelah proses frame blocking.	
int channels		Untuk menyimpan jumlah channel *audio file*.	
int sampelPerFrame		Untuk menyimpan jumlah sampel dalam satu *frame*.	
int totalFrame		Untuk menyimpan total *frame* dalam satu *audio file*.	
int sampleLastFrame		Untuk menyimpan jumlah sampel *frame* terakhir.	
double[][] window		Untuk menyimpan nilai window Hamming dalam satu *frame*.	
double[][] windowedSignal		Untuk menyimpan nilai sampel yang sudah di windowing dalam satu *frame*.	
HashMap<Integer, double[][]> windows		Untuk menyimpan nilai-nilai window Hamming setiap *frame*.	
HashMap<Integer, double[][]> windowedSignals		Untuk menyimpan nilai-nilai sampel yang sudah di windowing dalam setiap *frame*	
Method	***Parameter***	**Deskripsi**	**Return**
doWindowing	-	Berfungsi untuk menghitung nilai-nilai sampel yang melalui proses	-

		windowing tiap frame.	
computeWindow	-	Berfungsi untuk menghitung nilai-nilai window.	-
windowHamming	• int N	Berfungsi untuk menghitung nilai-nilai window Hamming.	• double[] w
getWindowedSignals	-	Untuk mengakses nilai-nilai sampel yang telah di windowing.	• HashMap<Integer, double[][]> windowedSignals

FastFourierTransform Class	
Atribut	**Deskripsi**
int channels	Untuk menyimpan jumlah channel *audio file*.
int totalFrame	Untuk menyimpan total *frame* dalam satu *audio file*.
HashMap<Integer, double[][]> signals	Untuk menyimpan nilai-nilai sampel yang akan di proses.
double[][] fftSignal	Untuk menyimpan nilai-nilai magnitude pada *frame* tertentu dalam domain frekuensi setelah di proses dengan Fast Fourier Transform.
HashMap<Integer, double[][]> fftSignals	Untuk menyimpan nilai-nilai *magnitude* pada tiap *frame* dalam domain frekuensi setelah di proses dengan Fast Fourier Transform.

Method	***Parameter***	**Deskripsi**	**Return**
doFastFourierTransform	-	Untuk melakukan proses Fast Fourier Transform.	-

getFftSignals	-	Untuk mengakses nilai-nilai *magnitude* setelah proses Fast Fourier Transform.	• HashMap<Integer, double[][]> fftSignals

MelFrequencyWrapping Class	
Atribut	**Deskripsi**
HashMap<Integer, double[][]> fftSignals	Untuk menyimpan nilai-nilai *magnitude* pada tiap *frame* dalam domain frekuensi setelah di proses dengan Fast Fourier Transform.
int channels	Untuk menyimpan jumlah channel *audio file*.
int totalFrame	Untuk menyimpan total *frame* dalam satu *audio file*.
int sampelPerFrame	Untuk menyimpan jumlah sampel dalam satu *frame*.
int numFilters	Untuk menyimpan jumlah *filter* Mel Filterbank yang digunakan.
int sampleRate	Untuk menyimpan nilai frekuensi sampling
double lowerFrequency	Untuk menyimpan nilai frekuensi batas bawah.
double upperFrequency	Untuk menyimpan nilai frekuensi batas atas.
double[] mels	Untuk menyimpan nilai-nilai mel.
double[] frequency	Untuk menyimpan nilai-nilai frekuensi hasil konversi dari nilai-nilai mel.
int[] fftBin	Untuk menyimpan nilai-nilai fft bin.
double[] logEnergy	Untuk menyimpan nilai-nilai log energy sebanyak jumlah *filter* yang sudah di tentukan.
double[] filterbank	Untuk menyimpan nilai-nilai Mel Filterbank.
HashMap<Integer, double[][]> logEnergies	Untuk menyimpan nilai-nilai log energy hasil dari proses Mel Frequency Wrapping pada nilai-nilai *magnitude* dalam domain frekuensi pada tiap *frame*.

Method	*Parameter*	Deskripsi	Return
doMelFrequencyWrapping	-	Untuk melakukan proses Mel-Frequency Wrapping.	-
convertFrequencyToMel	-	Berfungsi untuk mengubah nilai-nilai frekuensi menjadi nilai-nilai mel.	• double[] mels
convertMelToFreq	-	Berfungsi untuk mengubah nilai-nilai mel menjadi nilai-nilai frekuensi.	• double[] frequency
convertToFrequency	• double mel	Untuk merubah nilai mel ke nilai frekuensi.	• double (700 * (Math.exp(mel / 1125) - 1))
convertToFFTBin	-	Berfungsi untuk mendapatkan indeks fft terdekat dengan memproses nilai-nilai frekuensi yang didapat setelah mengubah nilai-nilai mel menjadi nilai-nilai frekuensi.	• int[] fftBin
createFilterbank	-	Berfungsi untuk menghitung nilai-nilai filterbank.	• double[][] filterbank
computeLogEnergy	-	Berfungsi untuk	• double[] logEnergy

		menghitung nilai *log energy*.	
getFrequency	-	Untuk mengakses nilai-nilai frekuensi yang sudah diubah dari nilai mel.	• double[] frequency
getFilterbank	-	Untuk mengakses nilai-nilai filterbank.	• double[][] filterbank
getLogEnergies	-	Untuk mengakses nilai-nilai *log energy* setelah proses Mel-Frequency Wrapping.	• HashMap<Integer, double[][]> logEnergies

DiscreteCosineTransform Class			
Atribut		**Deskripsi**	
int channels		Untuk menyimpan jumlah *channel audio file*.	
int totalFrame		Untuk menyimpan total *frame* dalam satu *audio file*.	
int M		Untuk mendefinisikan jumlah *filter*.	
int numCoefficients		Untuk mendefinisikan jumlah koefisien DCT.	
HashMap<Integer, double[][]> logEnergies		Untuk menyimpan nilai-nilai log energy hasil dari proses Mel Frequency Wrapping pada nilai-nilai *magnitude* dalam domain frekuensi pada tiap *frame*.	
double[] cepstrum		Untuk menyimpan nilai-nilai hasil DCT.	
HashMap<Integer, double[][] cepstrums		Untuk menyimpan hasil proses Discrete Cosine Transform pada tiap *frame*.	
Method	***Parameter***	**Deskripsi**	**Return**
doDCT	-	Untuk melakukan proses	-

		Discrete Cosine Transform.	
getCepstrums	-	Untuk mengakses nilai-nilai *cepstrum* setelah proses Discrete Cosine Transform.	• HashMap<Integer, double[][] cepstrums
KMeansClustering Class			
Atribut		**Deskripsi**	
int dimension		Untuk mendefinisikan dimensi K-Means Clustering.	
int totalData		Mendefinisikan jumlah data yang akan di proses oleh K-Means Clustering.	
int totalCentroid		Untuk Mendefinisikan jumlah *centroid* yang akan digunakan.	
double distance		Untuk menyimpan jarak yang dihitung dengan Euclidean Distance antara *centroid* dan *acoustic vector*.	
double minDistance		Untuk menyimpan jarak terkecil dari suatu *acoustic vector* ke *centroid*.	
boolean stillMoving		Untuk mendefinisikan apakah masih ada perubahan anggota *cluster* atau tidak.	
int[] clusterMembers		Untuk menyimpan jumlah anggota tiap *cluster*.	
ArrayList<AcousticVector> acousticVectors		Untuk menyimpan *acoustic vector* yang akan di proses oleh K-Means Clustering.	
ArrayList<double[]> centroids		Untuk menyimpan seluruh nilai *centroid*.	
ArrayList<String> speakerNames		Untuk menyimpan nama-nama pembicara yang disimpan di *database*.	
String[] clusterNames		Untuk menyimpan nama-nama tiap *cluster*.	
ArrayList<VectorWeight> vectorWeights		Untuk menyimpan bobot tiap pembicara pada tiap *cluster*.	
MathHelper mathHelper		Untuk melakukan operasi matematika tertentu, seperti Euclidean Distance dan mencari nilai *power of two*.	
Method	***Parameter***	**Deskripsi**	**Return**

doClustering	-	Untuk melakukan proses K-Means Clustering.	-
nearestNeighborClassification	-	Berfungsi untuk menentukan jarak terdekat dari suatu *acoustic vector* terhadap suatu *centroid* dan menentukan termasuk *cluster* mana *acoustic vector* tersebut.	-
updateCentroids	-	Berfungsi untuk menghitung kembali nilai *centroid*.	-
setAcousticVectors	• ArrayList<AcousticVector> acousticVectors	Untuk melakukan *setting* nilai *acoustic vector* yang akan diproses dengan K-Means Clustering	-
setCentroids	• ArrayList<double[][]> Centroids	Untuk melakukan *setting* nilai-nilai *centroid* awal.	-
isThereSameClusterName	-	Untuk memeriksa apakah ada nama *cluster* yang sama atau tidak.	• boolean thereIsSame
setClusterName	• int clusterNumber	Untuk menentukan nama	-

		setiap *cluster*.	
getClusterName	-	Untuk mendapatkan semua nama *cluster* yang sudah ditentukan.	• ArrayList<String> speakerNames
getClusterNameAtIndex	• int index	Untuk mendapatkan nama cluster yang sudah ditentukan pada indeks tertentu.	• String clusterName.get(index)
setWeightsEverySpeakerPerCluster	-	Untuk menentukan bobot tiap pembicara pada tiap *cluster*.	-
getSortedWeights	-	Untuk mendapatkan *list* bobot pembicara tiap *cluster* yang sudah di-*sort* dari yang terbesar hingga yang terkecil.	• ArrayList<VectorWeight> vectorWeights
getZeroWeight	• String key	Untuk mendapatkan jumlah bobot nol, yaitu pembicara yang memiliki bobot nol pada *cluster* tertentu.	• int zeroWeight
clusterNameIsNotExist	• int index • ArrayList<Integer>	Untuk memeriksa apakah nama *cluster* sudah ada atau	• boolean notExist

	indexSpeakerWeight	belum.	
isThereNoClusterMember	-	Untuk memeriksa apakah ada *cluster* yang tidak memiliki anggota atau tidak.	• boolean thereIsNoClusterMember
setSpeakerNames	• ArrayList<String> speakerNames	Untuk menentukan nama-nama pembicara untuk proses K-Means Clustering.	-
setMathHelper	• MathHelper mathHelper	Untuk mengisi variabel mathHelper.	-

4.2.1 Implementasi *Silence Removal*

Pada proses *silence removal* ini sinyal dengan nilai sampel yang kecil akan dihilangkan. Rentang nilai sampel yang akan dihilangkan adalah nilai sampel yang memiliki nilai -5 sampai 5. Berikut ini adalah *pseudocode* untuk *silence removal*:

Langkah-langkah dalam melakukan *silence removal* adalah sebagai berikut:

i. Mendeklarasikan variabel untuk menyimpan banyak nilai yang akan dihilangkan dan int removedSilenceSignal.
ii. Memeriksa nilai sampel yang nilainya -5 sampai 5 pada signal dengan melakukan for loop berdasarkan jumlah channel dan jumlah sampel.
iii. Jika nilai sampel dari signal adalah -5 sampai 5, maka tambahkan satu pada variabel removed. Proses ini terus berlanjut sampai semua nilai sampel dibaca.
iv. Setelah itu mendeklarasikan array 2 dimensi removedSilenceSignal dengan panjang array 1 x (jumlah sampel – removed).
v. Lakukan for loop untuk membaca nilai sampel yang nilai nya lebih besar dari 5 atau lebih kecil dari -5 untuk dimasukkan ke dalam array remvedSilenceSignal.

4.2.2 Implementasi *Frame Blocking*

Pada proses *frame blocking* sinyal akan dimasukkan ke dalam beberapa *frame-frame* berdasarkan panjang *frame*, *overlapping*, dan *sample rate*. Langkah-langkah dalam proses *frame blocking* adalah sebagai berikut:

i. Menentukan jumlah sampel tiap *frame* berdasarkan berdasarkan panjang *frame* dan *sample rate*.
ii. Memeriksa apakah jumlah sampel tiap frame adalah bilangan *power of two* atau bukan, jika bukan maka akan dicari bilangan *power of two* terdekat. Karena pada saat Fast Fourier Transform jumlah nilai input harus bilangan *power of two*.

iii. Menentukan total *frame* dengan melakukan operasi div terhadap panjang sinyal dengan setengah dari jumlah sampel tiap *frame*.
iv. Menghitung total sampel yang akan diperoleh jika melakukan frame blocking.
v. Mengelompokkan nilai-nilai sampel ke dalam *frame-frame* dengan menggunakan for loop. Nilai-nilai sampel tersebut akan disimpan ke dalam *hashmap* dengan *frame* sebagai *key* dari *hashmap* tersebut.
vi. Pada *frame* terakhir jumlah sampel tidak selalu sama dengan jumlah sampel yang telah ditentukan untuk setiap *frame*.

4.2.3 Implementasi *Windowing*

Pada proses *windowing* ini nilai sampel tiap *frame* akan dikalikan dengan nilai-nilai *window* Hamming. Berikut ini adalah langkah-langkah implementasi proses *windowing*:

i. Menghitung nilai-nilai window Hamming lalu menyimpannya didalam *hashmap*.
ii. Mengalikan nilai sampel dengan nilai window Hamming pada tiap *frame*.
iii. Lalu melakukan *zero padding* pada *frame* terakhir jika jumlah sampel pada *frame* terakhir belum sama dengan jumlah sampel tiap *frame* yang telah ditentukan pada proses frame blocking.

4.2.4 Implementasi Fast Fourier Transform

Pada penelitian ini penulis menggunakan *library* java Apache Commons Math 3.4.1 untuk memproses Fast Fourier Transform yang diimplementasikan. Langkah-langkah dalam proses Fast Fourier Transform ini adalah sebagai berikut:

i. Method transform dipanggil dengan parameter double[][] samples adalah nilai-nilai sampel yang telah di windowing. Parameter samples didefinisikan dengan array dua dimensi, dimensi pertama adalah jumlah channel, dimensi kedua adalah jumlah sampel input. Namun pada penelitian ini jumlah channel untuk setiap *file audio* adalah 1 (mono).

ii. Mendefinisikan array tempConversion utuk penyimpanan sementara. Didefinisikan dalam array satu dimensi sebanyak jumlah sampel input.
iii. Mendefinisikan array fft sebagai kembalian untuk method ini, mengembalikan nilai magnitude sebanyak jumlah sampel input. Didefinisikan dalam array dua dimensi, dimensi pertama adalah channel, dimensi kedua adalah jumlah sampel input. Namun pada penelitian ini jumlah channel untuk setiap *file audio* adalah 1 (mono).
iv. Melakukan inisialisasi objek FastFourierTransformer dengan parameter DftNormalization.STANDARD untuk ketentuan normalisasi *standard*.
v. Mendefinisikan objek Complex dan memanggil method transform dari objek FastFourierTransformer untuk mendapatkan nilai kompleks yang terdiri dari bilangan riil dan imajiner.
vi. Menghitung nilai magnitude dengan menjumlahkan kuadrat dari bilangan riil dan imajiner lalu mencari nilai akar kuadratnya. Nilai *magnitude* disimpan dalam tempConversion, untuk setiap channelnya.
vii. Mengisi array fftSignal dengan tempConversion untuk setiap channelnya. Mengembalikan nilai-nilai magnitude melalui array fftSignal.
viii. Nilai-nilai *magnitude* yang dihasilkan akan disimpan didalam *hashmap* untuk setiap *frame*-nya.

4.2.5 Implementasi Mel Frequency Wrapping

Pada proses implementasi Mel-Frequency Wrapping ini akan dihitung nilai *log energy* pada setiap *frame*. Berikut ini adalah langkah-langkah implementasi proses Mel-Frequency Wrapping:

i. Menghitung nilai-nilai mel dan menyipannya di dalam array double satu dimensi.
ii. Lalu mengubah nilai-nilai mel tersebut ke dalam nilai-nilai frekuensi.
iii. Menghitung nilai-nilai fft bin dengan mengkonversikan nilai-nilai frekuensi.
iv. Menghitung nilai mel filterbank.

v. Menghitung nilai *log energy* dengan mengalikan nilai-nilai *magnitude* yang dihasilkan dari proses Fast Fourier Transform dengan nilai-nilai mel filterbank.
vi. Nilai *log energy* akan disimpan didalam *hashmap* berdasarkan *frame*-nya.
vii. Proses ini akan terus berlangsung sampai seluruh *frame* diproses.

4.2.6 Implementasi Discrete Cosine Transform

Pada proses ini akan dihitung nilai DCT dengan memproses nilai *log energy*. Berikut ini adalah langkah-langkah proses implementasi Discrete osine Transform:

i. Memproses nilai-nilai *log energy* dengan rumus Discrete Cosine Transform.
ii. Mengambil dan menyimpan ke dalam *hashmap* koefisien 2-13 dari nilai Discrete Cosine Transform yang dihasilkan untuk setiap *frame*.

4.2.7 Implementasi K-Means Clustering

Proses klasifikasi pembicara pada penelitian ini menggunakan metode K-Means Clustering untuk membuat *cluster* yang masing-masing *cluster* akan berisi nilai-nilai fitur dari data suara yang di *training*. Proses K-Means Clustering diimplementasikan pada *pseudo code* di bawah ini. Langkah-langkah melakukan K-Means Clustering adalah sebagai berikut:

i. Melakukan inisialisasi variabel.
ii. Menetapkan jumlah anggota tiap *cluster* menjadi 0. Untuk iterasi pertama, jumlah anggota tiap *cluster* adalah 0. Untuk iterasi berikutnya jumlah anggota tiap *cluster* ditetapkan kembali menjadi 0 karena akan dilakukan kembali pengukuran jarak terdekat antara *centroid* tiap *cluster* terhadap setiap *file*.
iii. Menghitung jarak terdekat antara *centroid* tiap *cluster* terhadap setiap *file*. Perhitungan jarak dilakukan dengan menggunakan rumus Euclidean Distance, dengan mencari selisih terdekat antara nilai *vector* tiap *file* dengan *centroid* tiap *cluster*. Lalu mengganti nilai minDistance dengan

nilai distance jika nilai distance lebih kecil dibandingkan dengan nilai minDistance.

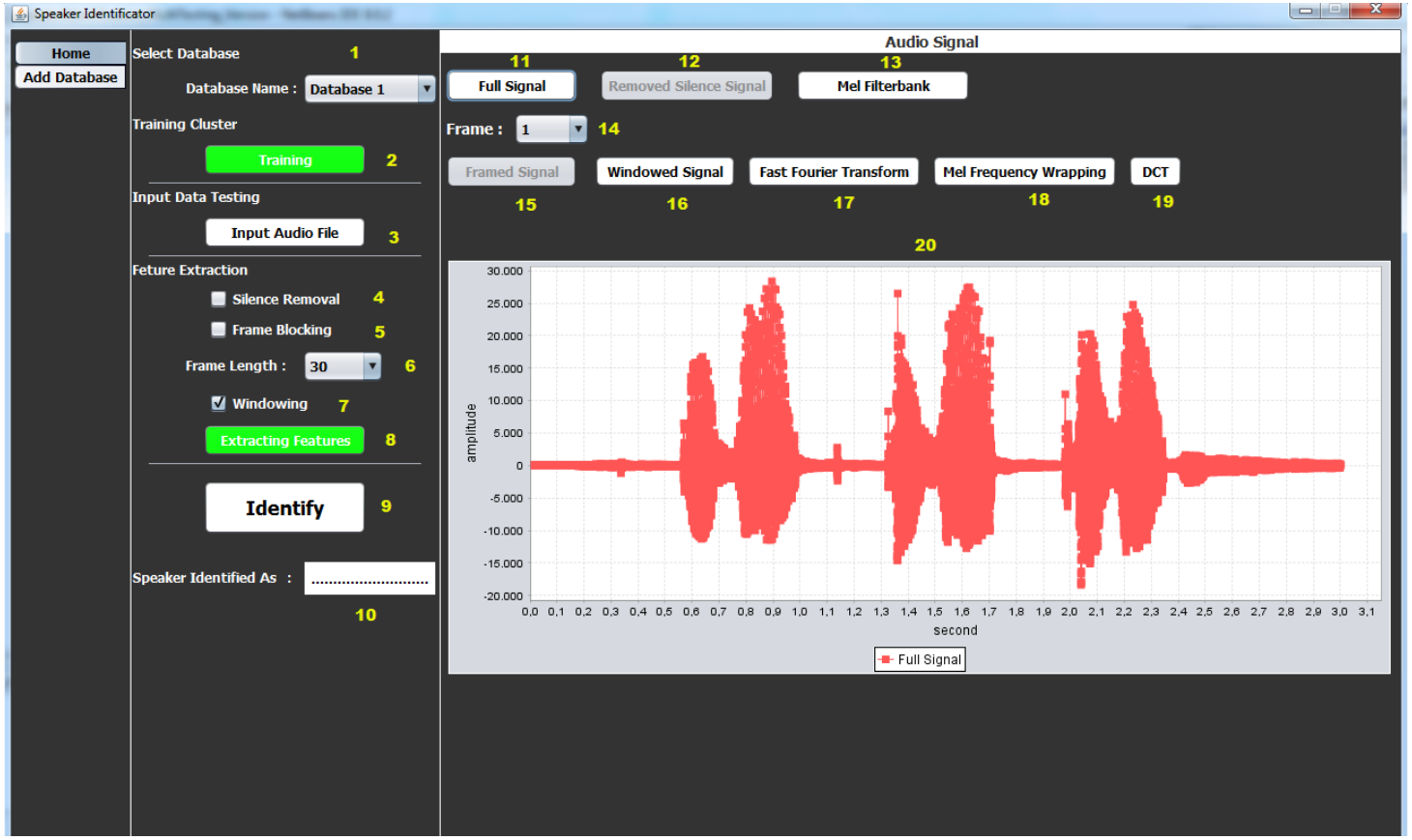
iv. Setelah menemukan *cluster* terdekat setiap *file*, maka selanjutnya adalah menentukan *cluster file* tersebut dengan method setCluster(x).

v. Menetapkan nilai tiap *centroid* menjadi 0, karena akan dilakukan *update centroid*.

vi. Lalu mengupdate nilai *centroid* tiap *cluster* karena tiap *cluster* sudah memiliki anggota. Nilai *centroid* baru dihitung berdasarkan nilai-nilai *vector* setiap anggota *cluster* yang dirata-ratakan. Pada method calculateCentroid(), nilai-nilai *vector* tiap *file* pada *cluster* tersebut akan dijumlahkan lalu dibagi dengan jumlah anggota *cluster* tersebut.

vii. Jika stillMoving berstatus *true*, maka iterasi akan dilanjutkan sampai stillMoving berstatus *false* yang artinya tidak ada lagi perpindahan anggota *cluster* dari satu *cluster* ke *cluster* lain.

4.2.8 Implementasi *Interface*



Gambar 4.1 GUI untuk *training* dan *testing*

Tabel 4.2 Keterangan GUI *training* dan *testing*

No	Keterangan
1	*Combo box* untuk memilih *database* yang akan digunakan untuk proses *training*. Pilihan *database* yang ada adalah sebagai berikut: 1. Database 1 (tidak menggunakan *silence removal* dan *frame blocking*) 2. Database 2 (menggunakan *silence removal* dan tidak menggunakan *frame blocking*) 3. Database 3 (Tidak menggunakan *silence removal* dan menggunakan *frame blocking* dengan panjang *frame* 30 ms) 4. Database 4 (Menggunakan *silence removal* dan *frame blocking* dengan panjang *frame* 30 ms) 5. Database 5 (Tidak menggunakan *silence removal* dan menggunakan *frame blocking* dengan panjang *frame* 60 ms) 6. Database 6 (Menggunakan *silence removal* dan *frame blocking* dengan panjang *frame* 60 ms) 7. Database 7 (tidak menggunakan *silence removal* dan *frame blocking*, jumlah *data training* tiap pembicara adalah 25) 8. Database 8 (Menggunakan *silence removal* dan tidak menggunakan *frame blocking*, dengan *data training* tiap pembicaranya adalah 25)
2	Tombol untuk melakukan proses *training* dengan K-Means Clustering.
3	Untuk melakukan *input file audio* untuk *testing*.
4	Checkbox untuk melakukan *silence removal*.
5	Checkbox untuk melakukan *frame blocking*.

6	*Combo box* untuk memilih panjang *frame*.
7	Checkbox untuk melakukan *windowing*.
8	Tombol untuk melakukan ekstraksi fitur.
9	Tombol untuk mengidentifikasi pembicara.
10	Label untuk menampilkan *output* hasil identifikasi.
11	Tombol untuk menampilkan grafik sinyal asli.
12	Tombol untuk menampilkan grafik sinyal yang sudah dihilangkan *silence*-nya.
13	Tombol untuk menampilkan *mel filterbank*.
14	*Combo box* untuk memilih *frame* yang akan ditampilkan grafik sinyalnya.
15	Tombol untuk menampilkan grafik sinyal pada *frame* yang dipilih.
16	Tombol untuk menampilkan grafik sinyal yang sudah di *windowing* pada *frame* yang sudah dipilih.
17	Tombol untuk menampilkan grafik nilai *magnitude* yang dihasilkan dari proses Fast Fourier Transform pada *frame* yang sudah dipilih.
18	Tombol untuk menampilkan grafik nilai *log energy* pada *frame* yang sudah dipilih.
19	Tombol untuk menampilkan grafik nilai DCT pada *frame* yang sudah dipilih.
20	*Panel* untuk menampilkan grafik sinyal.

Gambar 4.2 GUI untuk memasukkan data ke dalam *database*

Tabel 4.3 Keterangan GUI Untuk Menambahkan Data

No	Keterangan
1	*Combo box* untuk memilih *database* yang akan digunakan untuk menyimpan data atau menghapus data. Pilihan *database* yang ada adalah sebagai berikut: 1. Database 1 (tidak menggunakan *silence removal* dan *frame blocking*) 2. Database 2 (menggunakan *silence removal* dan tidak menggunakan *frame blocking*) 3. Database 3 (Tidak menggunakan *silence removal* dan menggunakan *frame blocking* dengan panjang *frame* 30 ms) 4. Database 4 (Menggunakan *silence removal* dan *frame blocking* dengan panjang *frame* 30 ms) 5. Database 5 (Tidak menggunakan *silence removal* dan menggunakan *frame blocking* dengan panjang *frame* 60 ms) 6. Database 6 (Menggunakan *silence removal* dan *frame blocking* dengan panjang *frame* 60 ms) 7. Database 7 (tidak menggunakan *silence removal* dan *frame blocking*, jumlah *data training* tiap pembicara adalah 25) 8. Database 8 (Menggunakan *silence removal* dan tidak menggunakan *frame blocking*, dengan *data training* tiap pembicaranya adalah 25)
2	Untuk melakukan *input file audio* untuk *training*.
3	*Checkbox* untuk melakukan *silence removal*.
4	*Checkbox* untuk melakukan *frame blocking*.
5	*Combo box* untuk memilih panjang *frame*.

6	*Checkbox* untuk melakukan *windowing*.
7	Tombol untuk melakukan ekstraksi fitur.
8	*Combo box* untuk menentukan id pembicara untuk disimpan di *database*.
9	Tombol untuk menyimpan ke *database*.
10	*Textfield* untuk ID *file* yang ingin dihapus dari *database*.
11	Tombol untuk menghapus data dari *database*.
12	Tombol untuk menampilkan grafik sinyal asli.
13	Tombol untuk menampilkan grafik sinyal yang sudah dihilangkan *silence*-nya.
14	Tombol untuk menampilkan *mel filterbank*.
15	*Combo box* untuk memilih *frame* yang akan ditampilkan grafik sinyalnya.
16	Tombol untuk menampilkan grafik sinyal pada *frame* yang dipilih.
17	Tombol untuk menampilkan grafik sinyal yang sudah di *windowing* pada *frame* yang sudah dipilih.
18	Tombol untuk menampilkan grafik nilai *magnitude* yang dihasilkan dari proses Fast Fourier Transform pada *frame* yang sudah dipilih.
19	Tombol untuk menampilkan grafik nilai *log energy* pada *frame* yang sudah dipilih.
20	Tombol untuk menampilkan grafik nilai DCT pada *frame* yang sudah dipilih.
21	*Panel* untuk menampilkan grafik sinyal.

4.3 Pengujian

Pengujian sistem dilakukan dengan menguji apakah sistem sudah mampu mengidentifikasi pembicara dengan tepat. Proses identifikasi dilakukan dengan menghitung jarak antara nilai fitur pembicara terhadap *cluster* terdekat.

Dalam penelitian ini pengujian yang akan dilakukan adalah:

1. Seberapa besar presentase akurasi dari sistem yang dikembangkan dapat mengenali pembicara dengan tepat.
2. Apakah K-Means Clustering dapat memodelkan pembicara dengan baik?

Dalam pengujian ini penulis melakukan perekaman suara dengan konfigurasi sebagai berikut:

Tabel 4.4 Konfigurasi perekaman suara

File Format	WAV
Frekuensi Sampling	8000 Hz
Channel	*Mono*
Audio Format	16 bit

Lalu untuk Mel-Frequency Ceptral Coefficients penulis menggunakan *frame length* 30 ms, *overlapping* sebesar 50%, *window* Hamming, dan jumlah *cepstrum* sebanyak 13 [ZIL11]. Untuk jumlah *filter* penulis menggunakan 32 buah *filter* karena pada penelitian yang dilakukan oleh Vibha Triwari dengan menggunakan 32 buah *filter* menghasilkan akurasi yang lebih baik [TIW10].

Pengujian dilakukan dengan menggunakan jumlah *data training* yang beragam dan menggunakan *data testing* sebanyak 3 *file audio*. Penulis melakukan pengujian dengan 15 data untuk setiap pembicara dan 3 buah *file audio* di luar *data training* untuk melakukan *testing*.

Untuk setiap kategori *data training* penulis melakukan percobaan sebanyak 3 kali dengan *centroid* awal yang berbeda dan diambil akurasi rata-rata dari hasil setiap percobaan.

Pada pengujian kategori 1, pengujian dilakukan dengan tanpa *silence removal* dan *frame blocking* dengan panjang *frame* 30 ms. Akurasi yang didapatkan adalah sebesar 40%. Berikut ini adalah hasil pengujiannya:

Tabel 4.5 Centroid Awal Pengujian Kategori 1

No	*Centroid* Awal
1	-55.49 -36.02 -59.4 -22.66 -26.66 3.73 -24.31 10.87 -9.69 8.94 -21.57 -7.58
2	-98.45 -54.43 -59.65 -34.67 -47.06 1.71 -18.73 10.56 -23.11 3.72 -29.98 4.39
3	-51.57 -26.96 -39.72 -13.97 -13.52 5.91 -21.93 6.39 -36.77 -8.85 -22.2 1.57
4	-47.77 -38.06 -46.66 -4.0 -28.87 -4.71 -16.45 7.68 -22.46 0.19 -21.54 2.7
5	-50.09 -34.81 -47.12 -24.05 -24.73 3.87 -13.63 6.41 -20.97 -2.15 -21.43 -4.53

Tabel 4.6 Hasil Cluster Pengujian Kategori 1

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Anshori	3658	783	710	403	861	901
2	*Cluster* 2	Diana	2165	387	571	680	325	202
3	*Cluster* 3	David C	624	153	143	44	110	174
4	*Cluster* 4	Yabes	4080	829	905	783	581	982
5	*Cluster* 5	Larissa	3498	653	476	895	928	546

Tabel 4.7 Hasil Pengujian Kategori 1

No	*Speaker*	*Identified As*		
		File 1	*File* 2	*File* 3
1	Anshori	Yabes	Yabes	Anshori
2	David C	Anshori	Yabes	Anshori
3	Diana	Yabes	Yabes	Larissa
4	Larissa	Anshori	Larissa	Larissa
5	Yabes	Yabes	Yabes	Yabes

Pada penguian kategori 2, pengujian dilakukan dengan *silence removal* dan *frame blocking* dengan panjang *frame* 30 ms. Akurasi yang dihasilkan adalah 46,7%. Berikut ini adalah hasil pengujiannya:

Tabel 4.8 *Centroid* Awal Pengujian Kategori 2

No	*Centroid* Awal
1	-56.12 -19.58 -56.28 -17.52 -34.42 14.09 -23.11 6.63 -13.84 19.98 -22.02 -6.26
2	-45.82 -25.21 -50.59 -18.14 -20.78 5.05 -10.56 4.01 -17.04 3.37 -23.38 -6.48
3	-118.83 -53.4 -58.68 -24.19 -53.1 23.75 -19.16 17.42 -17.4 0.09 -27.02 6.93
4	-44.1 -27.71 -56.88 -13.93 -29.15 11.19 -16.54 13.53 -28.18 -3.61 -29.47 -6.82
5	-58.85 -54.91 -70.45 -36.78 -27.04 -9.21 -19.09 6.53 -17.69 5.29 -26.42 -4.53

Tabel 4.9 Hasil *Cluster* Pengujian Kategori 2

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Yabes	3919	810	898	740	490	981
2	*Cluster* 2	Diana	3376	633	444	870	895	534
3	*Cluster* 3	David C	1921	358	547	538	264	214
4	*Cluster* 4	Larissa	1473	34	16	469	942	12
5	*Cluster* 5	Anshori	2539	787	720	93	70	869

Tabel 4.10 Hasil Pengujian Kategori 2

No	*Speaker*	*Identified As*		
		File 1	*File* 2	*File* 3
1	Anshori	Yabes	Yabes	Anshori
2	David C	Anshori	Yabes	Anshori
3	Diana	Yabes	Yabes	Diana
4	Larissa	Larissa	Larissa	Diana
5	Yabes	Yabes	Yabes	Yabes

Pada pengujian kategori 3, pengujian dilakukan dengan tanpa *silence removal* dan dengan *frame blocking* dengan panjang *frame* 60 ms. Akurasi yang dihasilkan adalah 26,7%. Berikut ini adalah hasil pengujiannya:

Tabel 4.11 *Centroid* Awal Pengujian Kategori 3

No	*Centroid* Awal
1	-43.9 -34.24 -47.73 1.4 -32.39 -8.27 -18.38 9.77 -41.21 -8.99 -24.31 17.48
2	-54.98 -14.86 -47.51 -17.39 -49.96 -0.45 -30.04 -0.14 -25.16 2.41 -27.35 1.99
3	-18.87 -66.37 -41.31 -21.62 -19.36 8.73 -29.52 6.26 -50.68 -21.63 -15.36 25.09
4	-18.2 -38.59 -48.77 -2.76 -28.61 -6.55 -29.73 7.91 -24.54 -6.4 -40.7 -0.24
5	-35.51 -27.58 -35.47 -18.56 -30.27 -12.67 -30.35 3.08 -20.4 9.34 -25.41 3.88

Tabel 4.12 Hasil Cluster Pengujian Kategori 3

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Larissa	797	15	14	220	546	2
2	*Cluster* 2	Diana	755	96	246	286	109	18
3	*Cluster* 3	Anshori	993	362	215	114	27	275
4	*Cluster* 4	David C	1202	339	381	95	28	359
5	*Cluster* 5	Yabes	3228	583	539	680	685	741

Tabel 4.13 Hasil Pengujian Kategori 3

No	*Speaker*	*Identified As*		
		File 1	*File* 2	*File* 3
1	Anshori	Yabes	Yabes	Yabes
2	David C	David C	Yabes	Yabes
3	Diana	Yabes	Yabes	Yabes
4	Larissa	Yabes	Yabes	Yabes
5	Yabes	Yabes	Yabes	Yabes

Pada pengujian ketegori 4, pengujian dilakukan dengan *silence removal* dan *frame blocking* dengan panjang *frame* 60 ms. Akurasi yang dihasilkan adalah 26,7%. Berikut ini adalah hasil pengujiannya:

Tabel 4.14 *Centroid* Awal Pengujian Kategori 4

No	*Centroid* Awal
1	-35.8 -18.86 -46.04 -7.78 -31.09 -4.74 -29.68 2.38 -24.22 5.55 -20.99 -1.6
2	-27.64 -22.06 -45.58 -15.92 -40.69 -6.36 -24.96 2.25 -23.54 -0.33 -30.26 0.33
3	-61.15 -10.91 -47.43 -20.48 -41.29 -1.23 -31.4 3.57 -33.86 -2.53 -34.54 9.14
4	-41.24 -18.16 -35.74 -9.36 -29.72 8.23 -20.93 5.78 -24.81 -0.14 -31.59 0.12
5	-38.03 -55.33 -45.17 -20.39 -36.85 -14.67 -36.15 10.75 -30.01 6.36 -27.88 5.25

Tabel 4.15 Hasil Cluster Pengujian Kategori 4

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Yabes	2994	542	508	632	634	678
2	*Cluster* 2	David C	983	265	311	113	59	235
3	*Cluster* 3	Diana	880	146	279	293	112	50
4	*Cluster* 4	Larissa	757	13	8	228	508	0
5	*Cluster* 5	Anshori	981	342	202	85	13	339

Tabel 4.16 Hasil Pengujian Kategori 4

No	*Speaker*	*Identified As*		
		File 1	*File* 2	*File* 3
1	Anshori	Yabes	Yabes	Yabes
2	David C	David C	Yabes	Yabes
3	Diana	Yabes	Yabes	Yabes
4	Larissa	Yabes	Yabes	Yabes
5	Yabes	Yabes	Yabes	Yabes

Pada pengujian kategori 5, pengujian dilakukan dengan tanpa *silence removal* dan tanpa *frame blocking*. Akurasi yang dihasilkan adalah 80%. Berikut ini adalah hasil pengujiannya:

Tabel 4.17 Centroid Awal Pengujian Kategori 5

No	*Centroid* Awal
1	-59.88 -22.97 -66.71 -12.86 -65.47 -9.41 -55.68 -9.05 -46.96 4.51 -44.88 1.85
2	-66.49 -20.6 -74.92 -16.5 -63.43 -9.55 -56.45 -5.27 -46.58 1.0 -45.77 5.55
3	-78.69 -27.47 -62.39 -23.95 -56.27 2.61 -53.48 -0.81 -60.11 -4.61 -44.93 11.25
4	-56.6 -44.22 -66.27 -7.88 -50.06 -6.46 -38.52 8.21 -40.44 -5.37 -48.54 -5.59
5	-46.75 -35.87 -69.1 -17.58 -55.77 -14.37 -47.17 -7.01 -39.01 1.57 -38.97 6.02

Tabel 4.18 Hasil Cluster Pengujian Kategori 5

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Anshori	18	14	1	0	0	3
2	*Cluster* 2	David C	15	1	14	0	0	0
3	*Cluster* 3	Diana	15	0	0	15	0	0
4	*Cluster* 4	Larissa	15	0	0	0	15	0
5	*Cluster* 5	Yabes	12	0	0	0	0	12

Tabel 4.19 Hasil Pengujian Kategori 5

No	*Speaker*	*Identified As*		
		File 1	*File* 2	*File* 3
1	Anshori	Anshori	Anshori	Anshori
2	David C	David C	David C	David C
3	Diana	Diana	Diana	Diana
4	Larissa	Larissa	Larissa	Larissa
5	Yabes	Anshori	Anshori	Anshori

Pada pengujian kategori 6, pengujian dilakukan dengan *silence removal* dan tanpa *frame blocking*. Akurasi yang dihasilkan adalah 80%. Berikut ini adalah hasil pengujiannya:

Tabel 4.20 *Centroid* Awal Pengujian Kategori 6

No	*Centroid* Awal
1	-66.71 -37.75 -67.74 -21.26 -60.49 -13.59 -52.67 -2.96 -39.61 7.77 -38.78 4.89
2	-76.19 -24.85 -69.18 -17.02 -62.18 -10.77 -56.95 -3.91 -48.7 4.13 -42.74 7.15
3	-76.0 -23.68 -66.42 -15.8 -62.41 5.04 -56.14 3.35 -53.67 -8.63 -55.21 12.8
4	-52.14 -39.74 -67.75 -9.86 -54.79 -4.51 -39.84 10.36 -40.29 -2.22 -49.05 -8.96
5	-52.36 -30.73 -66.92 -18.82 -55.56 -12.78 -51.65 -7.75 -42.38 3.89 -37.58 4.53

Tabel 4.21 Hasil Cluster Pengujian Kategori 6

No	*Cluster*	*Cluster Name*	Jumlah Anggota	Suara				
				Anshori	David C	Diana	Larissa	Yabes
1	*Cluster* 1	Anshori	15	12	0	0	0	3
2	*Cluster* 2	David C	18	3	15	0	0	0
3	*Cluster* 3	Diana	14	0	0	14	0	0
4	*Cluster* 4	Larissa	15	0	0	0	15	0
5	*Cluster* 5	Yabes	13	0	0	1	0	12

Tabel 4.22 Hasil Pengujian Kategori 6

No	*Speaker*	*Identified* As		
		File 1	*File* 2	*File* 3
1	Anshori	Anshori	Anshori	Anshori
2	David C	David C	David C	David C
3	Diana	Diana	Diana	Diana
4	Larissa	Larissa	Larissa	Larissa
5	Yabes	Anshori	Anshori	Anshori

Pada tabel 4.23 ditunjukkan hasil pengujian keseluruhan yang dilakukan pada penelitian ini. Akurasi terendah yang diperoleh adalah sebesar 20% dan akurasi terbesar yang dapat diperoleh pada penelitian ini adalah sebesar 80%.

Tabel 4.23 Hasil Pengujian Keseluruhan dengan 15 *Data Training*

Kategori	*Frame Length* (ms)	*Silence Removal*	*Frame Blocking*	Pengujian 1	Pengujian 2	Pengujian 3	Rata-rata
1	30	No	Yes	20%	20%	40%	26,7%
2	30	Yes	Yes	46,7%	46,7%	46,7%	46,67%
3	60	No	Yes	26,7%	26,7%	26,7%	26,7%
4	60	Yes	Yes	40%	26,7%	26,7%	31,1%
5	-	No	No	80%	80%	80%	80%
6	-	Yes	No	80%	80%	80%	80%

Penulis juga melakukan penelitian dengan menggunakan *data training* yang berbeda. *Data training* yang digunakan pada pengujian ini terdapat kalimat berita, kalimat tanya, dan kalimat perintah yang ducapkan setiap pembicara yang ditambahkan dengan *data training* sebelumnya. Dengan demikian, variasi nada yang dihasilkan oleh pembicara tersebut akan lebih bervariasi. Jumlah *data training* menjadi 25 buah *data training* untuk setiap pembicara. *Data testing* yang digunakan untuk pengujian ini pembicara mengucapkan kalimat-kalimat seperti kalimat berita, kalimat tanya, atau kalimat perintah. Berikut ini adalah hasil pengujiannya:

Tabel 4.24 Hasil Pengujian dengan 25 *Data Training*

Kategori	*Frame Length* (ms)	*Silence Removal*	*Frame Blocking*	Pengujian 1	Pengujian 2	Pengujian 3	Rata-rata
1	-	No	No	42,9%	42,9%	42,9%	42,9%
2	-	Yes	No	52,4%	33,3%	52,4%	46%

4.4 Pengujian *Threshold*

Pada *subbab* ini penulis akan menunjukkan hasil dari pengujian *threshold* yang dilakukan dalam penelitian ini. *Threshold* ini digunakan untuk memberi *output* yang menunjukkan bahwa data suara yang di uji tidak tidak terdeteksi karena pembicara tersebut tidak ada di dalam *database*. Berikut ini adalah hasil pengujiannya:

Tabel 4.25 Hasil Pengujian *Threshold* Keseluruhan

Kategori	*Threshold*	Jumlah Pembicara	*Data Training* Tiap Pembicara	*Data Testing* Tiap Pembicara di Dalam *Database*	Akurasi Deteksi	*Data Testing* Pembicara di Luar *Database*	Akurasi Penolakan
1	13	5	15	3	13,3%	3	100%
2	14	5	15	3	44,7%	3	33,3%
3	20	5	15	3	66,7%	3	66,7%
4	44	5	15	3	80%	3	0%

Yang dimaksud dengan akurasi penolakan pada tabel 4.25 adalah berapa banyak data suara pembicara yang tidak ada di *database* dianggap tidak ada. *Threshold* akan dibandingkan dengan jarak dari data suara terhadap suatu *cluster*. Jika jaraknya kurang dari atau sama dengan *threshold*, maka data suara pembicara tersebut akan dianggap ada di dalam *database* dan akan terdeteksi sebagai salah satu pembicara yang ada di dalam *database*. Dan sebaliknya jika jaraknya lebih besar dari *threshold*, maka akan dianggap tidak ada.

Dari hasil di atas ditunjukkan bahwa dengan *threshold* 20, maka akan mendapatkan hasil yang optimal karena dapat menolak data suara pembicara yang tidak ada di *database* walaupun tidak 100%, dan jua dapat memberikan akurasi deteksi yang cukup baik, yaitu sebesar 66,7%.

BAB V
KESIMPULAN & SARAN

5.1 Kesimpulan

Untuk menjawab penelitian ini, maka kesimpulan yang diambil adalah sebagai berikut:

1. Pada pengujian yang dilakukan dengan menggunakan *frame blocking*, akurasi yang dihasilkan lebih kecil dibandingkan dengan tanpa menggunakan *frame blocking*. Sehingga disimpulkan dengan tanpa menggunakan *frame blocking* hasil yang optimal dapat diperoleh dalam proses identifikasi pembicara. Dapat dilihat juga dari hasil K-Means Clustering yang diperoleh, K-Means Clustering dapat mengelompokkan pembicara jauh lebih baik saat proses ekstraksi fitur yang dilakukan tanpa menggunakan *frame blocking* dibandingkan dengan menggunakan *frame blocking*. Pengujian yang dilakukan tanpa menggunakan *frame blocking* menghasilkan akurasi tertinggi dengan akurasi sebesar 80 %.
2. Penggunaan *silence removal* pada penelitian ini memberikan peningkatan akurasi. Dapat dilihat pada pengujian kategori 1, 2, 3, dan 4, saat menggunakan *silence removal*, akurasi mengalami peningkatan dibandingkan pada saat tidak menggunakan *silence removal*.
3. Metode K-Means Clustering masih belum dapat memodelkan data dengan baik ketika pengujian dilakukan dengan menggunakan *frame blocking* pada tahap ekstraksi fitur. Hal ini dikarenakan variasi dalam satu *cluster* masih sangat besar, dimana masih banyak anggota *cluster* yang tidak seharusnya berada di dalam *cluster* tersebut. Namun pada pengujian dengan tanpa menggunakan ekstraksi fitur, K-Means Clustering dapat mengelompokkan pembicara dengan cukup baik. Seperti yang ditunjukkan pada tabel 4.18 dan tabel 4.21, *cluster* yang dihasilkan cukup baik. Sebaliknya pada tabel 4.6, 4.9, 4.12, 4.15 ditunjukkan bahwa *cluster* yang dihasilkan masih kurang baik sehingga berdampak pada akurasi pada proses identfikasi. Lalu metode K-Means Clustering ini akurasinya akan tergantung juga pada pemilihan

centroid awal. Dapat dilihat pada pengujian kategori 1 dan 4 pada tabel 4.23, dari 3 kali pengujian, akurasi yang didapatkan tidak selalu sama, karena *centroid* awalnya berbeda. Lalu pada pengujian yang menggunakan *frame blocking*, hasil dari K-Means Clustering akan memiliki *frame* dari *file* suara pembicara yang bukan merupakan *cluster* tersebut.

4. Pemilihan nilai *threshold* yang digunakan akan berpengaruh terhadap akurasi deteksi pembicara yang terdapat di *database*. Walaupun dapat menolak data suara pembicara yang tidak ada di *database*, akurasi deteksi pembicara yang ada di *database* akan menurun.

5.2 Saran

Setelah melihat kesimpulan di atas, saran yang dapat diberikan penulis untuk pengembangan selanjutnya adalah sebagai berikut:

1. Untuk pengembangan selanjutnya dapat dicoba dengan menambah jumlah *data training* untuk dapat memperoleh akurasi yang lebih baik.
2. Pada pengembangan selanjutnya sistem mampu menghilangkan *noise* pada *file* audio seperti menggunakan algoritma Voice Activity Detector (VAD) sehingga fitur fitur yang dihasilkan akan lebih mewakili informasi dari pembicara.
3. Pada penelitian selanjutnya dapat digunakan proses ekstraksi fitur lain seperti Gammatone Frequency Cepstral Coefficients. Menurut penelitian yang dilakukan oleh Md. Moinuddin dan Arunkumar N. Kanthi pada penelitian yang berjudul "*Speaker Identification Based On GFCC Using GMM*", GFCC memiliki performa yang lebih baik dibandingkan dengan MFCC baik dalam lingkungan dengan *noise* tinggi dan juga lingkungan yang tenang tidak ada *noise*. Dan juga dengan menggunakan GFCC, akurasi yang dihasilkan lebih baik dibandingkan dengan menggunakan MFCC.
4. Pada penelitian selanjutnya dapat menggunakan algoritma *machine learning* lainnya seperti, Support Vector Machine (SVM), untuk akurasi yang lebih akurat.

5. Pada penelitian selanjutnya dapat menggunakan Principal Component Analysis untuk memilih koefisien hasil dari perhitungan Discrete Cosine Transform yang paling berpengaruh.

DAFTAR PUSTAKA

[ALF12] Alfredo Maesa, Fabio Garzia, Michele Scarpiniti, and Roberto Cusani. 2012. "*Text Indepentdent Automatic Speaker Recognition System Using Mel-Frequency Cepstrum Coefficient and Gaussian Mixture Models*", Journal of Information Security, Vol. 3 No. 4.

[BAR05] Barret, Paul. 2005. "*Euclidean Distance: Raw, Normalized, and Double-Scaled Coefficients*", The Technical Whitepaper Series, Vol. 6.

[BEI11] Beigi, Homayoon. 2011. Fundamentals of Speaker Recognition. Springer Science & Business Media, New York.

[BHA13] Bhattacharjee, Utpal. 2013. "*A Comparative Study Of LPCC And MFCC Features For The Recognition Of Assamese Phonemes*", International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 1.

[HUA01] Huang, Xuedong, Alex Acero and Hsiao-Wuen Hon. 2001. Spoken Language Processing: A Guide To Theory, Algorithm And System Development. Prentice Hall, New Jersey.

[LYO11] Lyons, Richard G. 2011. Understanding Digital Signal Processing 3rd Edition. Prentice Hall, Boston.

[MAR01] Marshall, Dave. 2001. Human Hearing And Voice.

[SAR14] Sarma, Mousmita and Kandarpa Kumar Sarma. 2014. Phoneme-Based Speech Segmentation Using Hybrid Soft Computing Framework. Springer Science & Business Media, New Delhi.

[TIW10] Tiwari, Vibha. 2010. "*MFCC And Its Application In Speaker Recognition*". International Journal on Emerging Technologies.

[WUJ12] Wu, Junjie. 2012. Advances in K-means Clustering: A Data Mining Thinking. Springer Science & Business Media, Berlin.

[ZHU12] Zhu, Li and Qing Yang. 2012. "*Speaker Recognition System Based On Weighted Feature*". Physics Procedia, Vol. 25, 1515-1522.

[ZIL11] Zilvan, Vicky dan Furqon Hensan Muttaqien. 2011. "*Identifikasi Pembicara Menggunakan Algoritma VF15 dengan MFCC sebagai Pengekstraksi Ciri*". INKOM, Vol. 5, 35-45.