PENGENALAN SUARA UNTUK IDENTIFIKASI AKSARA JAWA MENGGUNAKAN *LINEAR PREDICTIVE CODING* DAN *HIDDEN MARKOV MODEL*

SKRIPSI

LABORATORIUM KOMPUTASI CERDAS & VISUALISASI

Untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer

Disusun Oleh :

GALIH JULIANTO PURNOMO

0910680056

KEMENTERIAN RISET TEKNOLOGI DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

FAKULTAS ILMU KOMPUTER

PROGRAM STUDI TEKNIK INFORMATIKA

MALANG

2015

LEMBAR PENGESAHAN

PENGENALAN SUARA UNTUK IDENTIFIKASI AKSARA JAWA MENGGUNAKAN *LINEAR PREDICTIVE CODING* DAN *HIDDEN MARKOV MODEL*

SKRIPSI

LABORATORIUM KOMPUTASI CERDAS DAN VISUALISASI

Untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer

Disusun oleh:

GALIH JULIANTO PURNOMO

NIM. 0910680056

Skripsi ini telah diuji dan dinyatakan lulus pada

1 Oktober 2015

Telah diperiksa dan disetujui oleh

Pembimbing I	**Pembimbing II**
Rekyan Regasari MP, S.T, M.T.	**Budi Darma Setiawan, S.Kom, M.Cs.**
NIK. 201102 770414 2 001	**NIP. 19841015 201404 1 002**

Mengetahui

Ketua Program Studi Teknik Informatika/Ilmu Komputer

Drs. Marji, M.T.

NIP.19670801 199203 1 001

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, Oktober 2015

Mahasiswa,

Galih Julianto Purnomo

NIM. 0910680056

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT karena berkat Rahmat, Karunia, dan Hidayah-Nya, penulis dapat menyelesaikan Laporan Tugas Akhir yang berjudul "**Pengenalan Suara untuk Identifikasi Aksara Jawa menggunakan *Linear Predictive Coding* dan Metode *Hidden Markov Model***".

Banyak sekali kesulitan dan hambatan yang penulis hadapi dalam penyusunan Laporan Tugas Akhir ini, akan tetapi berkat bantuan, bimbingan dan dorongan dari banyak pihak, akhirnya Laporan Tugas Akhir ini dapat penulis selesaikan sebagaimana mestinya. Untuk itu penulis mengucapkan terima kasih dan penghargaan yang setinggi-tingginya kepada banyak pihak yang telah memberikan bantuan baik pikiran maupun tenaga, waktu, sehingga Laporan Tugas Akhir ini dapat diselesaikan. Dalam kesempatan ini penulis mengucapkan banyak terima kasih kepada :

1. Ibu Rekyan Regasari MP, ST., MT. serta Bapak Budi Darma Setiawan, S.Kom., M.Cs. selaku dosen pembimbing I dan dosen pembimbing II yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan skripsi ini.
2. Kedua orangtua penulis, Ayahanda Mudjiran dan Ibunda Mistiani yang tidak pernah berhenti dalam memberikan doa, cinta, kasih sayang, serta dukungan kepada penulis selama ini.
3. Kakak penulis, Endah Novitiani yang selalu memberi dukungan dan motivasi agar selesainya skripsi ini.
4. Keluarga besar penulis, yang selalu memberikan dukungan dan doa kepada penulis.
5. Ir. Sutrisno, M.T, Ir. Heru Nurwasito, M.Kom., Himawat Aryadita, S.T, M.Sc., dan Edy Santoso, S.Si., M.Kom., selaku Ketua, Wakil Ketua 1, Wakil Ketua 2 dan Wakil Ketua 3 Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya.
6. Drs. Marji, M.T dan Issa Arwani, S.Kom, M.Sc selaku Ketua dan Sekretaris Program Studi Teknik Informatika/Ilmu Komputer Universitas Brawijaya.

7. Seluruh Dosen Teknik Informatika Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis.
8. Seluruh Civitas Akademika Teknik Informatika Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi di Teknik Informatika Universitas Brawijaya dan selama penyelesaian skripsi ini.
9. Mufida Nur Isnainia yang selalu memberikan saran, dukungan dan semangat kepada penulis.
10. Sahabat dan teman-teman dari TIF-C 09, Lab KCV serta seluruh angkatan TIF 2009 yang selalu bersama dalam keceriaan dan telah bersedia memberikan saran, bantuan, semangat, dan dukungan baik dalam bentuk material maupun non material demi terselesaikannya skripsi ini.
11. Seluruh pihak yang telah membantu kelancaran penulisan skripsi yang tidak dapat saya sebutkan satu persatu.

Penulis menyadari bahwa skripsi ini masih memiliki kekurangan dan masih belum sempurna, karena keterbatasan materi dan pengetahuan yang dimiliki penulis. Maka, saran dan kritik yang membangun dari semua pihak sangat diharapkan demi penyempurnaan selanjutnya. Semoga skripsi ini dapat bermanfaat dan berguna bagi semua pihak, baik penulis maupun pembaca, dan semoga Allah SWT meridhoi dan dicatat sebagai amalan ibadah, Amin.

Malang, 17 Oktober 2015

Penulis
gjulianto@gmail.com

ABSTRAK

Galih Julianto Purnomo. 2015. : Pengenalan Suara untuk Identifikasi Aksara Jawa Menggunakan Linear Predictive Coding dan Hidden Markov Model. Skripsi Program Studi Teknik Informatika/Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Brawijaya, Malang.

Dosen Pembimbing: Rekyan Regasari Mardi Putri, S.T., M.T. dan Budi Darma Setiawan, S.Kom, M.Cs.

Pengenalan suara merupakan salah satu cabang ilmu dalam teknologi informasi yang dapat digunakan untuk mengolah informasi dari sinyal digital agar suara dapat dikenali atau diidentifikasi. Tujuan dari penelitian ini untuk merancang dan mengimplementasikan teknologi pengenalan suara dalam identifikasi aksara jawa. Aksara jawa dipilih sebagai objek penelitian karena aksara jawa merupakan salah satu warisan budaya yang mulai dilupakan dan penelitian ini diharapkan dapat membantu kelestarian budaya tersebut. Metode yang dapat digunakan dalam pengenalan suara adalah *Linear Predictive Coding* (LPC) sebagai penganalisis sinyal dan *Hidden Markov Model* (HMM) yang digunakan sebagai pencocokan pola agar suara tersebut dapat dikenali. Hasil pengujian sistem menunjukkan bahwa sistem pengenalan suara untuk identifikasi aksara jawa membandingkan percobaan pengenalan sesuai dengan orde LPC (8, 10, 12, 14, dan 16). Hasil pengujian pada sistem menggunakan 80 data menunjukkan bahwa dengan penggunaan orde LPC yang bernilai 8 dapat menghasilkan akurasi tertinggi sebesar 55%. Sedangkan hasil akurasi sesuai orde LPC yang digunakan masing-masing 10, 12, 14, dan 16 yaitu 45%, 43.75%, 47.5%, dan 51.25%.

Kata kunci : Pengenalan suara, aksara jawa, *Linear Predictive Coding* (LPC), *Hidden Markov Model* (HMM), orde LPC

ABSTRACT

Galih Julianto Purnomo. 2015. : Voice Recognition to Identifiy Aksara Jawa using Linear Predictive Coding and Hidden Markov Model. Department of Computer Science and Information Technology, University of Brawijaya.
Advisors : Rekyan Regasari Mardi Putri, S.T., M.T. and Budi Darma Setiawan, S.Kom, M.Cs.

Voice recognition is one branch of science in information technology that can be used to process information from the digital signal so that voice can be recognized or identified. The purpose of this research is to design and implement speech recognition technology in the identification of Aksara Jawa. Aksara Jawa chosen as the research object because it is one of the cultural heritage is forgotten and the research is expected to help the preservation of the culture. The method can be used in speech recognition is Linear Predictive Coding (LPC) as a signal analyzer and Hidden Markov Model (HMM) is used as the voice pattern matching in order to be recognized. System test results showed that the voice recognition system for the identification of trial comparing Aksara Jawa recognition in accordance with the order of the LPC (8, 10, 12, 14, and 16). The test results on the system using the 80 data showed that the use of the LPC order worth 8 can generate the highest accuracy by 55%. While accuracy results in accordance order LPC used respectively 10, 12, 14, and 16, ie 45%, 43.75%, 47.5%, and 51.25%.

Keywords : *voice recognition, aksara Jawa, Linear Predictive Coding (LPC), Hidden Markov Model (HMM)*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SOURCECODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Aksara Jawa merupakan salah satu aksara budaya dan alat komunikasi di Indonesia, seperti kita kenal di masyarakat jawa contoh : jawa timur, jawa tengah, dan jawa barat. Namun masalah yang dihadapi sekarang adalah banyak masyarakat yang lupa akan warisan budaya yang tak ternilai bagi generasi berikutnya. Hal tersebut akan terasa manakala warisan tersebut diakui dan dicap sebagai milik bangsa lain, seperti kasus lagu rasa sayange dan reog ponorogo yang diakui sepihak oleh negara tetangga [DWI-10]. Aksara jawa merupakan peninggalan budaya yang tak ternilai harganya. Bentuk dan seni pembuatannya-pun patut untuk dipelajari dan dilestarikan tetapi hampir semua orang Jawa mengalami kesulitan membaca dan menulis huruf Jawa tersebut [ARI-05]. Hal tersebut menjadi kesulitan tersendiri dalam mempelajari dan melestarikan aksara Jawa itu sendiri.

Dalam teknologi informasi terdapat berbagai bidang ilmu untuk mengolah informasi, salah satunya adalah Pengolahan Sinyal Digital. Kemajuan teknologi dalam bidang Pengolahan Sinyal Digital (*Digital Signal Processing*) telah membawa dampak positif dalam kehidupan manusia. Disiplin ilmu dalam pengolahan sinyal digital yang memiliki dampak cukup besar adalah Pengenalan Suara. Pengenalan suara merupakan salah satu upaya agar suara dapat dikenali atau diidentifikasi sehingga dapat dimanfaatkan [TEG-11].

Penelitian tentang pengenalan suara telah banyak dilakukan dan dikembangkan hingga saat ini. Terdapat beberapa metode yang dapat diterapkan dalam pemrosesan sinyal suara, antara lain *Mel Frequency Cepstrum Coefisien* (MFCC), Transformasi *Wavelet*, dan *Linear Predictive Coding* (LPC). Sedangkan metode untuk pencocokan pola suara antara lain Jaringan Saraf Tiruan (JST), *Neuro-Fuzzy*, *Vector Quantization*, *Hidden Markov Model* (HMM), dan *Divide and Conquer*.

Linear Predictive Coding merupakan metode analisis sinyal suara yang menyatakan ciri-ciri penting dari sinyal suara tersebut dalam bentuk koefisien-koefisien LPC. *Linear Predictive Coding* merupakan salah satu teknik analisis sinyal percakapan yang paling *powerful* dan menyediakan ekstraksi fitur yang berkualitas baik dan efisien untuk digunakan dalam perhitungan. Sedangkan metode untuk pengenalan pola suara menggunakan *Hidden Markov Model* (HMM), dimana metode HMM ini suara dapat diasumsikan sebagai parameter acak dan dicari probabilitas yang maksimum sehingga suara tersebut dapat dikenali dalam pemodelan HMM [MUN-10]. HMM mempunyai kemampuan pemodelan dalam domain waktu yang lebih baik sehingga dapat menghasilkan keluaran yang tidak penuh derau dan penyisipan-penyisipan [ARD-03]. Penelitian sebelumnya yang berjudul '*Pengenalan Ucapan Kata Terisolasi dengan Metode Hidden Markov Model (HMM) melalui Ekstraksi Ciri Linear Predictive Coding (LPC)'*, menghasilkan akurasi yang tinggi untuk pengenalan ucapan kata yang berkorelasi tinggi pada pengujian dengan data rekaman maupun pengujian *on-line*.

Berdasarkan paparan informasi di atas, penulis mengambil judul skripsi Pengenalan Suara Untuk Identifikasi Aksara Jawa Menggunakan *Linear Predictive Coding* dan *Hidden Markov Model*.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka dapat dirumuskan permasalahan pada skripsi ini yaitu sebagai berikut:

1. Bagaimana mengimplementasikan algoritma *Linear Predictive Coding* dan *Hidden Markov Model* sebagai pengekstraksi fitur suara dan proses pencocokan fitur suara dalam aplikasi pengenalan suara yang dapat mengenali dan mengidentifikasi aksara Jawa.
2. Bagaimana analisis hasil akurasi dari pengenalan suara dengan metode *Linear Predictive Coding* (LPC) dan *Hidden Markov Model* (HMM).

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih terfokus, maka pada penelitian ini dibatasi oleh hal-hal berikut :

1. Objek yang akan diteliti dan dikenali dalam pengenalan suara ini adalah pengucapan huruf aksara jawa sebanyak 20 huruf : HA, NA, CA, RA, KA, DA, TA, SA, WA, LA, PA, DHA, JA, YA, NYA, MA, GA, BA, THA, NGA. Objek yang menjadi masukan hanya pengucapan suara per huruf aksara jawa.
2. File suara yang diproses adalah file berekstensi Waveform Audio (Wav) dengan sample rate 8000 Hz 8 bit mono.
3. Perekaman suara dilakukan dengan durasi waktu 1.5 detik.
4. Pengujian pengenalan suara dibatasi oleh beberapa faktor yaitu jarak bicara kurang lebih 1 cm didepan *microphone* eksternal dan jenis pembicara pria.

1.4 Tujuan

Tujuan yang ingin dicapai dalam pembuatan skripsi ini adalah:

1. Membangun aplikasi pengenalan suara untuk mengidentifikasi aksara jawa menggunakan metode *Linear Predictive Coding* (LPC) dan *Hidden Markov Model* (HMM).
2. Menganalisis hasil akurasi yang diperoleh dari pengenalan suara dengan metode *Linear Predictive Coding* (LPC) dan *Hidden Markov Model* (HMM).

1.5 Manfaat

Penulisan skripsi ini diharapkan mempunyai manfaat yang baik dan berguna bagi pembaca dan penulis. Adapun manfaat yang diharapkan adalah sebagai berikut:

1. Aplikasi yang dibangun dapat menjadi media untuk mempelajari huruf aksara jawa.
2. Aplikasi yang dibangun bisa digunakan sebagai dasar pengembangan aplikasi lebih lanjut pada ilmu yang berkaitan dengan aksara jawa dan sejenisnya.
3. Membantu para pengguna aplikasi dalam mengembangkan aplikasi pengenalan suara dengan menggunakan metode *Linear Predictive Coding* dan *Hidden Markov Model* atau metode algoritma yang lainnya.

1.6 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini sebagai berikut:

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan dan manfaat dari penelitian serta sistematika penulisan.

BAB II Tinjauan Pustaka

Bab ini berisi tentang teori-teori yang mendasari tugas akhir ini, meliputi pengenalan suara, pengolahan sinyal digital, metode pencocokan pola suara, dan pengenalan program yang digunakan.

BAB III Metode Penelitian dan Perancangan

Membahas metode yang digunakan dalam penelitian yang terdiri dari studi literatur, perancangan sistem perangkat lunak, implementasi sistem perangkat lunak, pengujian dan analisis, penulisan laporan, analisis kebutuhan dan perancangan perangkat lunak yang sesuai dengan teori yang ada.

BAB IV Implementasi

Membahas tentang implementasi dan sistem aplikasi yang sesuai dengan perancangan sistem yang telah dibuat.

BAB V Pengujian dan Analisis

Memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VI Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian program, serta saran-saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi tinjauan pustaka yang meliputi kajian pustaka dan dasar teori yang diperlukan untuk penelitian. Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Dasar teori membahas teori yang diperlukan untuk menyusun penelitian yang diusulkan.

Kajian pustaka pada penelitian ini adalah membahas penelitian sebelumnya yang berjudul '*Pengenalan Ucapan Kata Terisolasi dengan Metode Hidden Markov Model (HMM) melalui Ekstraksi Ciri Linear Predictive Coding (LPC)*'. Teori dasar yang akan dibahas pada bab ini adalah Aksara Jawa, Pengenalan Suara, Sinyal Suara Ucapan, *Linear Predictive Coding* (LPC), dan *Hidden Markov Model* (HMM).

2.1 Kajian Pustaka

Kajian pustaka pada penelitian ini adalah membahas penelitian sebelumnya yang berjudul '*Pengenalan Ucapan Kata Terisolasi dengan Metode Hidden Markov Model (HMM) melalui Ekstraksi Ciri Linear Predictive Coding (LPC)*'. Penelitian ini membahas mengenai pengenalan ucapan untuk menganalisis dan mengenali delapan ucapan kata Bahasa Indonesia yang memiliki korelasi tinggi antara kata yang satu dengan yang lainnya, seperti muka dan muak. Metode yang digunakan pada penelitian ini adalah *Linear Predictive Coding* (LPC) sebagai ekstraksi fitur dan *Hidden Markov Model* (HMM) sebagai pola pengenalannya.

Penelitian tersebut membuat program simulasi pengenalan kata yang terbagi menjadi 5 tahap. Tahap pertama yaitu akuisisi data kemudian data tersebut akan diekstraksi fitur sinyalnya menggunakan LPC. Tahap selanjutnya adalah proses pemodelan parameter HMM kemudian pengenalan kata itu sendiri. Penelitian tersebut menghasilkan akurasi yang tinggi untuk pengenalan ucapan kata yang berkorelasi tinggi pada pengujian dengan data rekaman maupun pengujian *on-line.*

Perbedaan yang dibuat penulis pada penelitian ini adalah pada penggunaan sistem pengenalan suara di khususkan untuk pengucapan aksara jawa. Perbedaan ini akan menyebabkan masukan yang dibutuhkan dan proses akhir sistem akan berbeda dengan penelitian sebelumnya. Hal tersebut dikarenakan aksara jawa hanya membutuhkan satu suku kata dalam pengucapannya.

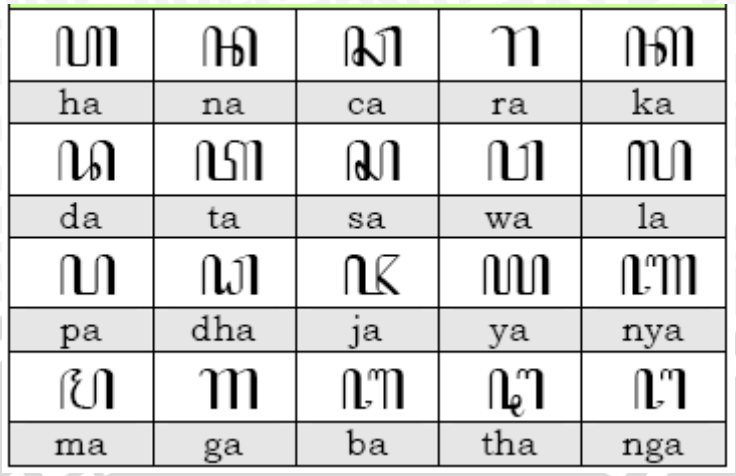
2.2 Aksara Jawa

Aksara Jawa merupakan salah satu aksara budaya dan alat komunikasi di Indonesia, seperti kita kenal di masyarakat jawa contoh : jawa timur, jawa tengah, jawa barat, dll. Bahkan bahasa dan aksara jawa digunakan masyarakat Suriname, Afrika Selatan sebagai alat komunikasi sehari-hari. Namun masalah yang dihadapi sekarang adalah banyak masyarakat yang lupa akan warisan budaya yang tak ternilai bagi generasi berikutnya. Hal tersebut akan terasa manakala warisan tersebut diakui dan dicap sebagai milik bangsa lain, seperti kasus lagu rasa sayange dan reog ponorogo yang diakui sepihak oleh negara tetangga.

Aksara jawa atau tulisan jawa lebih sering dikaitkan dengan legenda Aji Saka, dialah orang yang dianggap penggagasnya. 20 aksara baku Jawa tersebut dikaitkan dengan dua orang pengiring Aji Saka yang tinggal di pulau Majeti. Mereka diberi tanggungjawab menjaga keris pusaka dan sejumlah barang perhiasan. Di sini timbul suatu masalah sehingga kedua-duanya bertikai [DWI-10].

Huruf Aksara Jawa inti / Aksara Carakan merupakan aksara inti yang terdiri dari 20 suku kata atau biasa disebut *Dentawiyanjana*. Carakan (abjad Jawa) yang di gunakan di dalam ejaan bahasa Jawa pada dasarnya terdiri aras dua puluh aksara pokok yang bersifat silabik (bersifat kesukukataan). Masing – masing aksara pokok mempunyai aksara pasangan, yakni aksara yang berfungsi untuk menghubungkan suku kata tertutup konsonan dengan suku kata berikutnya, kecuali suku kata yang tertutup *wignyan, layar,* dan *cecak* [DWI-10].

Untuk selengkapnya dapat dilihat dari Gambar 2.1

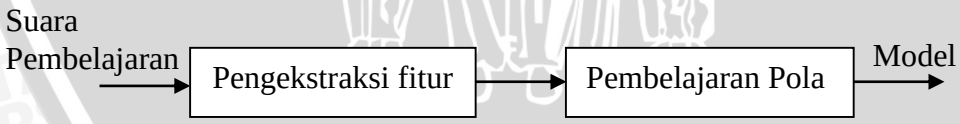


Gambar 2.1 Aksara Carakan

Sumber : [WIK-14]

2.3 Pengenalan Suara

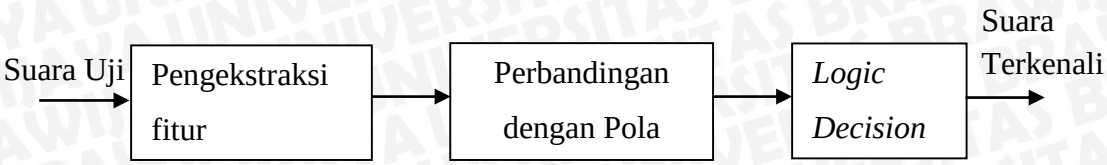
Pengenalan suara merupakan salah satu upaya agar suara dapat dikenali atau diidentifikasi sehingga dapat dimanfaatkan. Pengenalan suara dapat dibedakan ke dalam tiga bentuk pendekatan, yaitu pendekatan akustik-fonetik, pendekatan kecerdasan buatan, dan pendekatan pengenalan pola. Pendekatan yang akan digunakan dalam penelitian ini merupakan pendekatan pengenalan pola. Blok diagram pengenalan pola pada pengenalan suara ditunjukkan pada Gambar 2.2.



(a). Blok diagram pembelajaran pola.

Gambar 2.2 Blok diagram pembelajaran pola

Sumber : [NUR-12]



(b). Blok diagram pengenalan suara

Gambar 2.3 Blok diagram pembelajaran pola dan pengenalan suara

Sumber : [NUR-12]

Berikut ini merupakan penjelasan dari masing-masing blok :

1. Pengekstraksi fitur.

   Merupakan proses mendapatkan sederetan besaran pada bagian sinyal masukan untuk menetapkan pola pembelajaran atau pola uji.

2. Pembelajaran pola

   Satu atau lebih pola pembelajaran yang berhubungan dengan bunyi ucapan dari kelas yang sama, digunakan untuk membuat pola *representative* dari ciri-ciri kelas tersebut. Hasilnya yang biasa disebut dengan pola referensi, dapat menjadi sebuah model yang mempunyai karakteristik bentuk statistik dari ciri-ciri pola referensi.

3. Perbandingan dengan Pola Model

   Pola uji yang akan dikenali dibandingkan dengan setiap kelas pola referensi. Kesamaan besaran antara pola uji dengan setiap pola referensi akan dihitung.

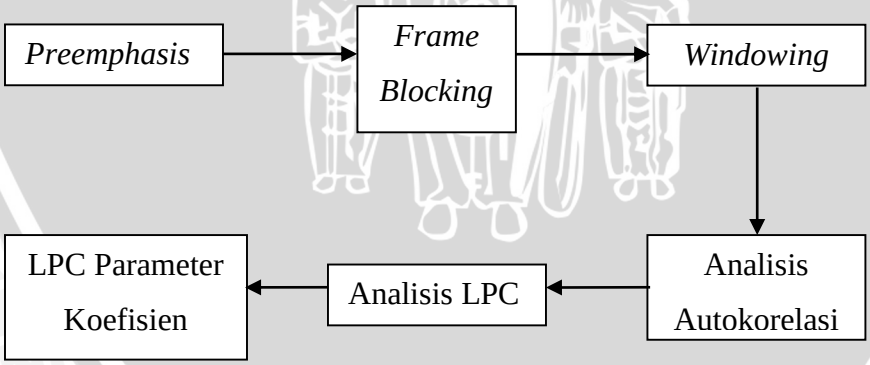
4. *Logic Decision*

   Menentukan kelas pola referensi mana yang paling cocok untuk data uji berdasarkan klasifikasi pola.

2.4 *Linear Predictive Coding* (LPC)

Ekstraksi fitur merupakan kegiatan melakukan ekstraksi terhadap informasi yang terdapat di suatu sinyal. *Linear Predictive Coding* (LPC) merupakan salah satu teknik analisis sinyal percakapan yang paling *powerful* dan menyediakan ekstraksi fitur yang berkualitas baik dan efisien untuk digunakan dalam perhitungan.

LPC dimulai dengan asumsi bahwa sinyal suara diproduksi oleh dengungan pada akhir sebuah pita suara. *The glotis* (ruang antara pita suara) menghasilkan dengungan, yang dicirikan oleh intensitas (kerasnya suara) dan frekuensi (*pitch*). Saluran vokal (tenggorokan dan mulut) bentuk pipa, yang dicirikan oleh resonansi, yang disebut *formant*. LPC melakukan analisis dengan cara memperkirakan *formant*, memisahkan *formant* dari sinyal, yang dinamakan proses *inverse filtering*, lalu mengestimasi intensitas dan frekuensi dari sinyal percakapan yang tersisa, yang disebut *residue*. Karena sinyal percakapan bervariasi seiring waktu, estimasi tersebut dilakukan untuk setiap potongan kecil dari sinyal, yang dinamakan *frame* [OKT-]. Prosedur untuk mendapatkan koefisien LPC diperlihatkan pada gambar dibawah.



Gambar 2.4 Blok diagram LPC

Sumber : [MUN-10]

Gambar diatas menunjukkan blok diagram LPC yang biasa digunakan dalam sistem pengenalan suara manusia.

Langkah-langkah dasar yang harus dilakukan mengenai blok diagram tersebut adalah :

1. *Preemphasis* : Proses dimana sinyal ucapan diubah menjadi sinyal digital.
2. *Frame Blocking* : Pada tahap ini sinyal yang telah di-*preemphasis*, diblok menjadi beberapa *frame*. Masing-masing *frame* berisi sampel sejumlah N, dan jarak antar *frame* dipisahkan sejumlah M sampel.
3. *Windowing* : Pada tahap ini dilakukan proses penjendelaan pada setiap bagian sinyal yang telah dibuat sebelumnya. Hal ini dilakukan untuk meminimalkan diskontinuitas pada bagian awal dan akhir sinyal.
4. Analisis Autokorelasi : Tiap bagian yang telah diberi *window* kemudian akan dibentuk autokorelasinya.
5. Analisis LPC : Langkah berikutnya adalah analisis LPC dimana semua nilai autokorelasinya yang telah dihitung pada tahap sebelumnya akan diubah menjadi koefisien LPC.

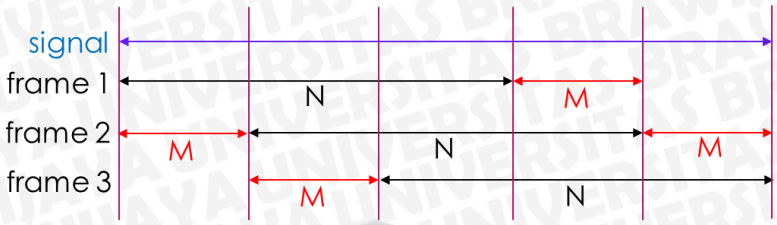
   Langkah-langkah analisis LPC untuk pengenalan suara adalah sebagai berikut :

1. *Preemphasis* terhadap cuplikan sinyal dengan persamaan *preemphasizer*:

$$s(n) = s(n) - a * s(n-1) \qquad (2.1)$$

   Dengan *s*(n) adalah sampel ke-n dan nilai *a* yang paling sering digunakan ialah 0.9375.

2. Membagi hasil *preemphasis s*(n) ke dalam *frame-frame* yang masing-masing memuat *N* buah sampel yang dipisahkan sejauh *M* buah sampel. Semakin *M* < *N* semakin baik perkiraan *spectral* LPC dari *frame* ke *frame*.



Gambar 2.5 Ilustrasi *frame blocking*

3. Melakukan *windowing* terhadap setiap *frame* yang telah dibentuk untuk meminimalkan diskontinuitas pada ujung awal dan ujung akhir setiap *frame* dengan persamaan *Hamming Window* untuk sampel ke-n adalah :

$$W(n) = 0.54 - 0.46\cos(2\pi n/N - 1)\,, 0 \le n \le N - 1 \qquad (2.2)$$

Kemudian untuk mendapatkan nilai sampel yang baru yaitu dengan mengalikan x(n) dengan *W*(n). x(n) merupakan sampel ke-n sedangkan *W*(n) merupakan hasil *Hamming Window* ke-n.

$$x(n) = x(n)W(n),\ \ 0 \le n \le N - 1 \qquad (2.3)$$

4. Analisis autokorelasi terhadap setiap *frame* hasil *windowing* x_1(n) dengan persamaan (2.4)

$$r_1(m) = \sum_{n=0}^{N-1-m} x_1(n)x_1(n+m) \qquad (2.4)$$

dengan N merupakan jumlah sampel per *frame* sedangkan *m* dimulai dari 0 dan nilai tertinggi dari *m* = *p*, dan *p* adalah orde LPC yang biasa bernilai 8 – 16.

5. Langkah selanjutnya dalam metode LPC yaitu analisis LPC, yaitu mengonversi *p*+1 buah hasil autokorelasi pada masing-masing *frame* menjadi koefisien LPC. $a_m = a_m^{(p)}$ untuk *m* = 1,2,…, *p*. Metode yang umum digunakan untuk langkah ini dikenal dengan sebutan metode Durbin. Metode ini dapat ditunjukkan dengan persamaan (2.5) - (2.9)

$$E^{(0)} = r(0) \qquad (2.5)$$

$$k_m = \frac{r(m) - \sum_{j=1}^{m-1} \alpha_j^{(m-1)} r(|m-j|)}{E^{(m-1)}}, \qquad 1 \le m \le p \qquad (2.6)$$

$$\alpha_m{}^{(m)} = k_m \qquad (2.7)$$

$$\alpha_j^{(m)} = \alpha_j^{(m-1)} - k_m \alpha_{m-j}^{(m-1)}, \quad (2.8)$$

$$E^{(m)} = \left(1 - {k_m}^2\right) E^{(m-1)} \quad (2.9)$$

$r(0)$ adalah hasil autokorelasi[0], $E^{(m)}$ adalah *error*, k_m adalah koefisien pantulan, $\alpha_j^{(m)}$ adalah koefisien prediksi untuk $1 \leq j \leq m$. [MUN-10].

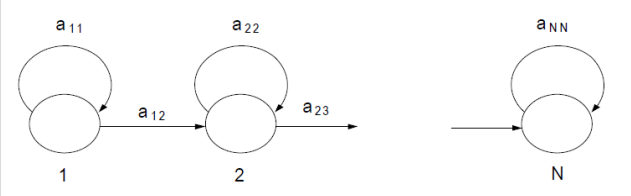
Dimana pada persamaan (2.5-2.9) dapat diselesaikan secara rekursif untuk masing-masing *frame* dengan $m = 1,2,\ldots,p$, dan yang akan dihasilkan adalah

$$a_m = koefisien\ LPC = \alpha_m^{(p)}, \quad 1 \leq m \leq p \quad \text{[RAB-93]}$$

Setiap *frame* akan menghasilkan koefisien LPC sejumlah *p*, maka setiap file WAV akan menghasilkan koefisien LPC sebanyak total *frame* dikali dengan *p* (ordeLPC) yang digunakan.

2.5 *Hidden Markov Model* (HMM)

Hidden markov model merupakan pemodelan probabilitas suatu sistem dengan mencari parameter-parameter markov yang tidak diketahui untuk memperoleh analisis sistem tersebut. Metode *hidden markov model* (HMM) mampu menangani perubahan statistik dengan memodelkan elemen-elemen menggunakan probabilitas. HMM memiliki tiga parameter utama yang harus dicari nilainya terlebih dahulu, ketiga parameter tersebut sebagai berikut :



Gambar 2.6 Contoh Model HMM kiri-kanan

Sumber : [ALI-13]

Untuk mengkarakteristikkan HMM dengan *N* cacah keadaan, perlu ditentukan terlebih dahulu parameter-parameter berikut ini.

a. Probabilitas transisi keadaan, a_{ij}

$a_{ij} = 0$ untuk $i < j$ dan $i > j + 1$, untuk $1 \leq i, j \leq N$ (2.10)

Parameter A dalam HMM dinyatakan dalam sebuah matriks dengan ukuran *M x M* dengan *M* adalah jumlah state yang ada.

b. Probabilitas keadaan awal, π_i

$\pi_i = 1$ untuk i = 1, dan $\pi_i = 0$ untuk $i \neq 1$ (2.11)

Parameter *π,* disebut sebagai parameter awal, merupakan probabilitas kemunculan suatu *state* di awal. Sama halnya dengan parameter *B*, parameter *π* juga dituliskan dalam bentuk matriks kolom dengan ukuran *M x 1*, dimana *M* adalah jumlah *state* nya

c. Parameter *B* disebut sebagai probabilitas *state*, merupakan proses kemunculan suatu *state* dalam deretan seluruh *state* yang ada. Parameter *B* dalam HMM dituliskan dalam bentuk matriks kolom dengan ukuran *M x 1*, dimana *M* merupakan jumlah seluruh *state* yang ada.

Fungsi rapat observasi keluaran kontinyu berupa fungsi *Gaussian* untuk setiap keadaan, yaitu:

$b_i = N(o_t , \mu_i, \sum_i)$ (2.12)

dengan *N* adalah distribusi normal, *o* adalah vektor observasi keluaran, *μi* adalah mean keadaan ke-*i*, dan ∑*i* adalah kovarians pada keadaan ke-*i*.

Setiap model HMM dari parameter-parameter yang diberikan di atas dapat dinyatakan sebagai $\lambda = (A,\pi,\mu, \sum)$ dengan A adalah matriks probabilitas transisi keadaan, *a*ij, yang berdimensi *NxN*, adalah vektor probabilitas transisi keadaan, π_i, dengan dimensi 1x*N*, *μ* adalah rerata pada setiap keadaan yang membentuk matriks *Nx1*, dan ∑ merupakan kovarians pada setiap keadaan yang membentuk matriks berdimensi 1x*N* [ALI-13].

Pelatihan model, diberikan sekumpulan sequence {*Oi*}, biasanya dihitung dengan menggunakan Baum-Welch re-estimation, dapat untuk menentukan parameter - parameter $(A,\pi,\mu, \sum)$ yang memaksimalkan probabilitas $P(\{Oi|\lambda\})$. Prosedur pelatihan dihentikan setelah konvergen dari *likelihood*. Langkah evaluasi yaitu menghitung probabilitas $P(O|\lambda)$, diberikan sebuah model λ dan sebuah

deretan *O* untuk dievaluasi, menggunakan algoritma *forward* [BIC-03]. Algoritma forward dan backward digunakan untuk menghitung probabilitas dari urutan nilai observasi yang diberikan oleh HMM dan untuk evaluasi HMM. Algoritma Baum-Welch digunakan untuk melatih parameter HMM jika diberikan dataset barisan-barisan tertentu agar dapat menemukan himpunan transisi *state* yang paling mungkin beserta probabilitas *output*-nya. Berikut ini penjelasan mengenai algoritma yang digunakan dalam HMM [MUL-08]:

a. Persoalan dalam evaluasi dapat diselesaikan dengan menggunakan algoritma yang dinamakan prosedur maju-mundur (*forward-backward* prosedur). Pertama, dijelaskan prosedur *forward*, diasumsikan variabel maju *αt(i)* didefinisikan sebagai :

$$\propto t(i) = P(O1, O2, \dots, Ot, qt = Si \mid \lambda \quad (2.13)$$

Yaitu kemungkinan rangkaian pengamatan parsial hingga waktu t dan berada pada state *Si* pada waktu *t*, jika diberikan model λ, maka *αt(i)* dapat dihitung dengan induksi sebagai berikut :

1. Inisialisasi

$$\propto t(i) = \pi ibi(O1), 1 \leq i \leq N \quad (2.14)$$

2. Induksi

$$\propto_{t+1} (j) = [\sum_{i=1}^{N} \propto_t (i) a_{ij}] b_j(O_{t+1}) \;\; 1 \leq t \leq T-1, \;\; 1 \leq j \leq N \quad (2.15)$$

3. Terminasi

$$P(O|\lambda) = \sum_{i=1}^{N} \propto_T (i) \quad (2.16)$$

Langkah pertama merupakan inisialisasi, langkah induksi merupakan langkah yang paling utama pada prosedur *forward*. *P(O|λ)* dapat dicari dengan menjumlahkan variabel maju dengan *t=T* dari semua state. *P(O|λ)* merupakan probabilitas model menghasilkan rangkaian pengamatan $O = O1, O2, \dots, OT$.

b. Dengan cara yang sejenis dapat didefinisikan variabel mundur *βt(i)* sebagai berikut :

$$\beta t(i) = P(Ot + 1, Ot + 2, \dots, OT | qt = Si, \lambda) \quad (2.17)$$

Yaitu kemungkinan rangkaian pengamatan dari *t+1* hingga *T* jika diberikan state *Si* pada waktu t dan model *λ*. *βt(i)* dapat diselesaikan sebagaimana *αt(i)*.

1. Inisialisasi

$$\beta T(i) = 1, 1 \leq i \leq N \quad (2.28)$$

2. Induksi

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \; t = T-1, T-2, \ldots, 1$$

$$1 \leq i \leq N \quad (2.19)$$

Variabel mundur akan digunakan pada proses estimasi nilai parameter-parameter HMM.

c. Persoalan evaluasi merupakan persoalan yang paling sulit , bisa dikatakan sebagai persoalan pelatihan yang digunakan untuk menghasilkan parameter model *A, B,* dan *π* optimal sehingga dapat dengan baik merepresentasikan rangkaian observasi yang terjadi. Kriteria optimal adalah memaksimalkan probabilitas rangkaian pengamatan, *P(O/λ)*, dengan diberikan model, *λ(A,B,π)*. Tidak ada pendekatan analitik untuk permasalah ini, namun terdapat prosedur iteratif seperti metode Baum-Welch yang dapat digunakan. Untuk mendeskripsikan formula pelatihan secara matematis, diasumsikan *ξt(i,j)* sebagai probabilitas berada pada *state i* pada waktu *t*, dan *state j* pada waktu *t+1*, diberikan model rangkaian pengamatan :

$$\xi t(i,j) = P(qt = Si, qt + 1 = Sj | O, \lambda) \quad (2.20)$$

Dengan menggunakan definisi variabel maju dan mundur, persamaan diatas dapat ditulis dalam bentuk :

$$\xi_t(i,j) = \frac{(\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j))}{(P(O|\lambda))} \quad (2.21)$$

$$\xi_t(i,j) = \frac{(\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j))}{(\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j))} \quad (2.22)$$

P(O/λ) merupakan nilai probabilitas model *λ* memberikan sequence *O*. Biasanya *P(O/λ)* sering diberi nilai 1, nilai harapan model *λ* memberikan sequence *O*. Menjumlahkan *ξt(i,j)* pada $1 \leq t \leq T-1$ menghasilkan jumlah

perpindahan dari *i* ke *j* yang diharapkan. Untuk kebutuhan pelatihan, didefinisikan probabilitas berada di *state i* pada waktu *t*, diberikan rangkaian pengamatan dan model sebagai *γt(i)*.

$$\gamma_t(i) = \frac{(\propto_t(i)\beta_t(i))}{(P(O|\lambda))} = \frac{(\propto_t(i)\beta_t(i))}{(\sum_{i=1}^{N} \propto_t(i)\beta_t(i))} \quad (2.23)$$

Selanjutnya, *γt(i)* dan *ξt(i,j)* dapat dihubungkan dengan menjumlahkan *ξt(i,j)* untuk semua j :

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \quad (2.24)$$

Menjumlahkan, *γt(i)* sepanjang waktu memberikan jumlah state i dikunjungi. Jika waktu *T* tidak dimasukkan, dengan kata lain menjumlahkan sepanjang *1 ≤ t ≤ T-1*, ini memberikan jumlah perpindahan dari state i.

$$\sum_{t=1}^{T-1} \gamma_t(i) = jumlah\ perpindahan\ yang\ diharapkan\ Si \quad (2.25)$$

$$\sum_{j=1}^{N} \xi_t(i,j) = jumlah\ perpindahan\ yang\ diharapkan\ dari\ Si\ ke\ Sj \quad (2.26)$$

Menggunakan formula di atas, dapat didefinisikan formula untuk melakukan estimasi terhadap nilai-nilai parameter HMM

$$\overline{a_{ij}} = \frac{(jumlah\ perpindahan\ yang\ diharapkan\ dari\ Si\ ke\ Sj)}{(jumlah\ perpindahan\ yang\ diharapkan\ dari\ Si)}$$

$$\overline{a_{ij}} = \frac{\left(\sum_{j=1}^{N} \xi_t(i,j)\right)}{\left(\sum_{t=1}^{T-1} \gamma_t(i)\right)} \quad (2.27)$$

$$\overline{b_j}(k) = \frac{(jumlah\ frekuensi\ yang\ diharapkan\ pada\ state\ j\ dan\ menghasilkan\ simbol\ Vk)}{(jumlah\ frekuensi\ pada\ state\ j)}$$

$$\overline{b_j}(k) = \frac{\left(\sum_{t=1}^{T-1} \gamma_t(i)\right)}{\frac{s.t.Ot=vk}{\left(\sum_{t=1}^{T-1} \gamma_t(i)\right)}} \quad (2.28)$$

BAB III
METODOLOGI PENELITIAN DAN PERANCANGAN

3.1 Metodologi Penelitian

Subbab ini berisi langkah – langkah yang dilakukan dalam penelitian ini yaitu studi literatur, analisis sistem, perancangan perangkat lunak, implementasi perangkat lunak dan pengujian perangkat lunak. Gambar 3.1 menunjukan diagram blok metode penelitian.

Gambar 3.1 Diagram Blok Metode Penelitian

3.1.1 Studi Literatur

Studi literatur dilakukan untuk mempelajari teori yang digunakan dalam pembuatan perangkat lunak pengenalan suara untuk identifikasi Aksara Jawa. Teori – teori yang dipelajari yaitu tentang :

1. Aksara Jawa
2. Pengenalan Suara
3. Proses Pengolahan Dan Pencocokan Suara
4. *Linear Predictive Coding*
5. *Hidden Markov Model*

3.1.2 Analisis dan Perancangan Perangkat Lunak

Perancangan yang dilakukan meliputi analisis kebutuhan, perancangan perangkat lunak, dan contoh perhitungan manual sistem yang akan dibangun. Tahap analisis kebutuhan terdiri dari dua bagian yaitu analisis kebutuhan data, dan analisis kebutuhan fungsional. Proses perancangan perangkat lunak mempunyai lima tahap, yaitu perancangan diagram blok, perancangan proses sistem, perancangan database, perancangan algoritma dan perancangan *user interface*. Perancangan algoritma sendiri terdiri dari perancangan ekstraksi fitur LPC dan perancangan *Hidden Markov Model*. Perancangan ekstraksi fitur LPC mempunyai lima tahap yaitu perancangan *preemphasis*, *block into frame*, *windowing*, *autokorelasi*, dan analisis LPC. Sedangkan perancangan HMM yaitu perancangan parameter HMM awal, algoritma *forward*, algoritma *backward*, matriks *Gamma*, matriks *Epsilon*, evaluasi, re-estimasi parameter HMM, dan pengenalan aksara.

3.1.2.1 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan dalam pembuatan perangkat lunak. Analisis kebutuhan dilakukan dengan mengidentifikasi kebutuhan sistem dan siapa saja yang terlibat di dalamnya. Berikut analisis kebutuhan dalam aplikasi Pengenalan Suara Identifikasi Aksara Jawa.

a) Kebutuhan Data

Data yang diolah oleh perangkat lunak ini adalah:

1) Data suara yang menjadi *input* dari sistem.
2) Data huruf aksara jawa yang akan menjadi *output* sistem.
3) Data hasil training yang akan di simpan di dalam database.

b) Kebutuhan Fungsional

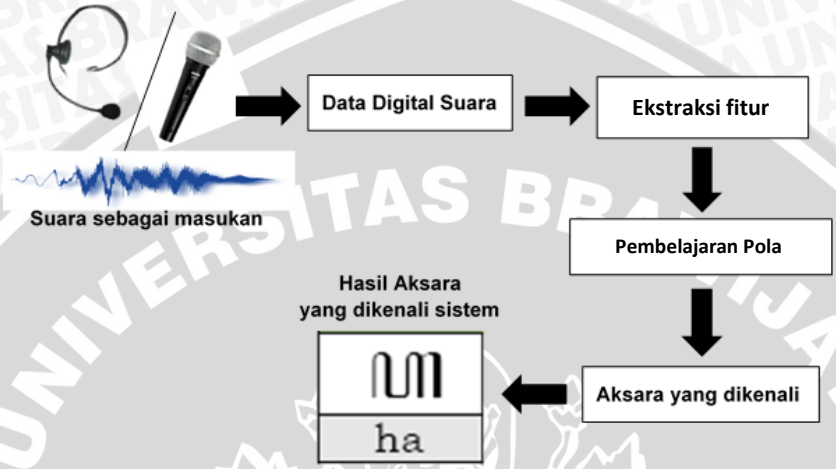
Fungsi-fungsi yang dimiliki oleh perangkat lunak ini adalah:

1) Program harus mampu menerima *input*-an berupa suara / file .wav.
2) Program harus mampu mengekstraksi fitur dari suara/file yang telah di-*input*-kan.
3) Perangkat lunak harus mampu melakukan training terhadap data training yang disiapkan.
4) Perangkat lunak harus mampu mengidentifikasi suara berdasarkan pola yang ada.
5) Perangkat lunak harus mampu menghasilkan aksara jawa berdasarkan suara yang telah dikenali.

3.1.2.2 Perancangan Perangkat Lunak

a) Arsitektur Sistem

Perancangan aplikasi pengenalan suara untuk identifikasi aksara jawa dapat dilihat lebih jelas pada gambar arsitektur di bawah :



Gambar 3.2 Arsitektur Sistem

Fungsi masing-masing bagian dalam rancangan sistem di atas adalah sebagai berikut :

1. Suara/file .wav sebagai sinyal *input* yang digunakan system.
2. *Microphone* sebagai media *recording*, yang dihubungkan langsung dengan komputer.
3. Sinyal suara hasil perekaman diubah dari bentuk analog ke digital menggunakan bantuan soundcard dalam *accessories windows* kemudian disimpan dalam bentuk *waveform (.wav).*
4. Pengolahan suara dilakukan untuk mendapatkan koefisien LPC dan ciri khas sinyal fitur suara menggunakan *Linear Predictive Coding.*
5. Setelah diolah, suara akan diidentifikasi oleh sistem sebagai pengucapan aksara dimana cara pengenalan pengucapan tersebut menggunakan *Hidden Markov Model.*
6. Aksara yang telah dikenali melalui pengidentifikasian akan ditampilkan di dalam aplikasi.

3.1.3 Implementasi

Setelah melakukan perancangan maka implementasi aplikasi pengenalan suara dilakukan dengan mengacu pada perancangan perangkat lunak. Aplikasi pengenalan suara ini akan diimplementasikan pada *desktop* menggunakan bahasa pemrograman C# dan database MySQL.

3.1.4 Pengujian Perangkat Lunak

Setelah tahapan implementasi selesai, maka tahapan penelitian ini dilanjutkan dengan melakukan pengujian terhadap sistem yang telah dibuat untuk melihat hasil dari implementasi algoritma. Pengujian yang dilakukan pada penelitian ini dilakukan untuk menunjukkan apakah perangkat lunak yang dibuat dapat bekerja sesuai dengan perancangan. Analisis dilakukan untuk melihat apakah metode yang digunakan dalam penelitian ini menghasilkan hasil yang baik. Hasilnya dapat dikatakan baik jika dapat mengidentifikasi aksara jawa dengan benar.

Pengujian melibatkan 20 aksara yang akan diteliti yaitu Ha, Na, Ca, Ra, Ka, Da, Ta, Sa, Wa, La, Pa, Dha, Ja, Ya, Nya, Ma, Ga, Ba, Tha, dan Nga. Pengujian pada penelitian ini dilakukan menggunakan 4 skenario yaitu pengujian terhadap fungsionalitas metode yang digunakan, pengujian terhadap jumlah kelas yang digunakan, pengujian terhadap pengaruh sinyal yang digunakan dalam setiap pengujian dan pengaruh orde LPC yang digunakan terhadap akurasi proses pengenalan suara dengan menggunakan orde LPC yang berbeda dalam setiap pengujian yaitu 8, 10, 12, 14, dan 16.

Tabel 3.1 Tabel skenario pengujian fungsional

No	Aksara	Benar/Salah
1		
…		
n		
Total		

Tabel 3.2 Tabel skenario pengujian terhadap jumlah kelas

No	N-kelas	
	Label	Benar/Salah
1		
…		
n		
Total		

Tabel 3.3 Tabel skenario pengujian terhadap panjang sinyal

No	Panjang sinyal	
	Label	Benar/Salah
1		
…		
80		
Total		

Tabel 3.4 Tabel skenario pengujian terhadap orde LPC

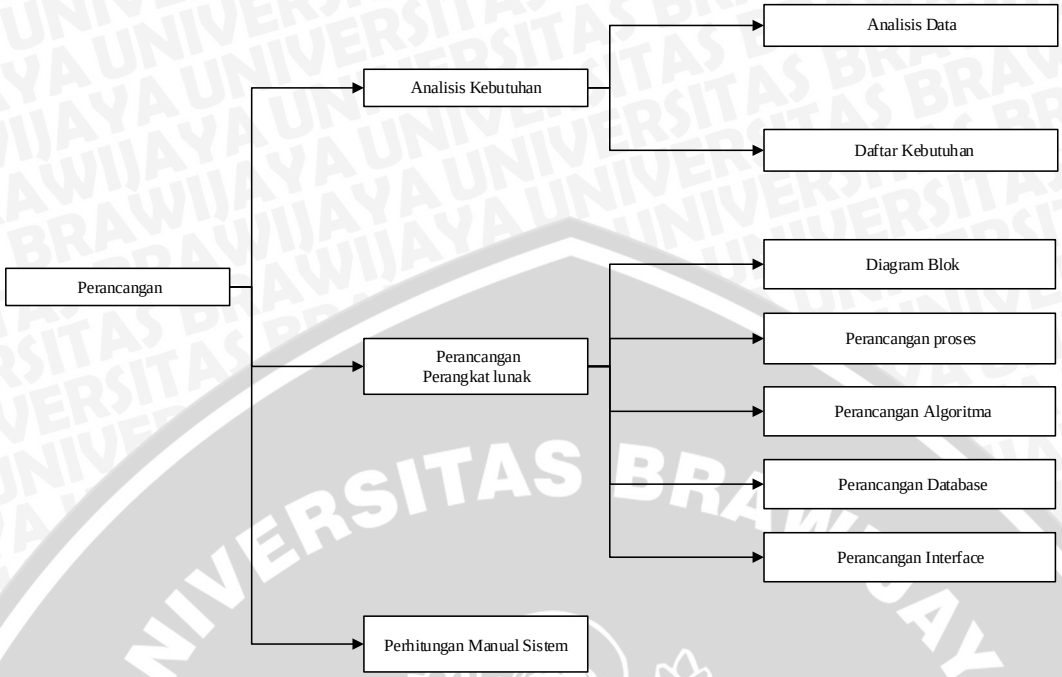
No	Orde LPC	
	Label	Benar/Salah
1		
…		
80		
Total		

3.1.5 Penulisan Laporan

Laporan penelitian ditulis setelah semua tahapan analisis kebutuhan, perancangan, implementasi dan pengujian sistem aplikasi telah selesai dilakukan. Laporan berisi dokumentasi perancangan aplikasi yang akan berguna untuk pengembangan aplikasi selanjutnya. Laporan juga berisi kesimpulan yang diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

3.2 Perancangan

Subbab ini membahas mengenai analisa dan perancangan aplikasi Pengenalan Suara untuk Identifikasi Aksara Jawa. Perancangan yang dilakukan meliputi analisa kebutuhan, perancangan perangkat lunak, dan contoh perhitungan manual sistem yang akan dibangun. Tahap analisis kebutuhan terdiri dari tiga langkah yaitu analisa kebutuhan data, dan analisa kebutuhan fungsional. Proses perancangan perangkat lunak mempunyai lima tahap, yaitu perancangan diagram blok, perancangan proses sistem, perancangan database, perancangan algoritma, dan perancangan *interface*. Tahap-tahap perancangan yang dilakukan seperti yang digambarkan pada Gambar 3.4 berikut ini.



Gambar 3.3 Pohon Perancangan

3.2.1 Analisis Kebutuhan

Pengembangan sebuah perangkat lunak bertujuan untuk menghasilkan perangkat lunak yang dapat memenuhi kebutuhan user. Setiap pengembangan sebuah sistem perangkat lunak memerlukan adanya dokumentasi terhadap kebutuhan-kebutuhan user agar tujuan tersebut tercapai.

Proses analisis kebutuhan dilakukan dengan acuan pengumpulan dan penetapan kebutuhan-kebutuhan dari aplikasi yang akan dibangun. Analisis kebutuhan dibagi menjadi dua tahap yaitu tahap analysis data dan tahap penjabaran daftar kebutuhan.

3.2.1.1 Analisis Data

Tahap analisis data dilakukan untuk mengidentifikasi kebutuhan data yang dibutuhkan aplikasi maupun yang akan dihasilkan oleh aplikasi.

Tabel 3.5 Analisis data

Analisis Data	Deskripsi Data
Suara / file .wav	Suara / file.wav merupakan data yang digunakan sebagai inputan dari aplikasi. Suara /file.wav yang menjadi inputan merupakan hasil rekaman pengguna saat mengucapkan aksara jawa.
Huruf aksara jawa	Huruf aksara jawa merupakan data yang akan dihasilkan aplikasi setelah melakukan pencocokan pola pada data input.
Data training	Data training merupakan data hasil training yang disimpan dalam *database* sebagai pola yang menjadi dasar pencocokan dengan data input.

3.2.1.2 Daftar Kebutuhan

Pada tahap daftar kebutuhan akan dijelaskan kebutuhan fungsional yang diperlukan oleh pengguna atau user dalam menggunakan aplikasi pengenalan aksara jawa. Spesifikasi kebutuhan fungsional pengguna atau user ditunjukkan pada tabel 3.3

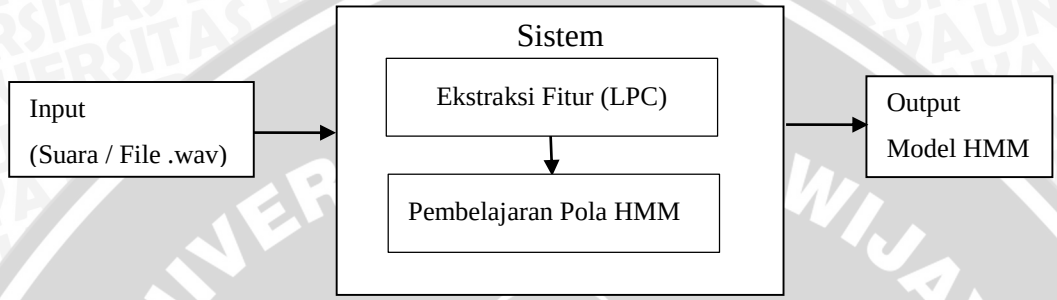
Tabel 3.6 Daftar kebutuhan fungsional

No	Kebutuhan
01	Program harus mampu menerima *input*-an berupa suara / file .wav.
02	Program harus mampu mengekstraksi fitur dari suara / file .wav yang telah di-*input*-kan.
03	Perangkat lunak harus mampu melakukan training terhadap data training yang disiapkan.
04	Perangkat lunak harus mampu mengidentifikasi suara berdasarkan pola yang ada.
05	Perangkat lunak harus mampu menghasilkan aksara jawa berdasarkan suara yang telah dikenali.

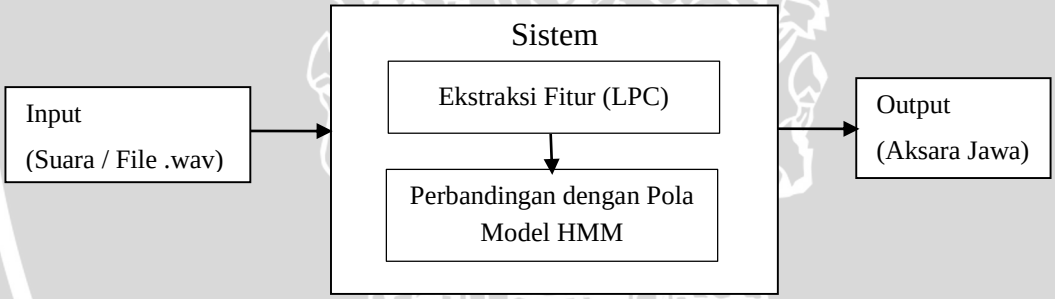
3.2.2 Perancangan Perangkat Lunak

Perancangan perangkat lunak terdiri dari lima tahap, antara lain perancangan diagram blok, perancangan proses, perancangan database, perancangan algoritma, dan perancangan *user interface*.

3.2.2.1 Diagram Blok



(a). Blok diagram pembelajaran pola.



(b). Blok diagram pencocokan pola.

Gambar 3.4 Diagram blok system

Berikut ini merupakan penjelasan dari masing-masing blok :

1. Input

   Input ke dalam sistem adalah memasukkan suara atau file rekaman. File yang di*input*kan berupa file wav hasil rekaman pengucapan aksara jawa.

2. Sistem : Ekstraksi Fitur

   Data rekam yang dimasukkan dalam sistem akan diproses menggunakan *Linear Predictive Coding* sebagai pengekstraksi fitur untuk menetapkan pola pembelajaran atau pola uji.

3. Sistem : Pembelajaran Pola

   Fitur-fitur yang dihasilkan pada proses ekstraksi akan digunakan untuk membuat pola referensi menggunakan *Hidden Markov Model* sehingga dapat menjadi sebuah model dari masing-masing aksara.

4. Sistem : Perbandingan dengan Pola

   Pola uji yang akan dikenali dibandingkan dengan setiap kelas pola referensi menggunakan *Hidden Markov Model*. Menentukan kelas pola referensi mana yang paling cocok untuk pola uji yang diinputkan.

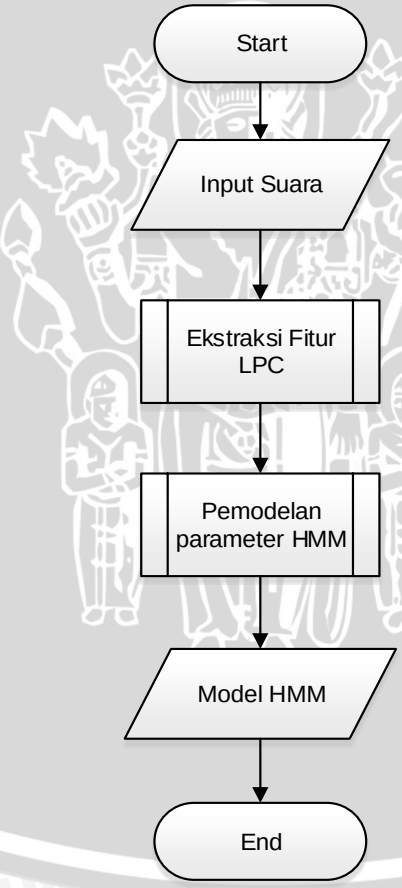
5. Output

   Output dari pembelajaran pola adalah pola model dari masing-masing kelas. Sedangkan output dari perbandingan dengan pola adalah kelas pola yang paling cocok dari data uji yang telah di-*input*-kan yang berupa huruf aksara jawa.

3.2.2.2 Perancangan Proses Pembelajaran Pola

Perancangan proses pembelajaran pola merupakan perancangan tahap atau urutan sistem untuk melakukan proses pembentukan pola model referensi. Masukan yang digunakan pada aplikasi ini berupa hasil rekaman suara *user* yang mempunyai ekstensi wav.

Proses pertama yang dilakukan oleh sistem adalah membaca file inputan *user*. Inputan tersebut kemudian akan melalui proses ekstraksi fitur menggunakan LPC dan setelah itu akan dilakukan pemodelan parameter HMM untuk menghasilkan pola model referensi HMM. Gambar 3.5 menunjukkan diagram alir pemodelan HMM.



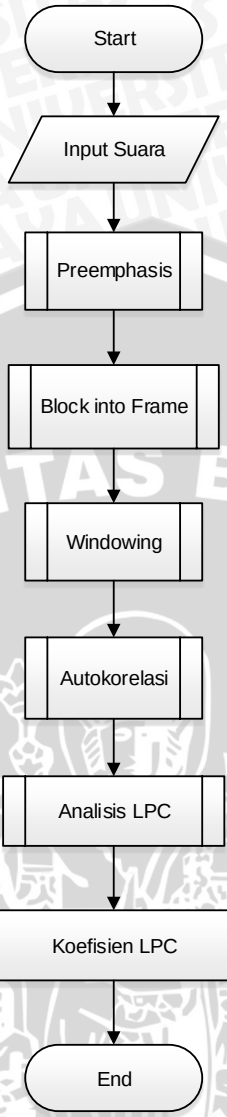
Gambar 3.5 Diagram Alir Pemodelan HMM

3.2.2.3 Ekstraksi Fitur LPC

Tahap ekstraksi fitur mempunyai beberapa proses yang dikerjakan oleh sistem. Proses-proses tersebut antara lain adalah *preemphasis, block into frame, windowing,* autokorelasi, dan analisis LPC. Pada tahap ekstraksi fitur langkah-langkahnya antara lain :

1. Memasukkan data hasil pembacaan input suara / file .wav.
2. Sinyal dari suara yang diinputkan akan diproses menjadi sinyal digital dalam fungsi *preemphasis*.
3. Sinyal yang telah dipreemphasis kemudian diblok menjadi beberapa frame yang berisikan *sample-sample* sinyal dalam fungsi *block into frame*.
4. Pembuatan *frame* dengan jumlah *frame* yang sudah dihitung dari langkah sebelumnya kemudian dilakukan penjendelaan dengan *Hamming Window*.
5. Menghitung nilai autokorelasi dari hasil *windowing*.
6. Menghitung dan menyimpan hasil koefisien LPC yang dihasilkan dari analisis LPC.

Diagram alir LPC dapat digambarkan pada Gambar 3.6 berikut ini :



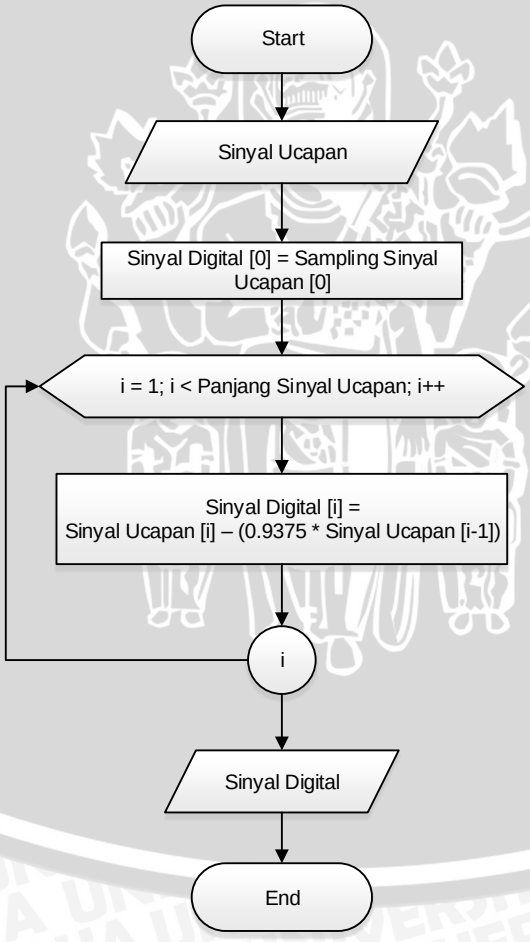
Gambar 3.6 Diagram Alir LPC

a) Perancangan Preemphasis

Pada proses ini sinyal ucapan yang telah diinputkan akan diubah menjadi sinyal digital. Pada proses preemphasis langkah-langkahnya antara lain :

1. Memasukkan sinyal ucapan yang akan diproses.
2. Menginisialisasi sinyal digital ke 0 dengan nilai sinyal ucapan ke 0.
3. Melakukan perulangan panjang sinyal ucapan pada sinyal ucapan.
4. Menghitung nilai sinyal digital dengan rumus dalam persamaan (2.1).
5. Melakukan iterasi sejumlah panjang sinyal ucapan sampai selesai dan didapatkan nilai-nilai sinyal digital.

Diagram alir proses preemphasis dapat digambarkan pada Gambar 3.7 berikut ini :



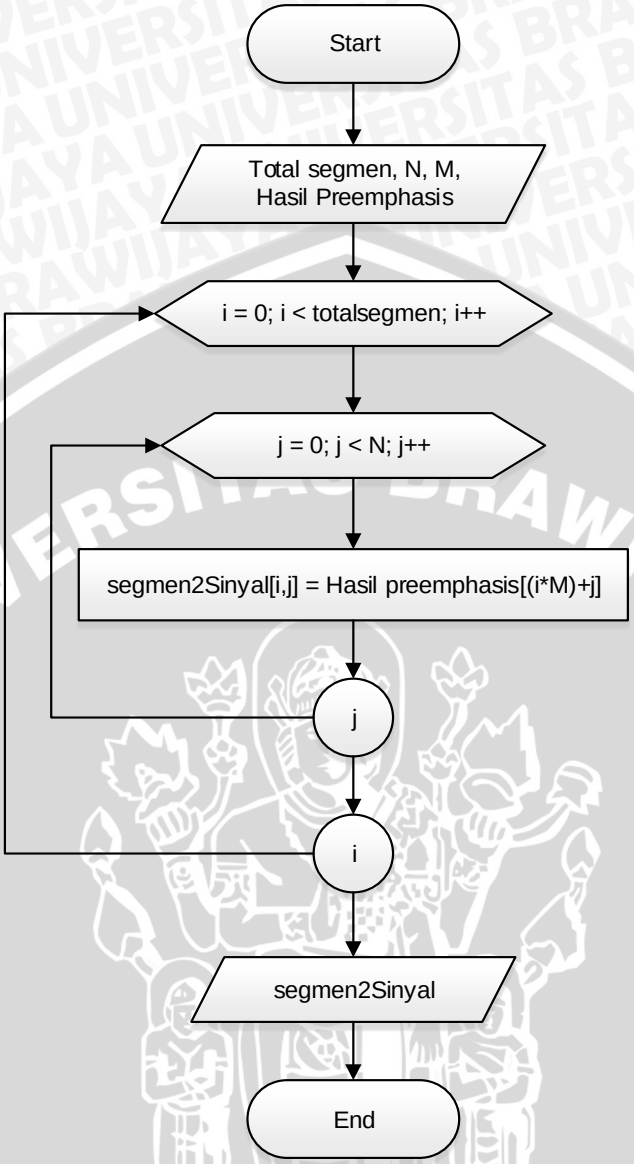
Gambar 3.7 Diagram Alir Preemphasis

b) Perancangan Frame Blocking

Pada proses ini sinyal yang telah dipreemphasis, diblok menjadi beberapa bagian dengan jumlah sample N, dan tiap bagian dipisahkan dengan sejumlah M sample. Pada proses block into frame langkah-langkahnya antara lain :

1. Memasukkan sinyal hasil preemphasis, jumlah sample (N), jumlah sample pemisah (M), dan panjang segmen.
2. Melakukan perulangan totalsegmen dan jumlah sample per frame (N).
3. Membentuk segmen sinyal dari hasil preemphasis.
4. Melakukan iterasi hingga mendapatkan hasil segmen-segmen dari sinyal preemphasis.

Diagram alir proses block into frame dapat digambarkan pada Gambar 3.8 berikut ini :



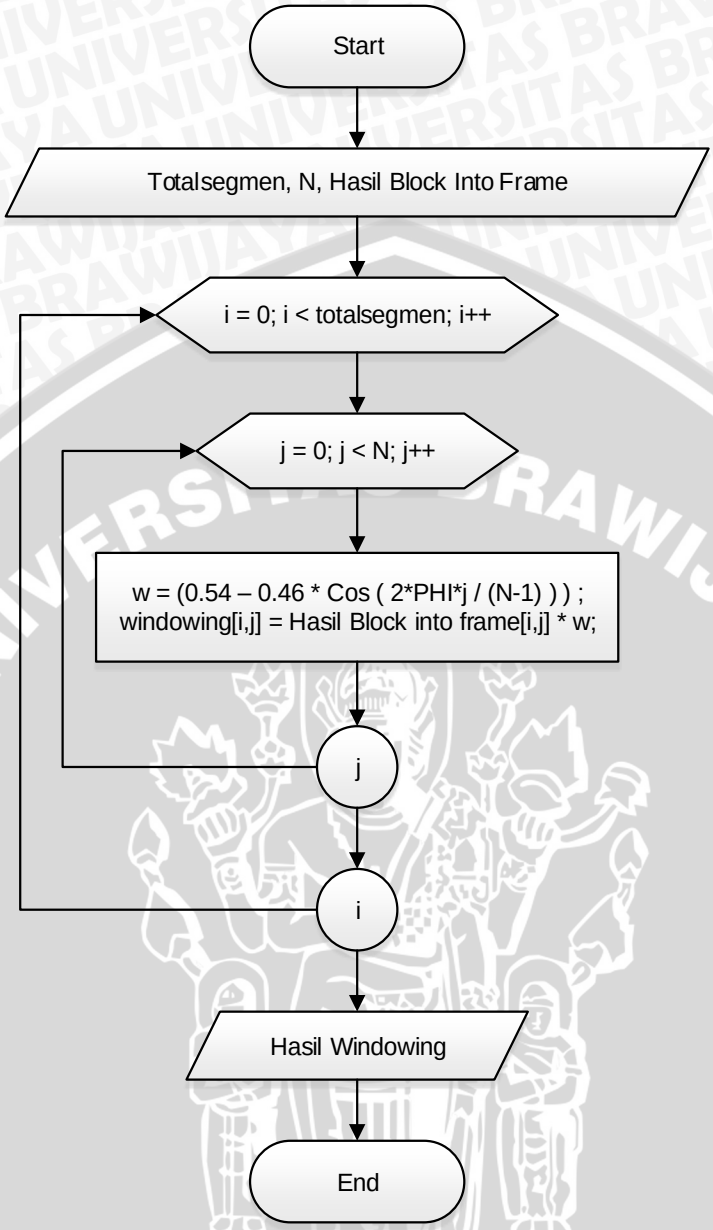
Gambar 3.8 Diagram Alir Block Into Frame

c) Perancangan Windowing

Pada proses ini dilakukan proses penjendelaan pada setiap bagian sinyal yang telah dibuat sebelumnya. Hal ini dilakukan untuk meminimalkan diskontinuitas pada ujung awal dan ujung akhir setiap *frame* dengan persamaan *Hamming Window*. Pada proses windowing langkah-langkahnya antara lain :

1. Memasukkan hasil *block into frame*, total segmen, jumlah sample per segmen (N).
2. Melakukan perulangan total segmen dan jumlah sample (N).
3. Memberi nilai pada variabel *hamming window* dengan persamaan (2.2).
4. Membentuk array *windowing* dengan mengalikan hasil *block into frame* dengan variabel *hamming window*.
5. Melakukan iterasi hingga mendapatkan hasil *windowing*.

Diagram alir proses *windowing* dapat digambarkan pada Gambar 3.9 berikut ini :



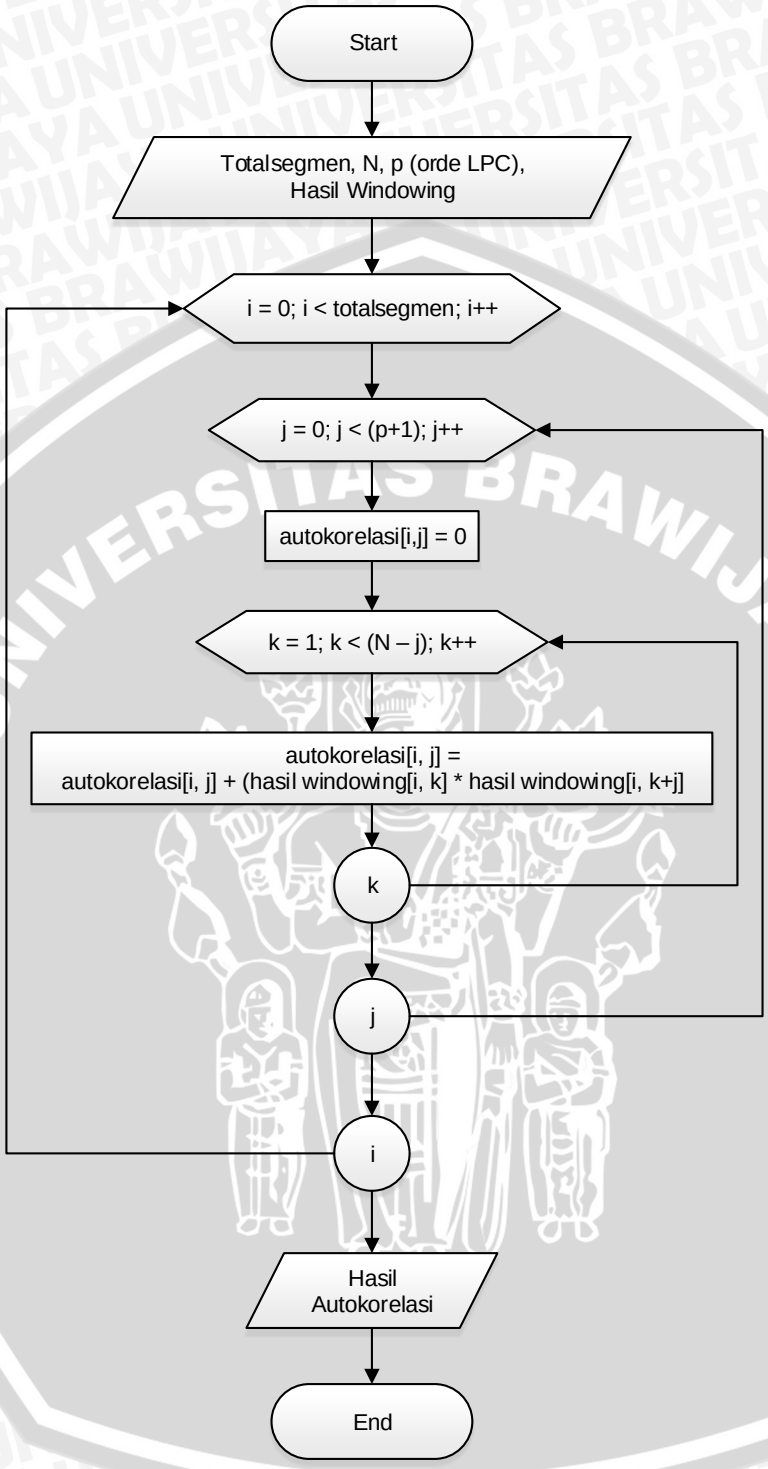
Gambar 3.9 Diagram Alir *Windowing*

d) Perancangan Autokorelasi

Pada proses ini tiap bagian yang telah diberi *window* kemudian akan dibentuk autokorelasinya. Pada proses autokorelasi langkah-langkahnya antara lain :

1. Memasukkan hasil proses *windowing*, total segmen, jumlah sample per frame (N), orde LPC (p).
2. Melakukan perulangan total segmen dan orde LPC+1.
3. Menginisialisasi variabel autokorelasi.
4. Melakukan perulangan jumlah sample (N) dikurangi orde LPC yang sedang diproses.
5. Membentuk nilai autokorelasi dengan persamaan (2.3).
6. Melakukan iterasi hingga mendapatkan hasil autokorelasi.

Diagram alir proses autokorelasi dapat digambarkan pada Gambar 3.10 berikut ini :



Gambar 3.10 Diagram Alir Autokorelasi

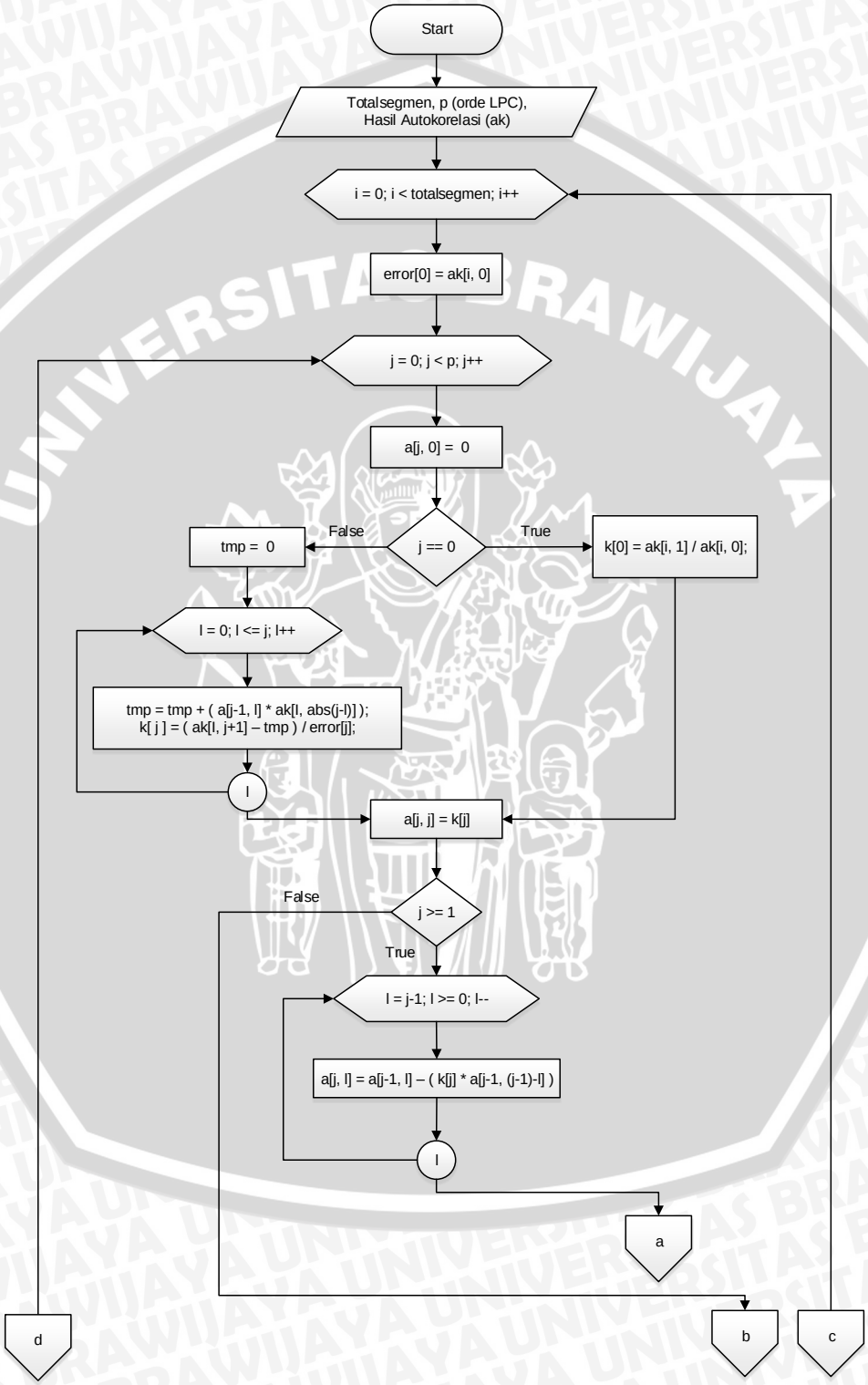
e) Perancangan Analisis LPC

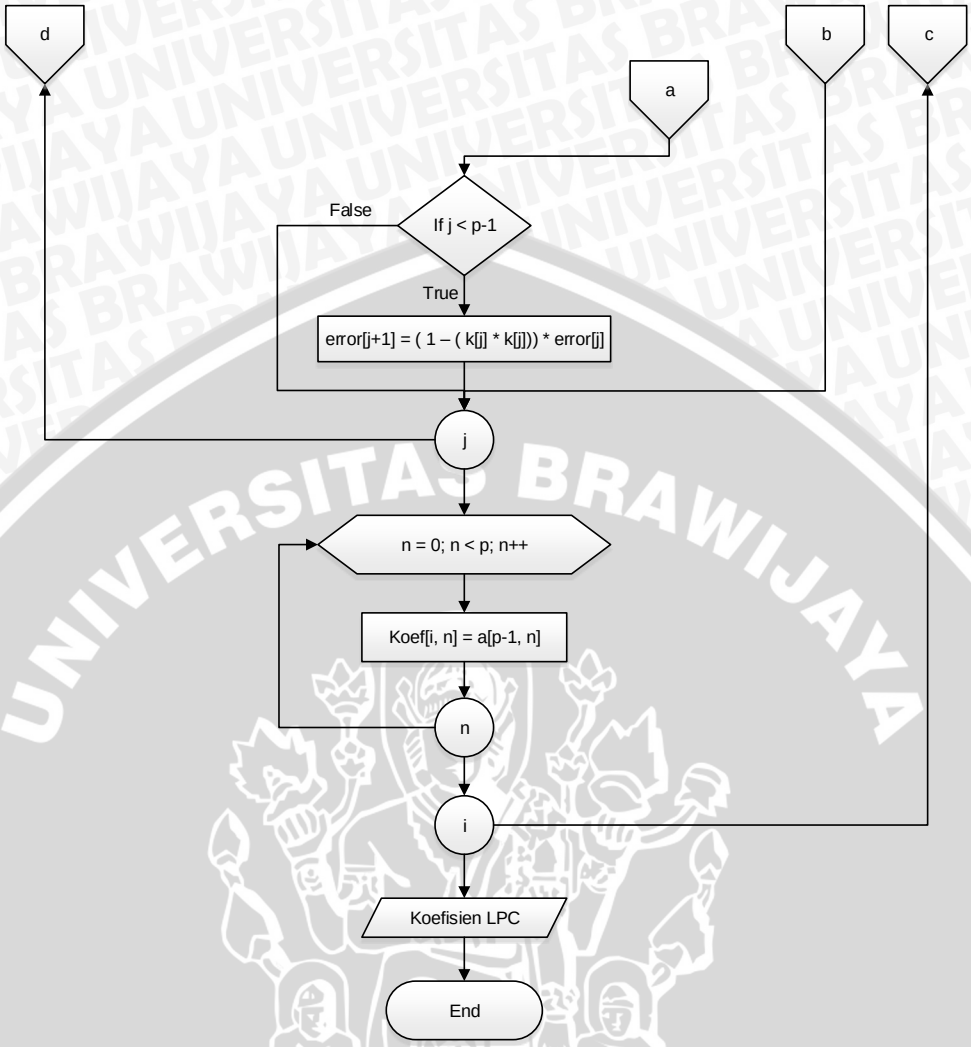
Pada proses ini semua nilai autokorelasi yang telah dihitung pada tahap sebelumnya akan diubah menjadi parameter LPC. Pada proses analisa LPC langkah-langkahnya antara lain :

1. Memasukkan hasil autokorelasi, total segmen, orde LPC (p).
2. Melakukan perulangan total segmen kemudian menginisialisasi variabel error ke 0 dengan nilai autokorelasi dari index ke [totalsegmen(i), 0].
3. Melakukan perulangan orde LPC kemudian menginisialisasi nilai koefisien prediksi ($a^{ordeLPC}$) index ke 0 dengan nilai 0.
4. Jika orde LPC = 0 maka memberi nilai koefisien pantulan (k) ke 0 dari nilai autokorelasi.
5. Jika orde LPC tidak sama dengan 0 maka hitung nilai temporary. Pada proses berikutnya melakukan perulangan orde LPC yang sedang diproses.
6. Membentuk nilai koefisien pantulan (k) dengan persamaan (2.6).
7. Melakukan iterasi hingga batas mencapai orde LPC–1.
8. Memberi nilai koefisien prediksi ($a^{ordeLPC}$) index ke orde LPC yang diproses dengan nilai koefisien pantulan (k) yang sudah dihitung.
9. Jika orde LPC >= 1 maka melakukan perulangan orde LPC dengan nilai awal ordeLPC-1. Jika tidak maka memberi nilai $error^{ordeLPC}$ dengan persamaan (2.9).
10. Memberi nilai koefisien prediksi ($a^{ordeLPC}$) dengan persamaan (2.8).
11. Melakukan iterasi hingga batas minimal sama dengan 0.
12. Melakukan iterasi hingga batas maksimal mencapai nilai sebesar orde LPC (p).
13. Melakukan perulangan dengan nilai awal 0 dan batas maksimal = (ordeLPC-1).
14. Membentuk nilai koefisien LPC dari koefisien prediksi (a) yang telah diproses.
15. Melakukan iterasi hingga menghasilkan koefisien LPC dari segmen yang sedang diproses.

16. Melakukan iterasi hingga semua segmen menghasilkan koefisien LPC.

Diagram alir proses analisa LPC dapat digambarkan pada Gambar 3.11 berikut ini :





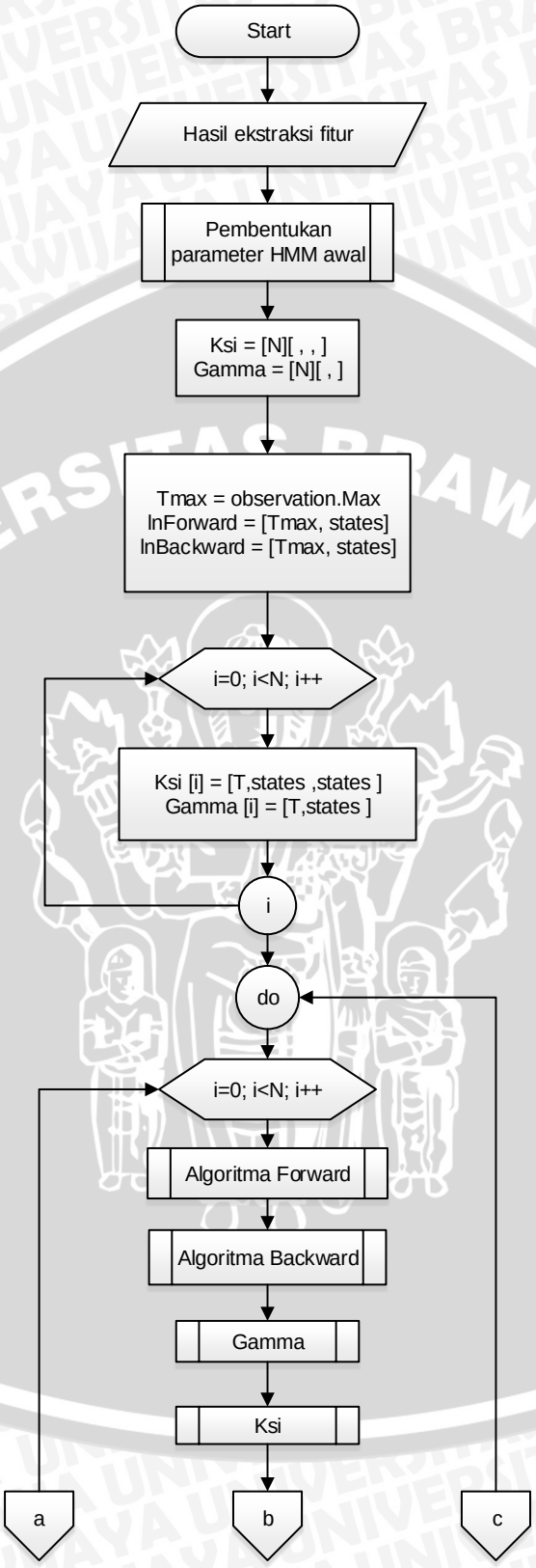
Gambar 3.11 Diagram Alir Analisis LPC

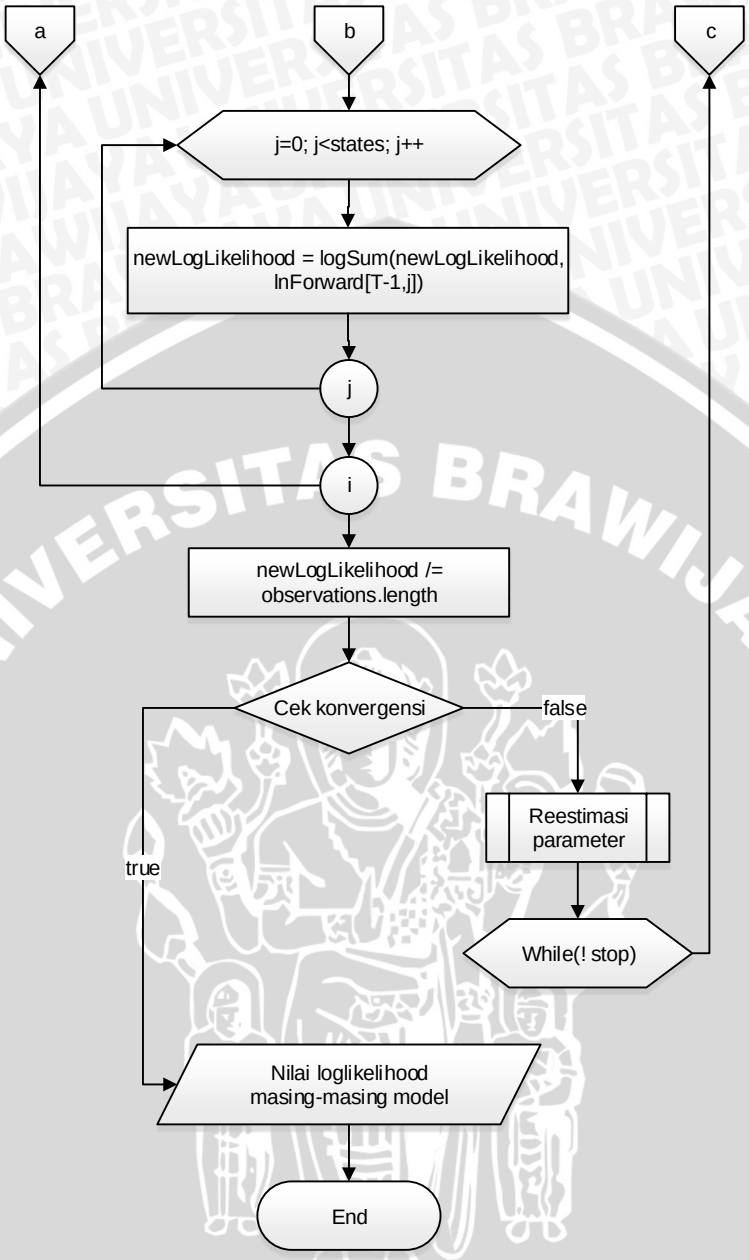
3.2.2.4 Pembelajaran Pola HMM

Tahap pembelajaran dan pengenalan pola mempunyai beberapa proses yang dikerjakan oleh sistem. Langkah-langkah dalam proses parameter HMM adalah sebagai berikut :

1. Hasil ekstraksi fitur.
2. Membentuk parameter HMM awal.
3. Menentukan ukuran matriks *ksi* dan *gamma*.
4. Melakukan perulangan untuk jumlah label dan menentukan definisi matriks *gamma* dan *ksi*.
5. Melakukan perulangan untuk jumlah label.
6. Menggunakan algoritma *forward* untuk menghitung nilai *ksi* dan *gamma*.
7. Menggunakan algoritma *backward* untuk menghitung nilai *ksi* dan *gamma*.
8. Menghitung nilai matriks *gamma*.
9. Menghitung nilai matriks *ksi*.
10. Melakukan perulangan untuk jumlah deretan atau *sequence*.
11. Menghitung nilai *loglikelihood* baru.
12. Melakukan iterasi hingga selesai.
13. Menetapkan *loglikelihood* baru sama dengan *loglikelihood* baru dibagi panjang observasi.
14. Jika rata-rata *loglikelihood* baru sudah konvergen maka ke langkah 15, jika tidak maka parameter HMM akan diestimasi kembali sampai konvergen.
15. Menyimpan nilai *loglikelihood* model.
16. Proses parameter HMM selesai.

Diagram alir proses pembelajaran dan pengenalan pola dengan HMM dapat digambarkan pada Gambar 3.12 berikut ini :


Start
Hasil ekstraksi fitur
Pembentukan parameter HMM awal
Ksi = [N][, ,]
Gamma = [N][,]
Tmax = observation.Max
lnForward = [Tmax, states]
lnBackward = [Tmax, states]
i=0; i<N; i++
Ksi [i] = [T,states ,states]
Gamma [i] = [T,states]
i
do
i=0; i<N; i++
Algoritma Forward
Algoritma Backward
Gamma
Ksi
a
b
c



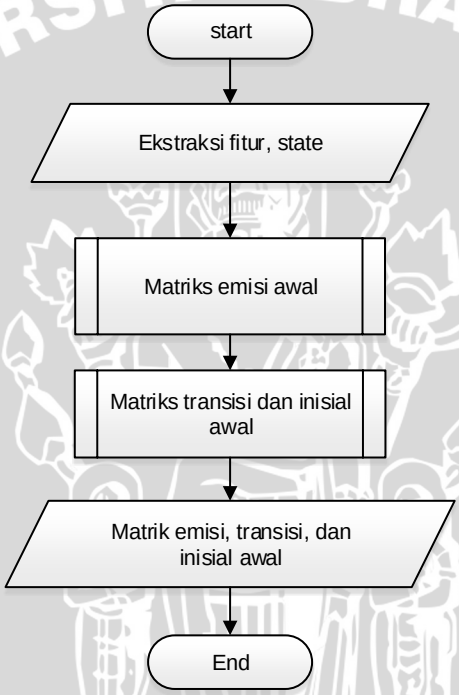
Gambar 3.12 Diagram Alir HMM

a) Perancangan Parameter HMM Awal

Langkah-langkah dalam proses parameter HMM awal adalah :

1. Hasil ekstraksi fitur dan *state* sebagai data input
2. Memproses data input ke dalam proses matriks emisi awal.
3. Memproses data input ke dalam proses matriks transisi dan inisial awal.
4. Proses parameter HMM awal selesai

Diagram alir proses parameter HMM awal dapat digambarkan pada Gambar 3.13 berikut ini :



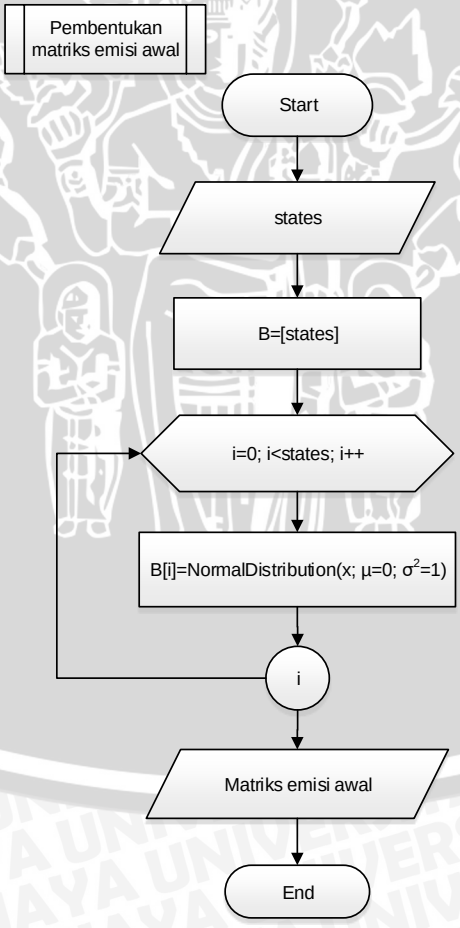
Gambar 3.13 Diagram Alir Parameter HMM Awal

Perancangan Matriks Emisi Awal

Langkah-langkah dalam proses matriks emisi awal adalah :

1. Jumlah *state* sebagai data input.
2. Menentukan ukuran matriks emisi /observasi.
3. Melakukan perulangan untuk *state*.
4. Menentukan matriks emisi ke-i yaitu distribusi normal dengan mean = 0 dan standar deviasi = 1.
5. Melakukan iterasi hingga selesai dan didapatkan nilai matriks emisi awal.
6. Proses matriks emisi awal selesai.

Diagram alir proses matriks emisi awal dapat digambarkan pada Gambar 3.14 berikut ini :



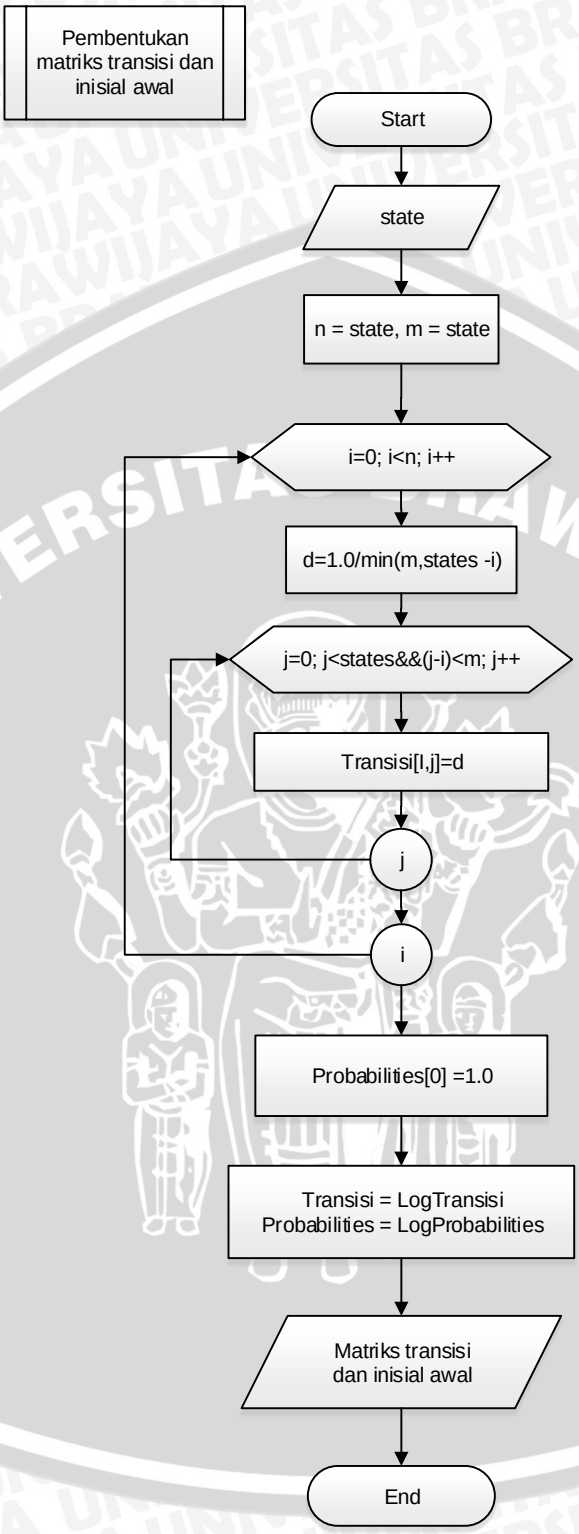
Gambar 3.14 Diagram Alir Matriks Emisi Awal

Perancangan Matriks Transisi dan Inisial Awal

Langkah-langkah pada proses transisi dan inisial awal adalah :

1. Nilai *state* sebagai data input.
2. Menentukan ukuran *n*.
3. Melakukan perulangan untuk *state i*.
4. Melakukan perulangan untuk *state j*.
5. Melakukan perhitungan nilai transisi.
6. Melakukan iterasi *state i* dan *j* hingga selesai.
7. Menentukan nilai probabilitas urutan ke 0 dari *array*.
8. Menyimpan hasil log matriks transisi dan inisial awal.
9. Proses matriks transisi dan matriks inisial awal selesai.

Diagram alir proses matriks transisi dan inisial awal dapat digambarkan pada Gambar 3.15 berikut ini :



Gambar 3.15 Diagram Alir Matriks Transisi dan Inisial Awal

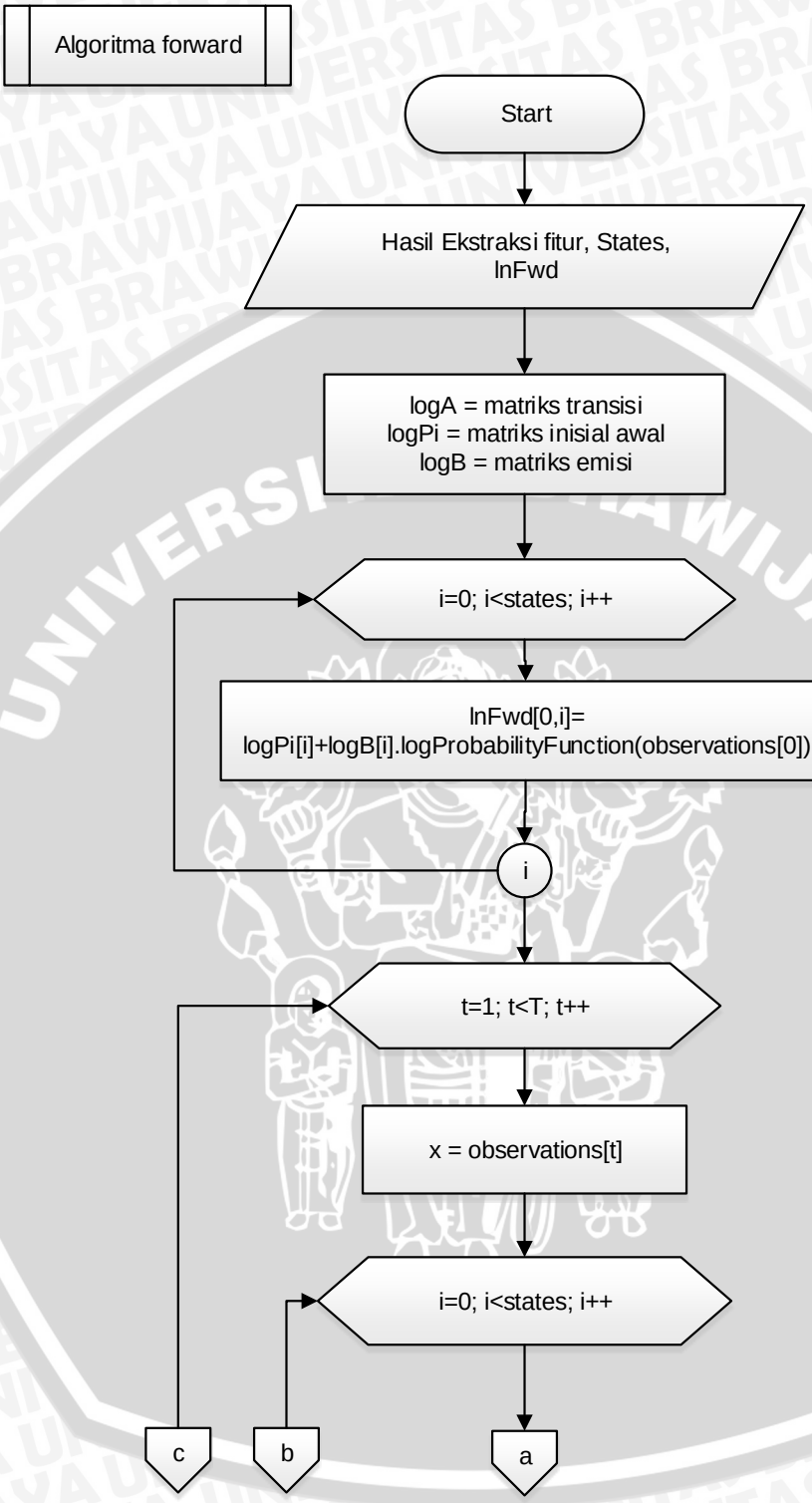
b) **Perancangan Algoritma *Forward***

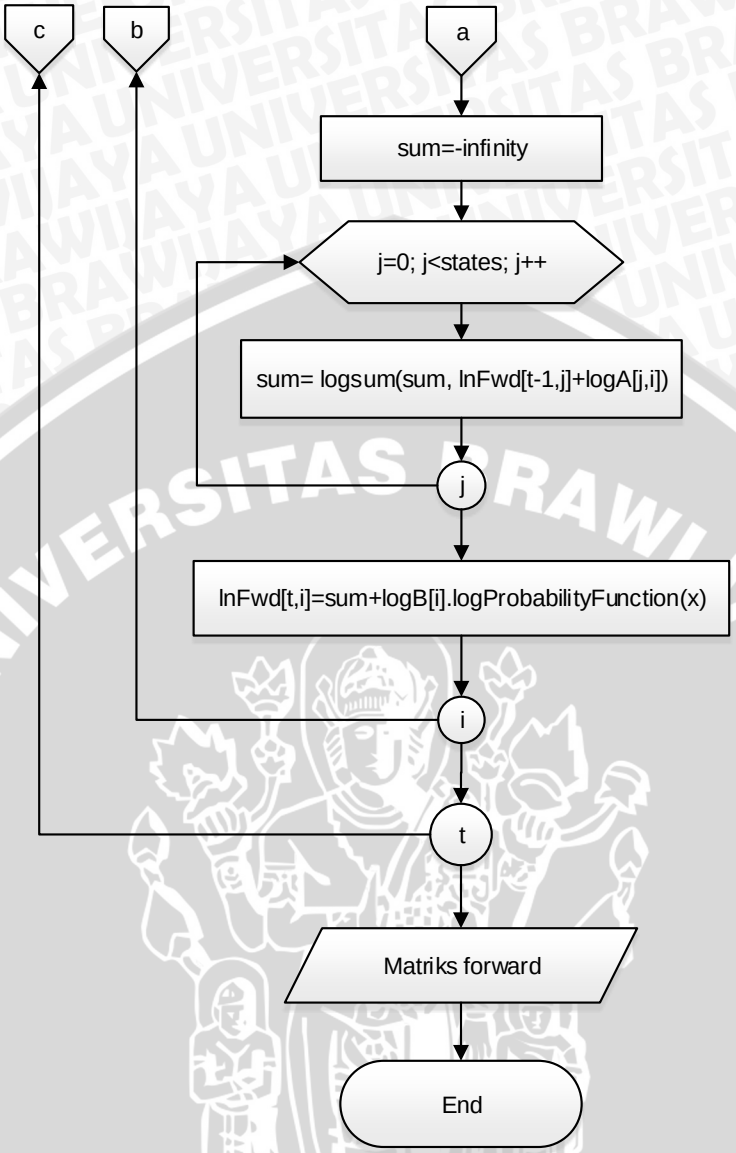
Algoritma *forward* digunakan untuk mengevaluasi parameter HMM. Sesuai dengan namanya algoritma *forward* berjalan dengan runut maju.

Langkah-langkah pada proses algoritma *forward* adalah :

1. Hasil ekstraksi fitur, *state*, matriks transisi, inisial dan emisi sebagai masukan.
2. Melakukan perulangan untuk *state i*.
3. Menghitung nilai inisialisasi dari algoritma *forward*.
4. Melakukan perulangan untuk panjang *sequence*.
5. Menginisialisai nilai x dengan *sequence* ke-t.
6. Melakukan perulangan untuk *state i*.
7. Menginisialisasi nilai sum.
8. Melakukan perulangan untuk *state j*.
9. Menghitung nilai sum dari nilai sum sebelumnya, nilai matriks *forward* sebelumnya, dan matriks transisi.
10. Menghitung proses induksi dari algoritma *forward* dengan menjumlahkan nilai sum dan nilai *probability density function* matriks emisi ke-*i* yang dievaluasi menggunakan nilai x.
11. Melakukan iterasi hingga selesai dan didapatkan output nilai matriks *forward*.
12. Proses algoritma *forward* selesai.

Adapun Diagram alir proses algoritma *forward* dapat digambarkan pada Gambar 3.16 berikut ini :


Algoritma forward
Start
Hasil Ekstraksi fitur, States, lnFwd
logA = matriks transisi
logPi = matriks inisial awal
logB = matriks emisi
i=0; i<states; i++
lnFwd[0,i]= logPi[i]+logB[i].logProbabilityFunction(observations[0])
i
t=1; t<T; t++
x = observations[t]
i=0; i<states; i++
c
b
a



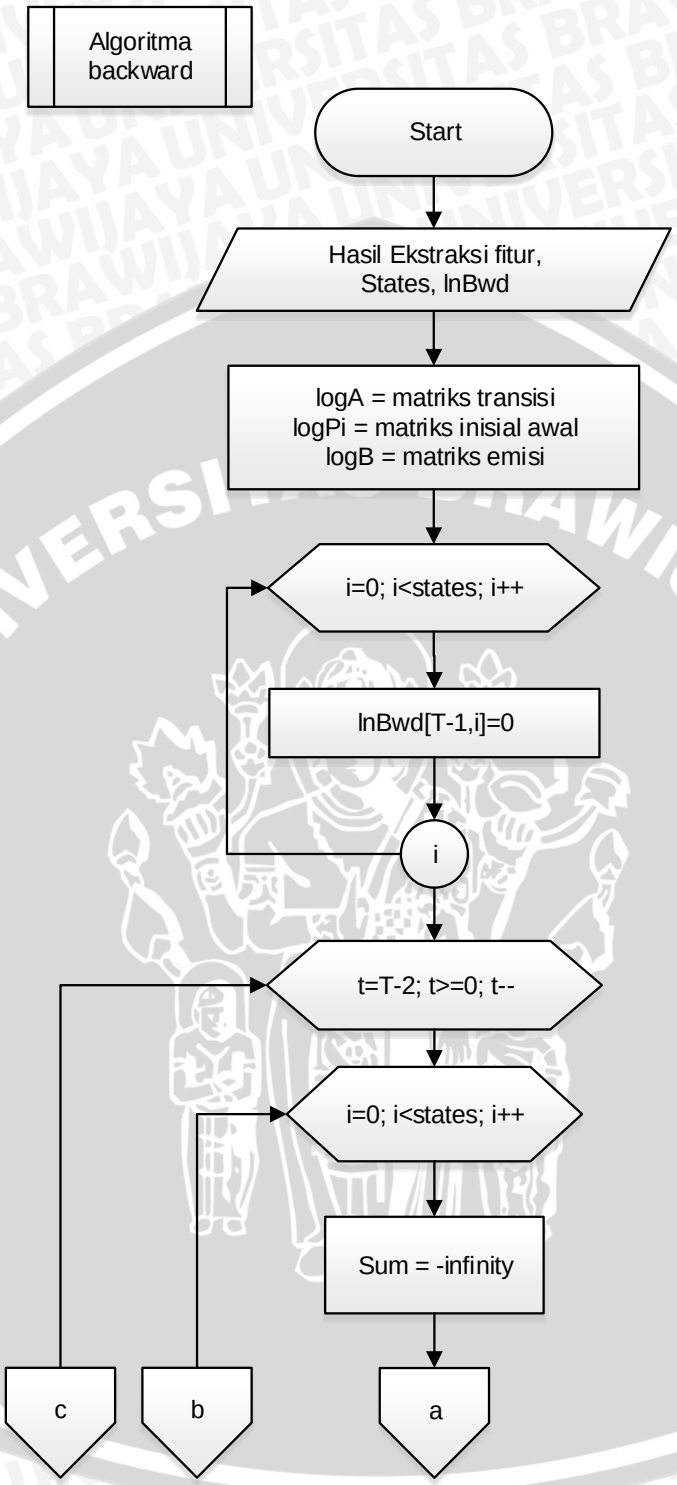
Gambar 3.16 Diagram Alir Algoritma Forward

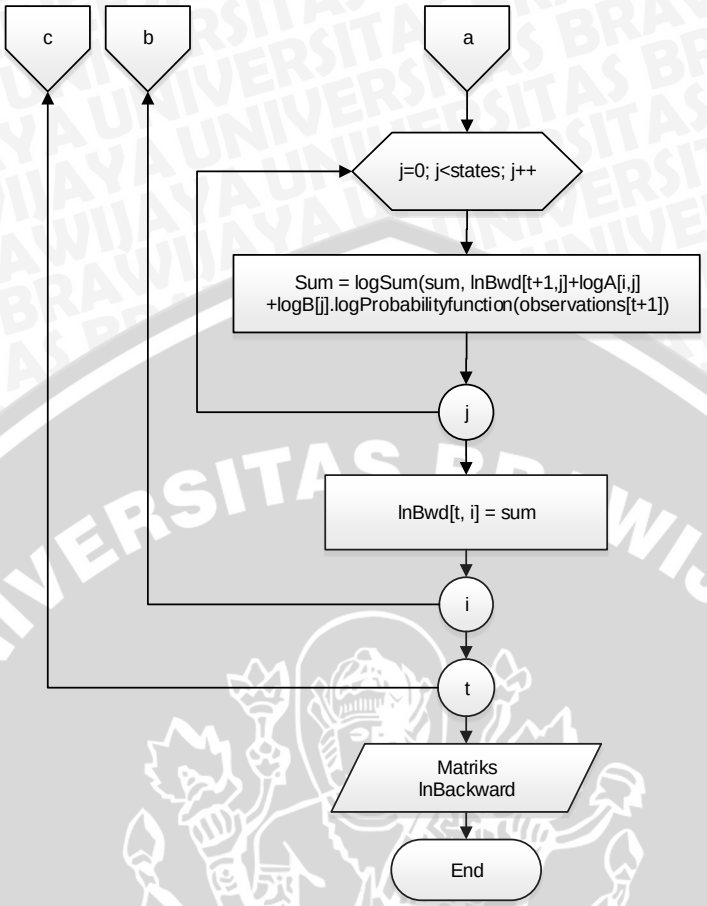
c) Perancangan Algoritma *Backward*

Langkah-langkah dalam proses algoritma *backward* adalah :

1. Hasil ekstraksi fitur, *state*, matriks transisi, inisial, dan emisi sebagai masukan.
2. Melakukan perulangan untuk *state i*.
3. Menghitung nilai inisialisasi algoritma *backward*.
4. Melakukan perulangan untuk panjang *sequence*.
5. Melakukan perulangan untuk *state i*.
6. Menginisialisasi nilau sum.
7. Melakukan perulangan untuk *state j*.
8. Menghitung nilai sum dari nilai sum sebelumnya, matriks transisi dan nilai *probability density function* matriks emisi ke-*i* yang dievaluasi menggunakan *sequence* ke-t+1.
9. Menghitung proses induksi dari algoritma *backward* dari nilai sum yang dihasilkan pada iterasi sebelumnya.
10. Melakukan iterasi hingga selesai dan didapatkan output nilai matriks *backward*.
11. Proses algoritma *backward* selesai.

Diagram alir proses algoritma *backward* dapat digambarkan pada Gambar 3.17 berikut ini :


Algoritma backward
Start
Hasil Ekstraksi fitur, States, lnBwd
logA = matriks transisi
logPi = matriks inisial awal
logB = matriks emisi
i=0; i<states; i++
lnBwd[T-1,i]=0
i
t=T-2; t>=0; t--
i=0; i<states; i++
Sum = -infinity
c
b
a



Gambar 3.17 Diagram Alir Algoritma Backward

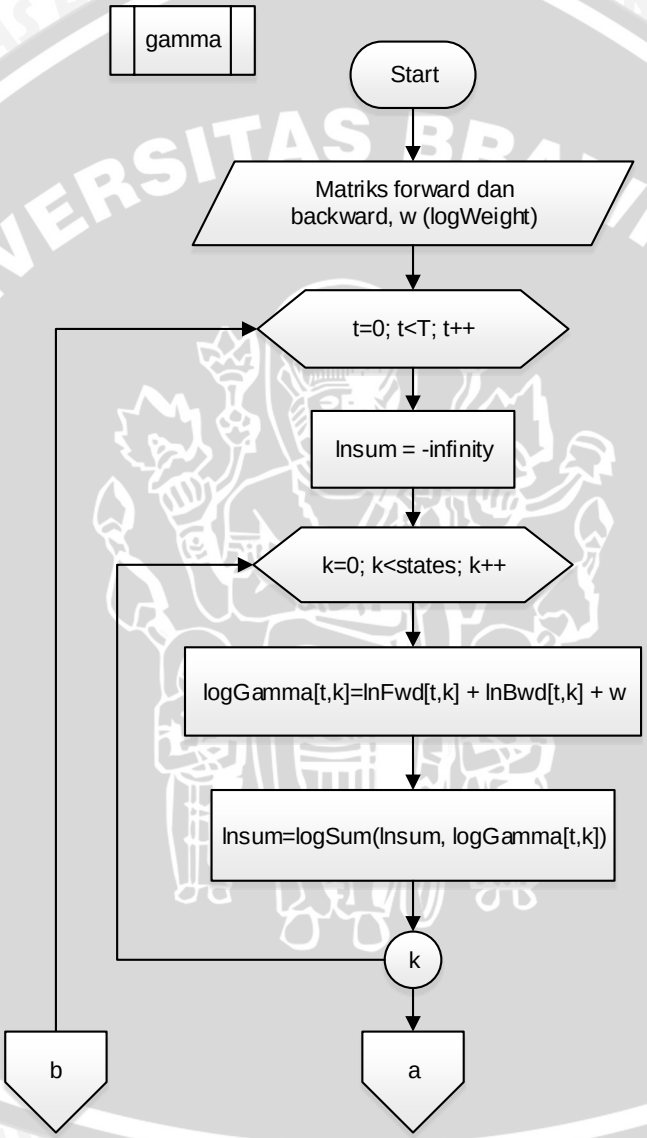
d) **Perancangan Matriks *Gamma***

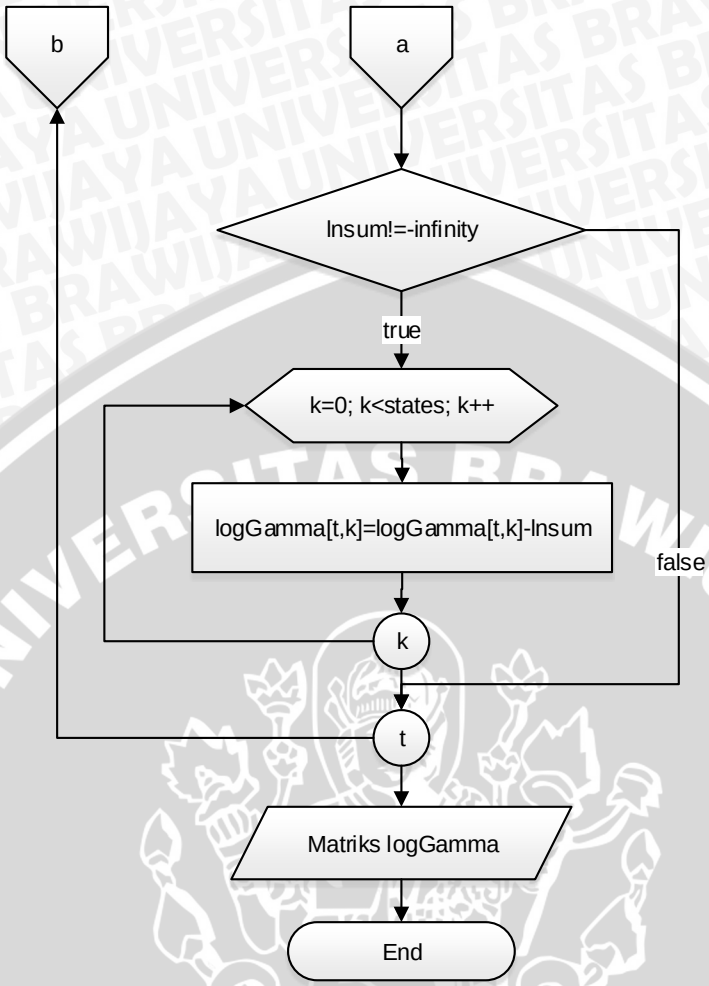
Langkah-langkah pada proses matriks *gamma* adalah :

1. Matriks *forward*, *backward* dan *weight* sebagai masukan.
2. Melakukan perulangan untuk panjang *sequence.*
3. Menginisialisasi nilai *lnsum*.
4. Melakukan perulangan untuk *state.*
5. Menghitung nilai matriks *gamma* dengan menjumlahkan nilai matriks *forward*, matriks *backward* dan w.
6. Menghitung nilai baru *lnsum* dari nilai *lnsum* sebelumnya dan nilai matriks *gamma.*
7. Jika nilai *lnsum* tidak sama dengan -infinity maka ke langkah 8, jika sama dengan -infinity maka ke langkah 10.

8. Melakukan perulangan untuk *state*.
9. Menghitung nilai *gamma* yang baru.
10. Melakukan iterasi hingga selesai dan didapatkan output nilai matriks *gamma*.

Diagram alir proses matriks *gamma* dapat digambarkan pada Gambar 3.18 berikut ini :





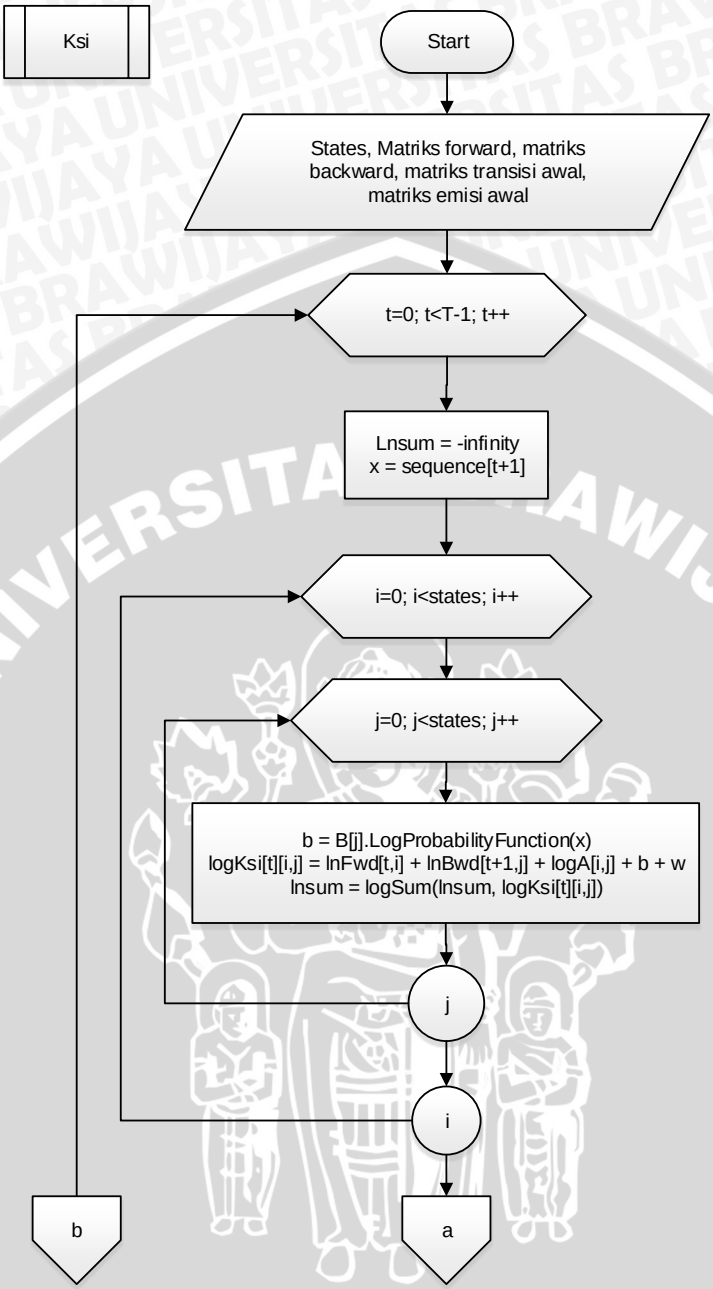
Gambar 3.18 Diagram Alir Matriks Gamma

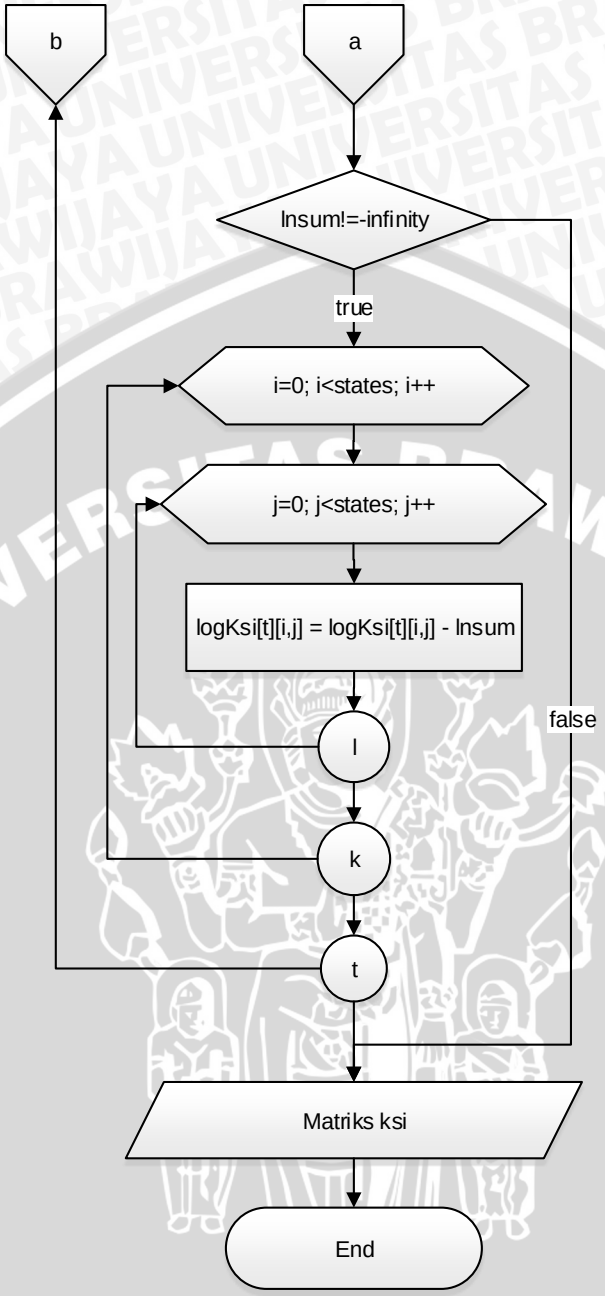
e) **Perancangan Matriks *Ksi***

Langkah-langkah pada proses matriks *ksi* adalah :

1. State, matriks *forward*, matriks *backward*, matriks transisi awal, dan matriks emisi awal sebagai data input.
2. Melakukan perulangan untuk panjang deretan / *sequence*.
3. Inisialisasi lnsum dan x (sequence ke-t+1).
4. Melakukan perulangan untuk *state i.*
5. Melakukan perulangan untuk *state j.*
6. Menghitung nilai b dengan nilai *probability density function* matriks emisi ke-*j* yang dievaluasi menggunakan *sequence* ke-x.
7. Menghitung matriks *ksi* dengan menjumlahkan matriks *forward*, matriks *backward*, matriks transisi, nilai b, dan w.
8. Menghitung nilai *lnsum* yang baru dari *lnsum* sebelumnya dan matriks *ksi*.
9. Melakukan perulangan *state j* hingga selesai.
10. Melakukan perulangan *state i* hingga selesai.
11. Jika *lnsum* tidak sama dengan -infinity maka ke langkah 12, jika tidak maka ke langkah 7.
12. Melakukan perulangan untuk *state i*.
13. Melakukan perulangan untuk *state j*.
14. Menghitung nilai baru dari matriks *ksi*.
15. Melakukan iterasi hingga selesai dan didapatkan output nilai matriks *ksi* yang baru.
16. Proses matriks *ksi* selesai.

Diagram alir proses matriks *ksi* dapat digambarkan pada Gambar 3.19 berikut ini :


Ksi
Start
States, Matriks forward, matriks backward, matriks transisi awal, matriks emisi awal
t=0; t<T-1; t++
Lnsum = -infinity
x = sequence[t+1]
i=0; i<states; i++
j=0; j<states; j++
b = B[j].LogProbabilityFunction(x)
logKsi[t][i,j] = lnFwd[t,i] + lnBwd[t+1,j] + logA[i,j] + b + w
lnsum = logSum(lnsum, logKsi[t][i,j])
j
i
b
a



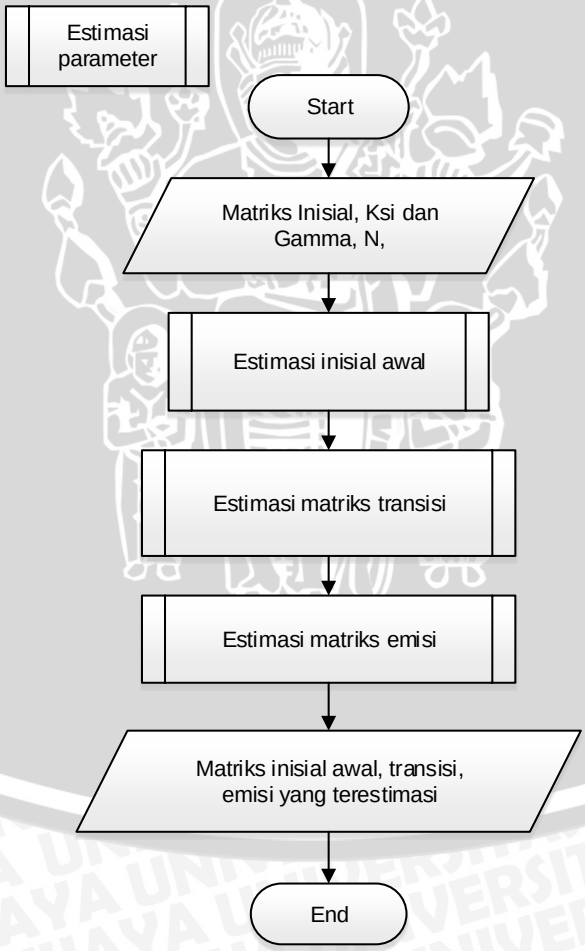
Gambar 3.19 Diagram Alir Matriks *Ksi*

f) Perancangan Re-estimasi Parameter HMM

Langkah-langkah dalam proses re-estimasi parameter HMM adalah :

1. Matriks *ksi* dan matriks *gamma* sebagai data input.
2. Masuk pada proses re-estimasi inisial awal.
3. Masuk pada proses re-estimasi matriks transisi.
4. Masuk pada proses re-estimasi matriks emisi.
5. Didapatkan output matriks inisial awal, matriks transisi, dan matriks emisi yang telah terestimasi kembali.
6. Proses re-estimasi parameter selesai.

Diagram alir proses re-estimasi parameter dapat digambarkan pada Gambar 3.20 berikut ini :



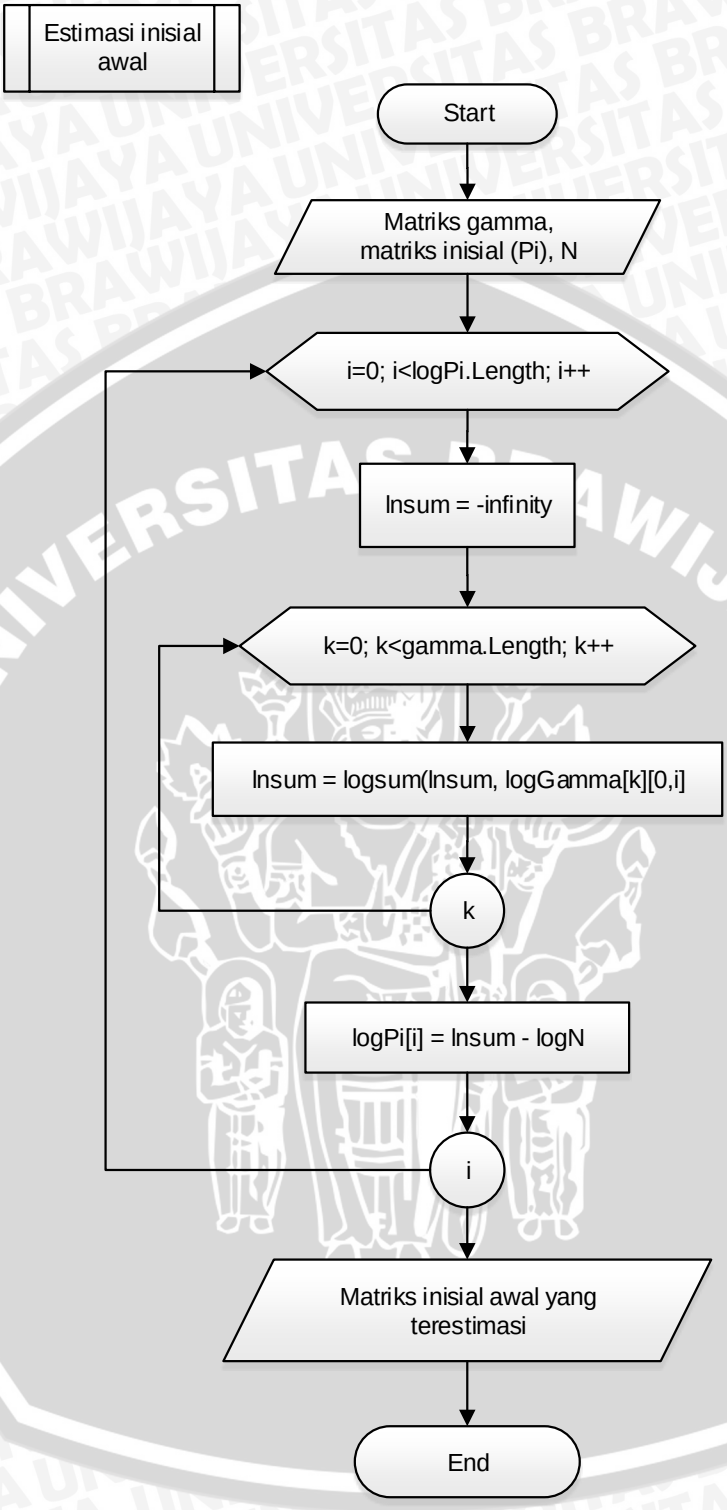
Gambar 3.20 Diagram Alir Re-estimasi Parameter HMM

Perancangan Re-estimasi Matriks Inisial

Langkah-langkah pada proses re-estimasi matriks inisial adalah :

7. Matriks *gamma,* matriks inisial awal, dan N (jumlah observasi) sebagai input.
8. Melakukan perulangan untuk panjang matriks inisial awal.
9. Menginisialisasi nilai *lnsum*.
10. Melakukan perulangan untuk panjang matriks *gamma*.
11. Melakukan proses perhitungan *lnsum* dari nilai *lnsum* sebelumnya dan matriks *gamma*.
12. Melakukan iterasi panjang matriks *gamma* hingga selesai.
13. Menghitung nilai matriks inisial.
14. Melakukan iterasi hingga selesai.
15. Proses re-estimasi matriks inisial selesai.

Diagram alir proses re-estimasi matriks inisial dapat digambarkan pada Gambar 3.21 berikut ini :



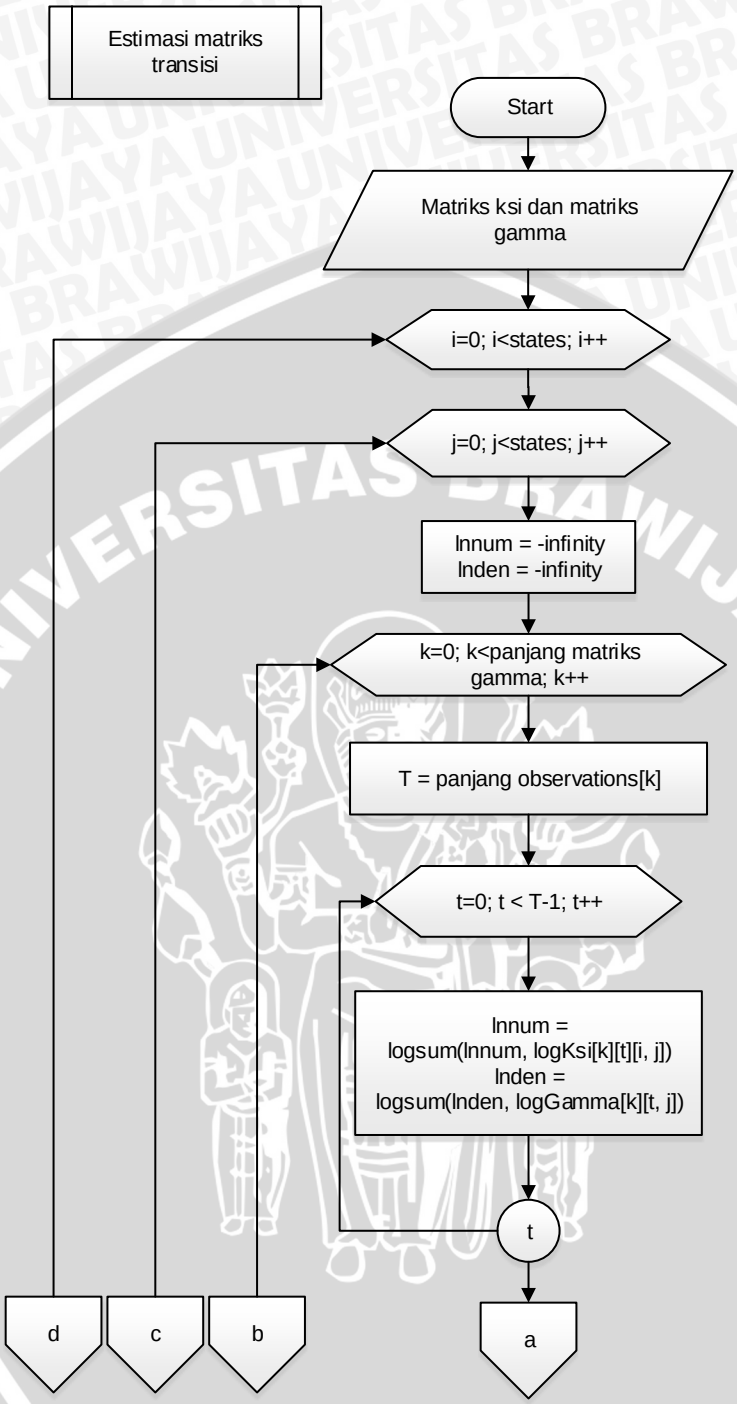
Gambar 3.21 Diagram Alir Re-estimasi Matriks Inisial

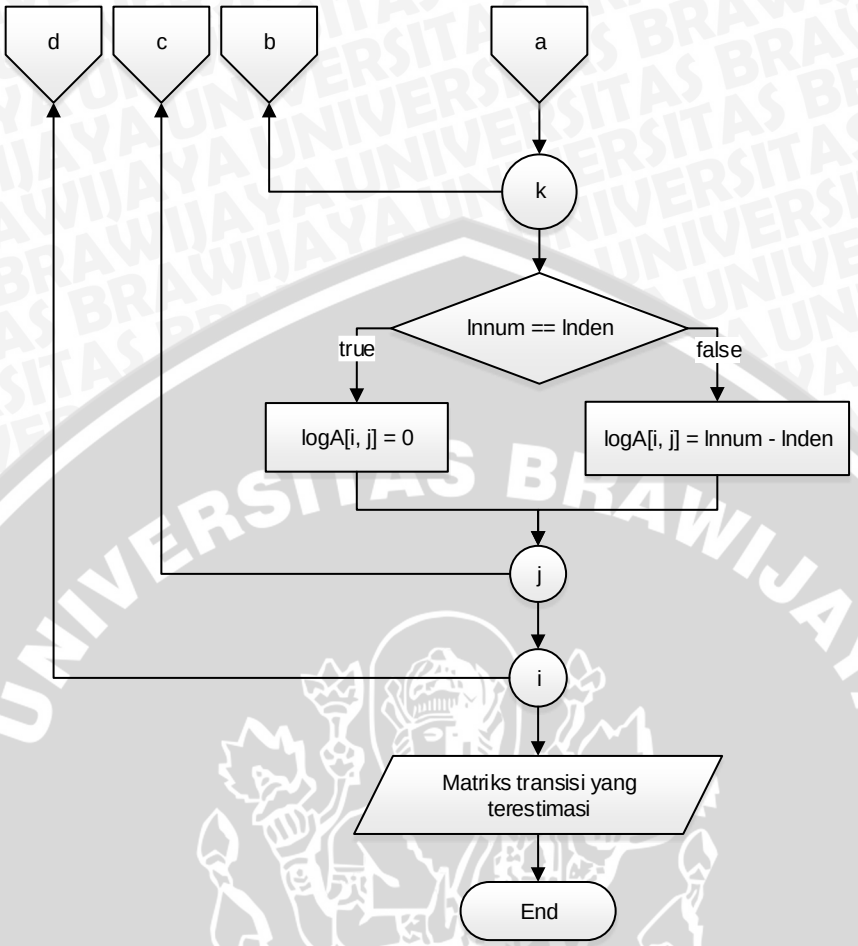
Perancangan Re-estimasi Matriks Transisi

Langkah-langkah pada proses re-estimasi matriks transisi adalah :

1. Matriks *ksi* dan *gamma* sebagai data input.
2. Melakukan perulangan untuk *state i.*
3. Melakukan perulangan untuk *state j.*
4. Menginisialisasi *lnnum* dan *lnden*.
5. Melakukan perulangan untuk panjang matriks *gamma*.
6. Menginisialisasi nilai T sebagai panjang *sequence*.
7. Melakukan perulangan untuk T (panjang *sequence*).
8. Melakukan perhitungan untuk mendapatkan nilai *lnnum* dari *lnnum* sebelumnya dan matriks *ksi* juga menghitung nilai *lnden* dari *lnden* sebelumnya dan matriks *gamma*.
9. Melakukan perulangan untuk panjang *sequence*.
10. Melakukan perulangan untuk panjang matriks *gamma*.
11. Jika nilai *lnnum* sama dengan *lnden* maka matriks transisi bernilai 0, jika tidak maka menghitung matriks transisi dari *lnnum* dan *lnden*.
12. Melakukan iterasi sampai selesai dan didapatkan output matriks transisi yang terestimasi kembali.
13. Proses re-estimasi matriks transisi selesai.

Diagram alir proses re-estimasi matriks transisi dapat digambarkan pada Gambar 3.22 berikut ini :


Estimasi matriks transisi
Start
Matriks ksi dan matriks gamma
i=0; i<states; i++
j=0; j<states; j++
lnnum = -infinity
lnden = -infinity
k=0; k<panjang matriks gamma; k++
T = panjang observations[k]
t=0; t < T-1; t++
lnnum = logsum(lnnum, logKsi[k][t][i, j])
lnden = logsum(lnden, logGamma[k][t, j])
t
a
d
c
b



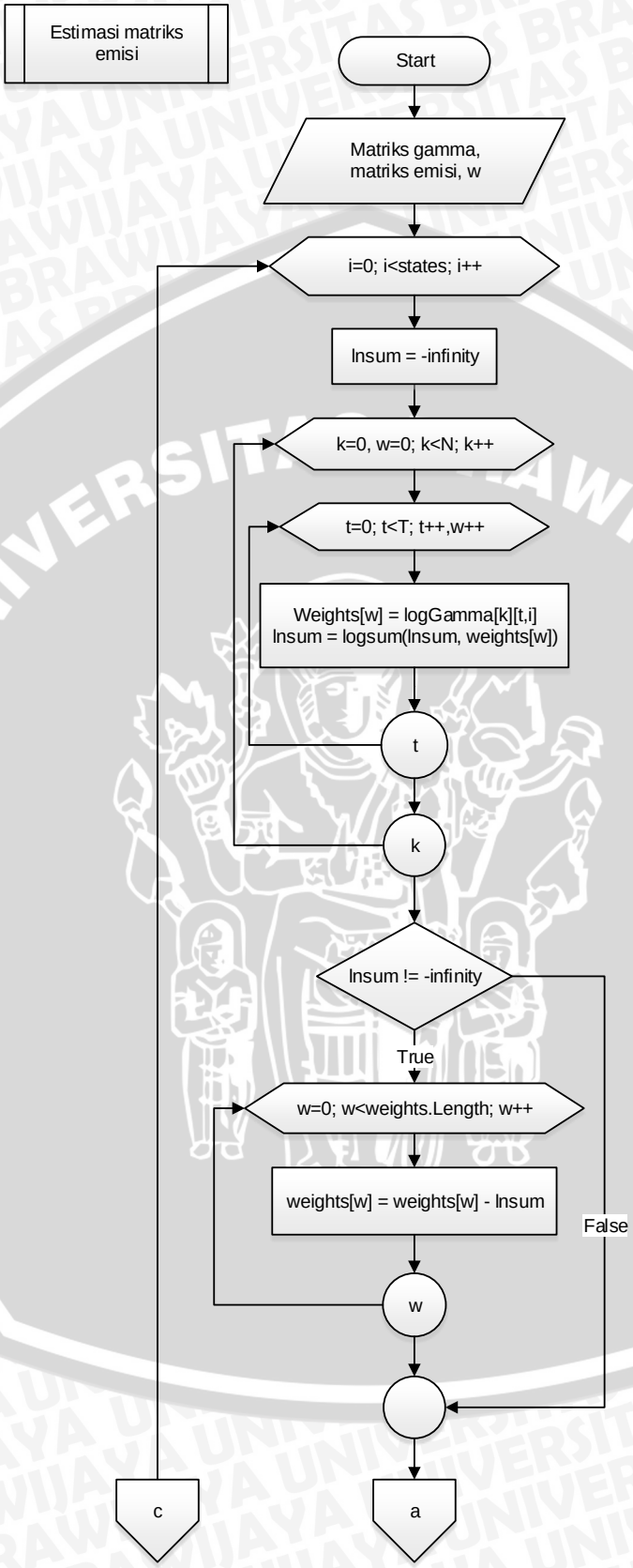
Gambar 3.22 Diagram Alir Re-estimasi Matriks Transisi

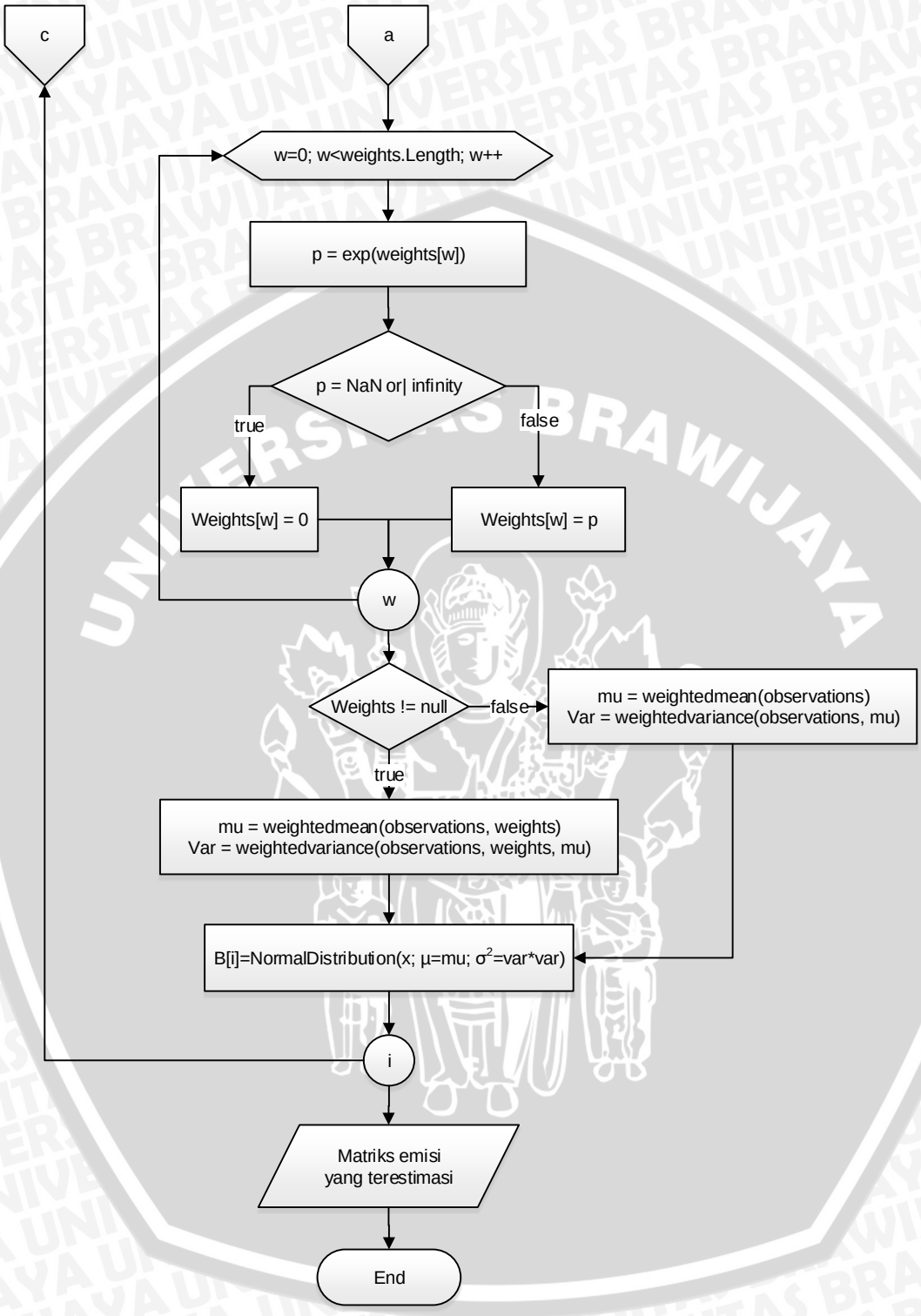
Perancangan Re-estimasi Matriks Emisi

Langkah-langkah reestimasi matriks emisi :

1. Matriks *gamma*, matriks emisi sebagai data input.
2. Melakukan perulangan untuk *state i.*
3. Menginisialisasi nilai *lnsum*.
4. Melakukan perulangan untuk jumlah label (N) dan inisialisasi indeks w.
5. Melakukan perulangan untuk panjang deretan / *sequence.*
6. Hitung nilai *weights* indeks ke w dengan matriks *gamma*.
7. Hitung nilai *lnsum* dari nilai *weights* dan *lnsum* sebelumnya.
8. Melakukan perulangan untuk panjang deretan / *sequence.*
9. Jika nilai *lnsum* ≠ *-infinity* maka melakukan perulangan untuk *weights*.
10. Menghitung nilai *weights* dari nilai *weights* yang sebelumnya dan nilai *lnsum*.
11. Konversi nilai *weights* ke dalam probabilitas dengan melakukan perulangan terhadap *weights*.
12. Hitung nilai *weights* dari nilai *exponent weights* sebelumya hingga iterasi selesai dan mendapatkan nilai probabilitas *weights*.
13. Jika *weights* ≠ *null* maka hitung nilai *mean* dan *variance* dari observasi dan *weights*.
14. Melakukan iterasi hingga didapatkan nilai re-estimasi matriks emisi.
15. Proses re-estimasi matriks emisi selesai.

Diagram alir proses re-estimasi matriks emisi dapat digambarkan pada Gambar 3.23 berikut ini :

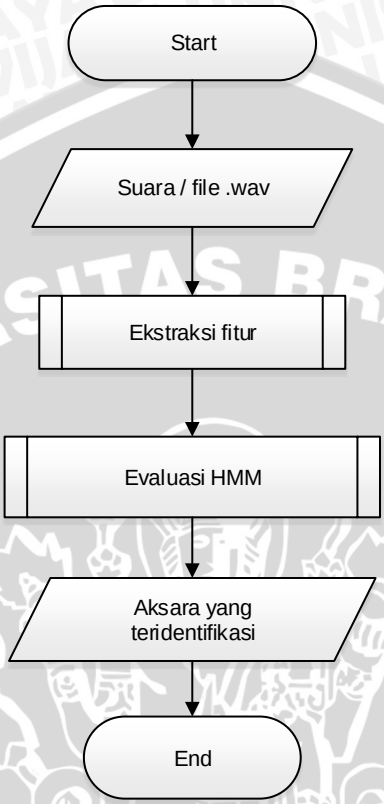

Estimasi matriks emisi
Start
Matriks gamma, matriks emisi, w
i=0; i<states; i++
lnsum = -infinity
k=0, w=0; k<N; k++
t=0; t<T; t++,w++
Weights[w] = logGamma[k][t,i]
lnsum = logsum(lnsum, weights[w])
t
k
lnsum != -infinity
True
False
w=0; w<weights.Length; w++
weights[w] = weights[w] - lnsum
w
c
a



Gambar 3.23 Diagram Alir Re-estimasi Matriks Emisi

3.2.2.5 Perancangan Proses Pengenalan Aksara

Setelah proses pelatihan, maka proses pengenalan aksara dapat dilakukan. Gambar 3.24 adalah diagram alir dari proses pengenalan.



Gambar 3.24 Diagram Alir Proses Pengenalan Aksara

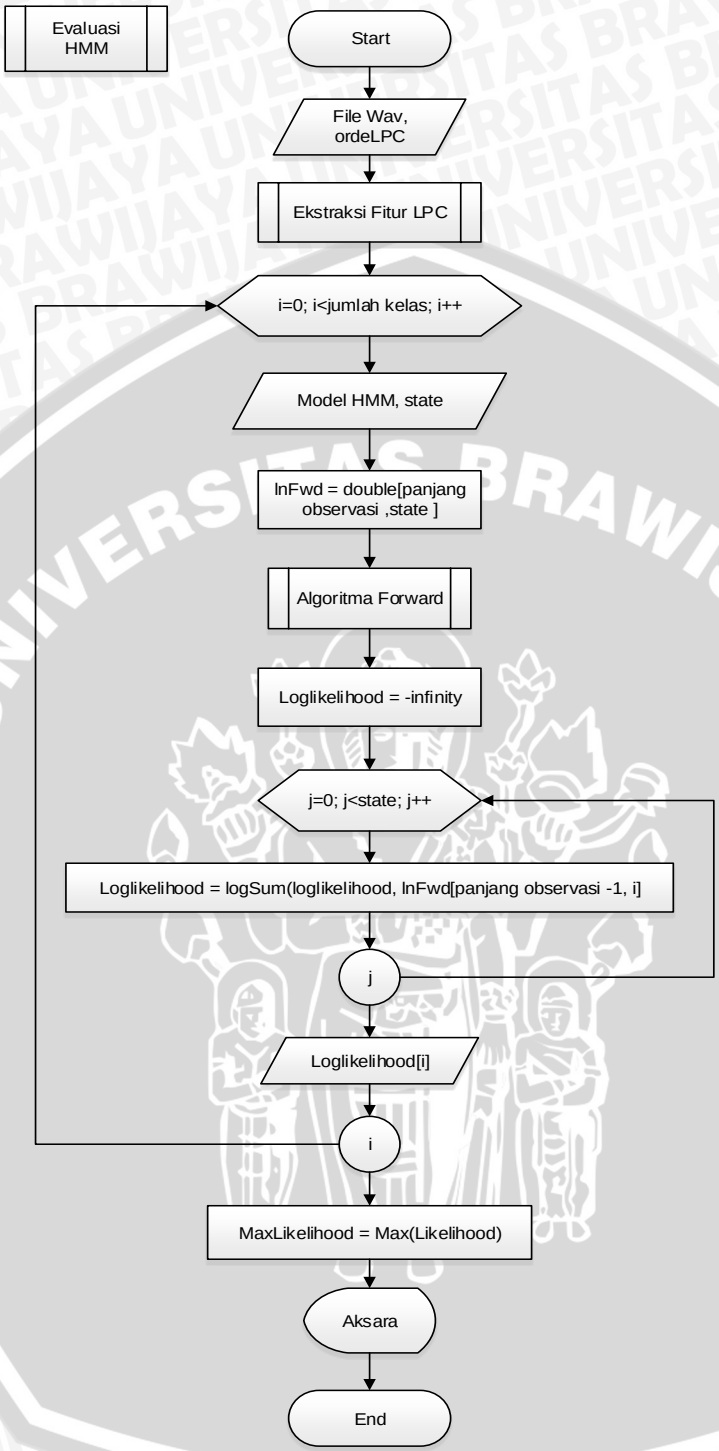
Pada proses pengenalan, masukan berupa suara/rekaman suara akan diekstraksi fiturnya dengan menggunakan *Linear Predictive Coding.* Hasil dari ekstraksi fitur akan dievaluasi menggunakan HMM untuk membandingkan dan mencocokkan pola dari data uji dengan model yang sudah dibentuk. Proses evaluasi HMM akan menghasilkan kelas referensi yang sesuai dengan data uji yang diinputkan. Aksara yang dikenali akan ditampilkan sesuai dengan kelas referensi yang dihasilkan.

a) Perancangan Evaluasi HMM

Langkah-langkah Evaluasi HMM :

1. File Wav *testing* dan orde LPC sebagai masukan.
2. Melakukan proses ekstraksi fitur LPC dari data masukan.
3. Melakukan perulangan untuk jumlah kelas aksara (i).
4. Model hasil metode HMM dan *state* sebagai masukan.
5. Menginisialisasi matriks *forward* (lnFwd).
6. Menghitung nilai matriks *forward* menggunakan algoritma *forward*.
7. Menginisialisasi *loglikelihood = -infinity*.
8. Melakukan perulangan untuk *state* (j).
9. Menghitung nilai *loglikelihood* dengan menjumlahkan *loglikelihood* dan nilai matriks *forward* dengan indeks [panjang observasi -1, i].
10. Melakukan iterasi j hingga selesai dan didapatkan nilai *loglikelihood* data *testing* terhadap model ke-i.
11. Melakukan iterasi i hingga didapatkan *loglikelihood* dari data *testing* terhadap setiap kelas.
12. Mencari *loglikelihood* tertinggi dari data *testing* dan didapatkan aksara dari data *testing* yang diinputkan.

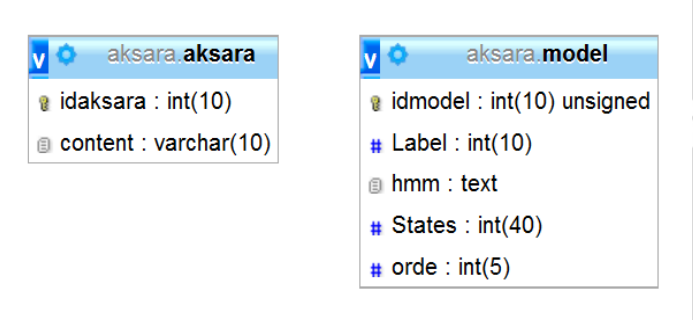
Diagram alir proses evaluasi HMM dapat digambarkan pada Gambar 3.24 berikut ini :



Gambar 3.25 Diagram alir evaluasi HMM

3.2.2.6 Perancangan Database

Perancangan database merupakan perancangan manajemen data yang akan digunakan sistem. Manajemen data termasuk *basis data*, yang mengandung data relevan untuk berbagai situasi dan diatur oleh *software* yang disebut *Database Management System* (DBMS). Sistem ini menggunakan DBMS yaitu MySQL. Terdapat 2 tabel dalam database yang digunakan, yaitu tabel aksara untuk menyimpan data *master* dari aksara dan tabel model untuk menyimpan hasil ekstraksi fitur dari data *training*. Struktur tabel yang akan digunakan untuk sistem pengenalan suara pada Gambar 3.25 sebagai berikut :

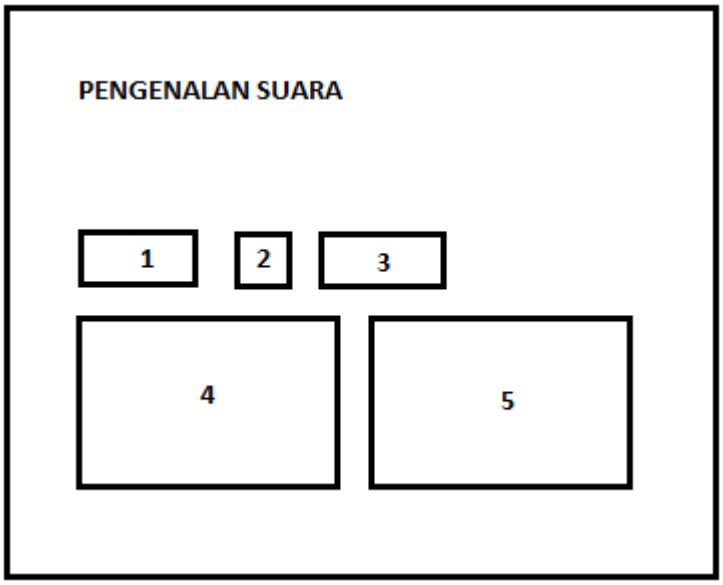


Gambar 3.26 Struktur tabel database

3.2.2.7 Perancangan Interface

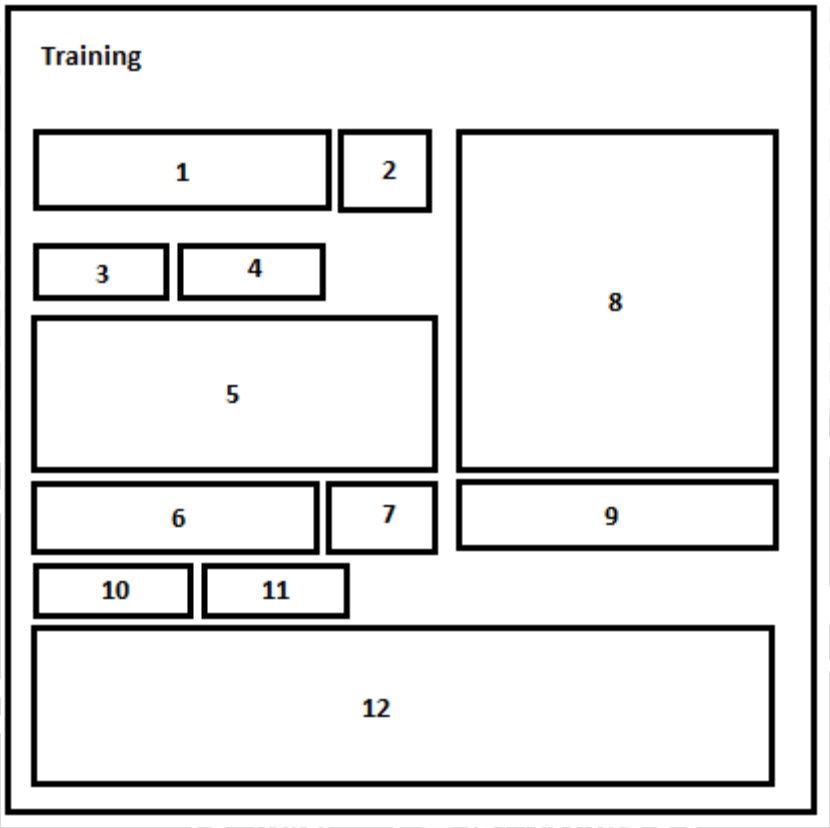
Tahapan implementasi algoritma merupakan tahap implementasi dari metode atau algoritma yang telah dirancang ke dalam suatu bahasa pemrograman. Tahapan ini akan menghasilkan suatu program aplikasi sebagai media representative terhadap hasil dari metode yang diusulkan. Terdapat dua hal dalam tahapan ini yaitu pembuatan kode program yang diusulkan dan pembuatan *interface* program sebagai sarana interaksi sistem dengan pengguna. Berikut ini perancangan *interface* yang akan digunakan dalam sistem pengenalan suara.

Pada Gambar 3.26 terdapat 5 item yaitu form orde LPC (1), tombol set orde LPC (2), tombol proses *training* (3), tombol *training* (4), dan tombol *testing* (5). Fungsi form orde LPC untuk memilih orde LPC yang digunakan, tombol proses *training* untuk melakukan proses *training* dengan orde LPC yang dipilih, tombol *training* untuk menampilkan program *training*, dan tombol *testing* untuk menampilkan program *testing*.



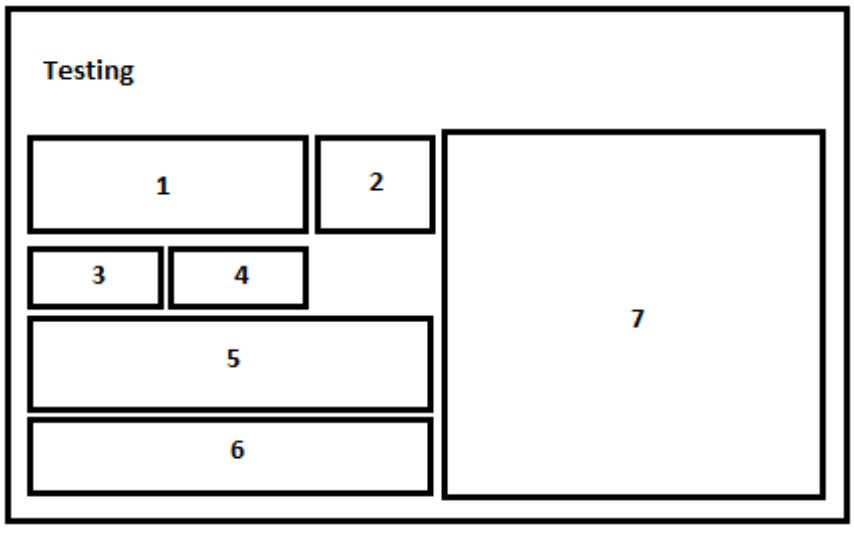
Gambar 3.27 Tampilan awal program

Pada Gambar 3.27 terdapat 12 item yang dapat digunakan. Form (1) untuk menampilkan *device* yang dapat dipilih untuk merekam. Tombol rekam (2) untuk memulai perekaman. Tombol *open* (3) untuk memilih dan membuka file wav. Tombol *play* (4) untuk memainkan file wav yang telah dipilih. Form (5) untuk menampilkan properties file wav. Form (6) untuk memilih aksara. Tombol ekstraksi (7) untuk melakukan proses ekstraksi fitur. Form (8) untuk menampilkan hasil ekstraksi fitur. Tombol *insert* (9) untuk memasukkan data ke dalam *database.* Tombol *refresh* (10) untuk me-*load* ulang *database.* Tombol *training* (11) untuk melakukan proses *training.* Form (12) untuk menampilkan data *training* dalam *database*.



Gambar 3.28 Tampilan form *training*

Pada Gambar 3.28 terdapat 7 item yang dapat digunakan. Form (1) untuk menampilkan *device* yang dapat dipilih untuk merekam. Tombol rekam (2) untuk memulai perekaman. Tombol *open* (3) untuk memilih dan membuka file wav. Tombol *play* (4) untuk memainkan file wav yang telah dipilih. Form (5) untuk menampilkan properties file wav. Tombol *test* (6) untuk melakukan proses *testing* data yang telah dipilih. Form (7) untuk menampilkan hasil *testing* pengenalan.



Gambar 3.29 Tampilan form *testing*

3.2.3 Perhitungan Manual

Pada subbab ini akan membahas mengenai contoh perhitungan manual untuk sistem pengenalan suara menggunakan *Linear Predictive Coding* (LPC) dan *Hidden Markov Model* (HMM). Contoh manualisasi tiap proses dari sistem ini adalah sebagai berikut:

3.2.3.1 Sampling file input

Pada tahap awal akan dilakukan proses *sampling* dari file yang akan di analisis oleh sistem. Misalkan perintah suara yang digunakan adalah kata *"ha"*. Sinyal suara di*sampling* sebanyak 12 sampel. Berikut merupakan sampel dari file input :

Tabel 3.7 Tabel hasil *sampling*

fx0	fx1	fx2	fx3	fx4	fx5	fx6	fx7	fx8	fx9	fx10	fx11
5	2	10	5	0	20	150	5	10	15	100	80

3.2.3.2 Preemphasis

Sampel dari file yang telah diinput kemudian akan melalui tahap *preemphasis*, yaitu konversi sinyal ucapan menjadi sinyal digital menggunakan persamaan 2.1. Berikut beberapa contoh perhitungan *preemphasis* indeks ke-0 dan ke-1 :

$s(0) = fx(0) = 5$

$s(1) = fx(1) - 0.95 * fx(0)$

$= 2 - 0.9375 * 5$

$= -2.6875$

Untuk indeks yang lainnya, perhitungannya sama dengan diatas hanya nilai tiap indeksnya diganti. Data sampel hasil *preemphasis* ditunjukkan pada tabel.

Tabel 3.8 Tabel hasil *preemphasis*

s(0)	5
s(1)	-2.6875
s(2)	8.125
s(3)	-4.375
s(4)	-4.6875
s(5)	20
s(6)	131.25
s(7)	-135.625
s(8)	5.3125
s(9)	5.625
s(10)	85.9375
s(11)	-13.75

3.2.3.3 Frame Blocking

Proses selanjutnya yaitu membagi hasil *preemphasis s*(n) ke dalam *frame-frame* yang masing-masing memuat *N* buah sampel yang dipisahkan sejauh *M* buah sampel. Untuk menghitung jumlah frame/segmen yg dihasilkan yaitu (Panjang

Sinyal – (*N*-*M*))/*M*. *N* dan *M* yang dipakai dalam contoh perhitungan adalah *N* = 9 sedangkan *M* = 3. Berikut merupakan hasil dari *frame blocking* :

$$jumlah\ segmen = (Panjang\ sinyal - (N - M))/M$$

$$jumlah\ segmen = \frac{12 - (9 - 3)}{3} = \frac{12 - 6}{3} = \frac{6}{3} = 2$$

Tabel 3.9 Tabel hasil *frame blocking*

Indeks	Segmen-0	Segmen-1
0	5	-4.375
1	-2.6875	-4.6875
2	8.125	20
3	-4.375	131.25
4	-4.6875	-135.625
5	20	5.3125
6	131.25	5.625
7	-135.625	85.9375
8	5.3125	-13.75

3.2.3.4 Windowing

Proses selanjutnya yaitu melakukan proses penjendelaan pada setiap bagian sampel sinyal yang telah dibuat sebelumnya dengan persamaan 2.2 dan 2.3. Berikut contoh perhitungan *windowing* untuk segmen-0 dan indeks (n) = 0 :

$$w(0) = 0.54 - 0.46\cos(2 * \pi * n/N - 1)$$

$$w(0) = 0.54 - 0.46\cos(2 * 180 * 0/9 - 1)$$

$$= 0.54 - 0.46\cos(0)$$

$$= 0.54 - 0.46 * 1$$

$$= 0.54 - 0.46 = 0.08$$

$$x(0) = x(0) * w(0)$$

$$= 5 * 0.08 = 0.4$$

Untuk indeks yang lainnya, perhitungannya sama dengan diatas hanya nilai tiap indeksnya diganti. Data sampel hasil *windowing* ditunjukkan pada tabel.

Tabel 3.10 Tabel hasil *windowing*

Indeks	w	Segmen-0	Segmen-1
0	0.08	0.4	-0.35
1	0.214731	-0.57709	-1.00655
2	0.54	4.3875	10.8
3	0.865269	-3.78555	113.5666
4	1	-4.6875	-135.625
5	0.865269	17.30538	4.596742
6	0.54	70.875	3.0375
7	0.214731	-29.1229	18.45344
8	0.08	0.425	-1.1

3.2.3.5 Autokorelasi

Proses selanjutnya yaitu menghitung autokorelasi pada setiap segmen sesuai dengan nilai orde LPC yang dipilih menggunakan persamaan 2.4. Orde LPC (p) yang digunakan dalam contoh perhitungan adalah 8, maka setiap segmen akan menghasilkan p+1 hasil autokorelasi. Berikut contoh perhitungan autokorelasi untuk segmen ke-0 :

Segmen-0, p = 0 :

$$ak(0) = \big(x(0) * x(0)\big) + \big(x(1) * x(1)\big) + \big(x(2) * x(2)\big) + \big(x(3) * x(3)\big) + (x(4) * x(4)) + \big(x(5) * x(5)\big) + \big(x(6) * x(6)\big) + \big(x(7) * x(7)\big) + (x(8) * x(8))$$

$$= (0.4 * 0.4) + (-0.5771 * -0.5771) + (4.3875 * 4.3875) + (-3.7856 * -3.7856) + (-4.6875 * -4.6875) + (17.3054 * 17.3054) + (70.875 * 70.875) + (-29.123 * -29.123) + (0.425 * 0.425)$$

$$= (0.16) + (0.3330) + (19.2502) + (14.3304) + (21.9727) + (299.4763) + (5023.2656) + (848.1419) + (0.1806)$$

$$ak(0) = 6227.1106$$

Segmen-0, p = 1 :

$$ak(1) = \big(x(0) * x(1)\big) + \big(x(1) * x(2)\big) + \big(x(2) * x(3)\big) + \big(x(3) * x(4)\big) + \big(x(4) * x(5)\big) + \big(x(5) * x(6)\big) + \big(x(6) * x(7)\big) + \big(x(7) * x(8)\big)$$

$$= (0.4 * -0.5771) + (-0.5771 * 4.3875) + (4.3875 * -3.7856) + (-3.7856 * -4.6875) + (-4.6875 * 17.3054) + (17.3054 * 70.875) + (70.875 * -29.123) + (-29.123 * 0.425)$$

$$= (-0.2308) + (-2.5320) + (-16.6091) + (17.7448) + (-81.1190) + (1226.5190) + (-2064.0838) + (-12.3772)$$

$$ak(1) = -932.6882$$

Segmen-0, p = 2 :

$$ak(2) = \big(x(0) * x(2)\big) + \big(x(1) * x(3)\big) + \big(x(2) * x(4)\big) + \big(x(3) * x(5)\big) + \big(x(4) * x(6)\big) + \big(x(5) * x(7)\big) + \big(x(6) * x(8)\big)$$

$$ak(0) = (0.4 * 4.3875) + (-0.5771 * -3.7856) + (4.3875 * -4.6875) + (-3.7856 * 17.3054) + (-4.6875 * 70.875) + (17.3054 * -29.123) + (70.875 * 0.425)$$

$$ak(0) = (1.755) + (2.1846) + (-20.5664) + (-65.5104) + (-332.2266) + (-503.9825) + (30.1219)$$

$$ak(2) = -888.2244$$

Segmen-0, p = 3 :

$$ak(3) = \big(x(0) * x(3)\big) + \big(x(1) * x(4)\big) + \big(x(2) * x(5)\big) + \big(x(3) * x(6)\big) + \big(x(4) * x(7)\big) + \big(x(5) * x(8)\big)$$

$$= (0.4 * -3.7856) + (-0.5771 * -4.6875) + (4.3875 * 17.3054) + (-3.7856 * 70.875) + (-4.6875 * -29.123) + (17.3054 * 0.425)$$

$$= (-1.5142) + (2.7051) + (75.9274) + (-268.3010) + (136.5135) + (7.3548)$$

$$ak(3) = -47.3145$$

Segmen-0, p = 4 :

$$ak(4) = \big(x(0) * x(4)\big) + \big(x(1) * x(5)\big) + \big(x(2) * x(6)\big) + \big(x(3) * x(7)\big) + \big(x(4) * x(8)\big)$$

$$= (0.4 * -4.6875) + (-0.5771 * 17.3054) + (4.3875 * 70.875) + (-3.7856 * -29.123) + (-4.6875 * 0.425)$$

$$= (-1.8750) + (-9.9868) + (310.9641) + (110.2462) + (-1.9922)$$

$$ak(4) = 407.3563$$

Segmen-0, p = 5 :

$$ak(5) = \big(x(0) * x(5)\big) + \big(x(1) * x(6)\big) + \big(x(2) * x(7)\big) + \big(x(3) * x(8)\big)$$

$$= (0.4 * 17.3054) + (-0.5771 * 70.875) + (4.3875 * -29.123) + (-3.7856 * 0.425)$$

$$ak(0) = (6.9222) + (-40.9012) + (-127.7766) + (-1.6098)$$

$$ak(5) = -163.3645$$

Segmen-0, p = 6 :

$$ak(6) = \big(x(0) * x(6)\big) + \big(x(1) * x(7)\big) + \big(x(2) * x(8)\big)$$

$$ak(0) = (0.4 * 70.875) + (-0.5771 * -29.123) + (4.3875 * 0.425)$$

$$ak(0) = (28.3500) + (16.8065) + (1.8647)$$

$$ak(6) = 47.0212$$

Segmen-0, p = 7 :

$$ak(7) = \big(x(0) * x(7)\big) + \big(x(1) * x(8)\big)$$

$$= (0.4 * -29.123) + (-0.5771 * 0.425)$$

$$= (-11.6492) + (-0.2453)$$

$$ak(7) = -11.8944$$

Segmen-0, p = 8 :

$$ak(8) = \big(x(0) * x(8)\big)$$

$$= (0.4 * 0.425)$$

$$= (0.1700)$$

$$ak(8) = 0.1700$$

Untuk segmen yang lainnya, perhitungannya sama dengan diatas hanya sampel yang digunakan adalah sampel dalam segmen tersebut. Data sampel hasil autokorelasi ditunjukkan pada tabel.

Tabel 3.11 Tabel hasil autokorelasi

Indeks (p)	Segmen-0	Segmen-1
0	6227.11065	31781.3782
1	-932.688189	-14760.1828
2	-888.2244	-1391.2808
3	-47.3145	-2016.4351
4	407.3563	2320.5278
5	-163.3645	69.7076
6	47.0212	-31.5174
7	-11.8944	-5.3515
8	0.1700	0.385

3.2.3.6 Analisis LPC

Proses terakhir dalam LPC yaitu mengubah hasil autokorelasi pada masing-masing *frame* menjadi koefisien LPC dengan algoritma Levinson-Durbin pada persamaan 2.5 – 2.9. Berikut contoh perhitungan analisis LPC untuk segmen ke-0 :

Segmen-0, untuk iterasi p = 0 maka k[0] = ak[0,1]/ak[0,0] :

$e[0] = ak[0,0] = 6227.1106$

$$k[0] = \frac{ak[0,1]}{ak[0,0]} = \frac{-932.6881}{6227.1106} = -0.1498$$

$\alpha[0,0] = k[0] = -0.1498$

$e[1] = (1 - k[0]^2) * e[0] = (1 - (-0.1498)^2) * 6227.1106 = 6087.4139$

Segmen-0, p = 1, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 0 : $tmp = 0$

$tmp = tmp + (\alpha[p-1, m] * ak[0, |p-m|])$

$= 0 + (\alpha[0,0] * ak[0, |1-0|])$

$$= 0 + (\alpha[0,0] * ak[0,1])$$

$$= 0 + (-0.1498 * -932.6881)$$

$$tmp = 0 + 139.6968 = 139.6968$$

$$k[1] = \frac{ak[0,p+1] - tmp}{e[p]} = \frac{ak[0,1+1] - 139.6968}{e[1]}$$

$$= \frac{ak[0,2] - 139.6968}{e[1]}$$

$$= \frac{-888.2244 - 139.6968}{6087.4139}$$

$$= \frac{-1027.9212}{6087.4139}$$

$$k[1] = -0.1688$$

$\alpha[1,1] = k[1] = -0.1688$

Iterasi untuk menghitung nilai $\alpha[1,0]$

l = p-1 = 1-1 = 0 hingga l >= 0

l = 0 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[1,0] = \alpha[1-1,0] - (k[1] * \alpha[1-1,(1-1)-0])$

$= \alpha[0,0] - (k[1] * \alpha[0,0])$

$= -0.1498 - (-0.1689 * -0.1498)$

$= -0.1498 - (0.0253)$

$= -0.1751$

$e[2] = (1 - k[1]^2) * e[1] = (1 - (-0.1688)^2) * 6087.4139 = 5913.8390$

Segmen-0, p = 2, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 1: $tmp = 0$

$$tmp = tmp + (\alpha[p-1,m] * ak[0,|p-m|]) + (\alpha[p-1,m+1] * ak[0,|p-(m+1)|])$$

$$= 0 + (\alpha[2-1,0] * ak[0,|2-0|]) + (\alpha[2-1,1] * ak[0,|2-1|])$$

$$= 0 + (\alpha[1,0] * ak[0,2]) + (\alpha[1,1] * ak[0,1])$$

$$= 0 + (-0.1751 * -888.2244) + (-0.1688 * -932.6881)$$

$$tmp = 0 + 155.5281 + 157.4377 = 312.9658$$

$$k[2] = \frac{ak[0,p+1] - tmp}{e[p]} = \frac{ak[0,2+1] - 312.9658}{e[2]}$$

$$= \frac{ak[0,3] - 312.9658}{e[2]}$$

$$= \frac{-47.3145 - 312.9658}{5913.8390}$$

$$= \frac{-360.2803}{5913.8390}$$

$$k[2] = -0.06092$$

$\alpha[2,2] = k[2] = -0.06092$

Iterasi untuk menghitung nilai $\alpha[2,1]$ dan $\alpha[2,0]$

l = p-1 = 2-1 = 1 hingga l >= 0

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[2,1] = \alpha[2-1,1] - (k[2] * \alpha[2-1,(2-1)-1])$$

$$= \alpha[1,1] - (k[2] * \alpha[1,0])$$

$$= -0.1688 - (-0.06092 * -0.1751)$$

$$= -0.1688 - (0.0106)$$

$$= -0.1794$$

l = 0 : $\alpha[p,l] = \alpha[p-1,0] - (k[p] * \alpha[p-1,(p-1)-0])$

$$\alpha[2,0] = \alpha[2-1,0] - (k[2] * \alpha[2-1,(2-1)-0])$$

$$= \alpha[1,0] - (k[2] * \alpha[1,1])$$

$$= -0.1751 - (-0.06092 * -0.1688)$$
$$= -0.1751 - (0.0102)$$
$$= -0.1853$$

$e[3] = (1 - k[2]^2) * e[2] = (1 - (-0.06092)^2) * 5913.8390 = 5891.8912$

Segmen-0, p = 3, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 2: $tmp = 0$

$$tmp = tmp + (\alpha[p-1, m] * ak[0, |p-m|]) + (\alpha[p-1, m+1] * ak[0, |p-(m+1)|]) + (\alpha[p-1, m+2] * ak[0, |p-(m+2)|])$$
$$= 0 + (\alpha[3-1, 0] * ak[0, |3-0|]) + (\alpha[3-1, 1] * ak[0, |3-1|]) + (\alpha[3-1, 2] * ak[0, |3-2|])$$
$$= 0 + (\alpha[2, 0] * ak[0,3]) + (\alpha[2, 1] * ak[0,2]) + (\alpha[2, 2] * ak[0,1])$$
$$= 0 + (-0.1853 * -47.3145) + (-0.1795 * -888.2244) + (-0.0609 * -932.6881)$$
$$tmp = 0 + 8.7673 + 159.4362 + 56.8007$$
$$tmp = 225.0042$$
$$k[3] = \frac{ak[0, p+1] - tmp}{e[p]} = \frac{ak[0, 3+1] - 225.0042}{e[3]}$$
$$= \frac{ak[0,4] - 225.0042}{e[3]}$$
$$= \frac{407.3562 - 225.0042}{5891.8912}$$
$$= \frac{182.3520}{5891.8912}$$
$$k[3] = 0.0309$$

$\alpha[3,3] = k[3] = 0.0309$

Iterasi untuk menghitung nilai $\alpha[3,2], \alpha[3,1]\ dan\ \alpha[3,0]$

l = p-1 = 3-1 = 2 hingga l >= 0

l = 2 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[3,2] = \alpha[3-1,2] - (k[3] * \alpha[3-1,(3-1)-2])$

$= \alpha[2,2] - (k[3] * \alpha[2,0])$

$= -0.0609 - (0.0309 * -0.1853)$

$= -0.0609 - (-0.0057)$

$= -0.0552$

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[3,1] = \alpha[3-1,1] - (k[3] * \alpha[3-1,(3-1)-1])$

$= \alpha[2,1] - (k[3] * \alpha[2,1])$

$= -0.1794 - (0.0309 * -0.1794)$

$= -0.1794 - (-0.0055)$

$= -0.1739$

l = 0 : $\alpha[p,l] = \alpha[p-1,0] - (k[p] * \alpha[p-1,(p-1)-0])$

$\alpha[3,0] = \alpha[3-1,0] - (k[3] * \alpha[3-1,(3-1)-0])$

$= \alpha[2,0] - (k[3] * \alpha[2,2])$

$= -0.1853 - (0.0309 * -0.0609)$

$= -0.1853 - (-0.0018)$

$= -0.1835$

$e[4] = (1 - k[3]^2) * e[3] = (1 - (0.0309)^2) * 5891.8912 = 5886.2655$

Segmen-0, p = 4, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 3: $tmp = 0$

$tmp = 0 + (\alpha[4-1,0] * ak[0,|4-0|])$

$+ (\alpha[4-1,1] * ak[0,|4-1|])$

$+ (\alpha[4-1,2] * ak[0,|4-2|])$

$+ (\alpha[4-1,3] * ak[0,|4-3|])$

$= 0 + (\alpha[3,0] * ak[0,4]) + (\alpha[3,1] * ak[0,3])$

$+ (\alpha[3,2] * ak[0,2]) + (\alpha[3,3] * ak[0,1])$

$$= 0 + (-0.1835 * 407.3562) + (-0.1739 * -47.3145) + (-0.0552 * -888.2244) + (0.0309 * -932.6881)$$

$$tmp = 0 + -74.7498 + 8.2279 + 49.0299 + -28.82$$

$$tmp = -46.312$$

$$k[4] = \frac{ak[0, p+1] - tmp}{e[p]} = \frac{ak[0,5] - (-46.312)}{e[4]}$$

$$= \frac{-163.3645 - (-46.312)}{5886.2655}$$

$$= \frac{-117.0525}{5886.2655}$$

$$k[4] = -0.0198$$

$$\alpha[4,4] = k[4] = -0.0198$$

Iterasi untuk menghitung nilai $\alpha[4,3], \alpha[4,2], \alpha[4,1]\ dan\ \alpha[4,0]$

l = p-1 = 4-1 = 3 hingga l >= 0

l = 3 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[4,3] = \alpha[4-1,3] - (k[4] * \alpha[4-1,(4-1)-3])$

$= \alpha[3,3] - (k[4] * \alpha[3,0])$

$= 0.0309 - (-0.0198 * -0.1835)$

$= 0.0273$

l = 2 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[4,2] = \alpha[4-1,2] - (k[4] * \alpha[4-1,(4-1)-2])$

$= \alpha[3,2] - (k[4] * \alpha[3,1])$

$= -0.0552 - (-0.0198 * -0.1739)$

$= -0.0586$

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[4,1] = \alpha[4-1,1] - (k[4] * \alpha[4-1,(4-1)-1])$

$= \alpha[3,1] - (k[4] * \alpha[3,2])$

$= -0.1739 - (-0.0198 * -0.0552)$

$= -0.1750$

l = 0 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[4,0] = \alpha[4-1,0] - (k[4] * \alpha[4-1,(4-1)-0])$

$= \alpha[3,0] - (k[4] * \alpha[3,3])$

$= -0.1835 - (-0.0198 * 0.0309)$

$= -0.1828$

$e[5] = (1 - k[4]^2) * e[4] = (1 - (-0.0198)^2) * 5886.2655 = 5883.9578$

Segmen-0, p = 5, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 4: $tmp = 0$

$tmp = 0 + (\alpha[5-1,0] * ak[0,|5-0|])$
$+ (\alpha[5-1,1] * ak[0,|5-1|])$
$+ (\alpha[5-1,2] * ak[0,|5-2|])$
$+ (\alpha[5-1,3] * ak[0,|5-3|])$
$+ (\alpha[5-1,4] * ak[0,|5-4|])$

$= 0 + (\alpha[4,0] * ak[0,5]) + (\alpha[4,1] * ak[0,4])$
$+ (\alpha[4,2] * ak[0,3]) + (\alpha[4,3] * ak[0,2])$
$+ (\alpha[4,4] * ak[0,1])$

$= 0 + (-0.1828 * -163.3645)$
$+ (-0.1750 * 407.3562)$
$+ (-0.0586 * -47.3145)$
$+ (0.0272 * -888.2244)$
$+ (-0.0198 * -932.6881)$

$tmp = 0 + 29.8630 + -71.2873 + 2.7726 + -24.1597$
$+ 18.4672$

$tmp = -44.3442$

$k[5] = \frac{ak[0,p+1] - tmp}{e[p]} = \frac{ak[0,6] - (-44.3442)}{e[5]}$

$= \frac{47.0211 - (-44.3442)}{5883.9578}$

$= \frac{91.3653}{5883.9578}$

$k[5] = 0.0155$

$\alpha[5,5] = k[5] = 0.0155$

Iterasi untuk menghitung nilai $\alpha[5,4], \alpha[5,3], \alpha[5,2], \alpha[5,1]\ dan\ \alpha[5,0]$

l = p-1 = 5-1 = 4 hingga l >= 0

l = 4 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[5,4] = \alpha[4-1,4] - (k[5] * \alpha[5-1,(5-1)-4])$

$= \alpha[4,4] - (k[5] * \alpha[4,0])$

$= -0.0198 - (0.0155 * -0.1828)$

$= -0.017$

l = 3 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[5,3] = \alpha[5-1,3] - (k[5] * \alpha[5-1,(5-1)-3])$

$= \alpha[4,3] - (k[5] * \alpha[4,1])$

$= 0.0273 - (0.0155 * -0.1750)$

$= 0.0300$

l = 2 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[5,2] = \alpha[5-1,2] - (k[5] * \alpha[5-1,(5-1)-2])$

$= \alpha[4,2] - (k[5] * \alpha[4,2])$

$= -0.0586 - (0.0155 * -0.0586)$

$= -0.0576$

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[5,1] = \alpha[5-1,1] - (k[5] * \alpha[5-1,(5-1)-1])$

$= \alpha[4,1] - (k[5] * \alpha[4,3])$

$= -0.1750 - (0.0155 * 0.0273)$

$= -0.1754$

l = 0 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[5,0] = \alpha[5-1,0] - (k[5] * \alpha[5-1,(5-1)-0])$

$= \alpha[4,0] - (k[5] * \alpha[4,4])$

$= -0.1828 - (0.0155 * -0.0198)$

$= -0.1824$

$e[6] = (1 - k[5]^2) * e[5] = (1 - (0.0155)^2) * 5883.9578 = 5882.5441$

Segmen-0, p = 6, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 5: $tmp = 0$

$$tmp = 0 + (\alpha[6-1,0] * ak[0,|6-0|]) + (\alpha[6-1,1] * ak[0,|6-1|]) + (\alpha[6-1,2] * ak[0,|6-2|]) + (\alpha[6-1,3] * ak[0,|6-3|]) + (\alpha[6-1,4] * ak[0,|6-4|]) + (\alpha[6-1,5] * ak[0,|6-5|])$$

$$= 0 + (\alpha[5,0] * ak[0,6]) + (\alpha[5,1] * ak[0,5]) + (\alpha[5,2] * ak[0,4]) + (\alpha[5,3] * ak[0,3]) + (\alpha[5,4] * ak[0,2]) + (\alpha[5,5] * ak[0,1])$$

$$= 0 + (-0.1824 * 47.0211) + (-0.1754 * -163.3645) + (-0.0576 * 407.3562) + (0.0300 * -47.3145) + (-0.0170 * -888.2244) + (0.0155 * -932.6881)$$

$$tmp = 0 + -8.5766 + 28.6541 + -23.4637 + -1.4194 + 15.0998 + -14.4566$$

$$tmp = -4.1624$$

$$k[6] = \frac{ak[0,p+1] - tmp}{e[p]} = \frac{ak[0,7] - (-4.1624)}{e[6]}$$

$$= \frac{-11.8944 - (-4.1624)}{5882.5441}$$

$$= \frac{-7.732}{5882.5441}$$

$$k[6] = -0.0013$$

$[6,6] = k[6] = -0.0013$

Iterasi untuk menghitung nilai $\alpha[6,5], \alpha[6,4], \alpha[6,3], \alpha[6,2], \alpha[6,1]\ dan\ \alpha[6,0]$

l = p-1 = 6-1 = 5 hingga l >= 0

l = 5 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,5] = \alpha[6-1,5] - (k[6] * \alpha[6-1,(6-1)-5])$$
$$= \alpha[5,5] - (k[6] * \alpha[5,0])$$
$$= 0.0155 - (-0.0013 * -0.1824)$$
$$= 0.0152$$

l = 4 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,4] = \alpha[6-1,4] - (k[6] * \alpha[6-1,(6-1)-4])$$
$$= \alpha[5,4] - (k[6] * \alpha[5,1])$$
$$= -0.017 - (-0.0013 * -0.1754)$$
$$= -0.0172$$

l = 3 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,3] = \alpha[6-1,3] - (k[6] * \alpha[6-1,(6-1)-3])$$
$$= \alpha[5,3] - (k[6] * \alpha[5,2])$$
$$= 0.0300 - (-0.0013 * -0.0576)$$
$$= 0.0299$$

l = 2 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,2] = \alpha[6-1,2] - (k[6] * \alpha[6-1,(6-1)-2])$$
$$= \alpha[5,2] - (k[6] * \alpha[5,3])$$
$$= -0.0576 - (-0.0013 * 0.0300)$$
$$= -0.0575$$

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,1] = \alpha[6-1,1] - (k[6] * \alpha[6-1,(6-1)-1])$$
$$= \alpha[5,1] - (k[6] * \alpha[5,4])$$
$$= -0.1754 - (-0.0013 * -0.017)$$
$$= -0.1754$$

l = 0 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,0] = \alpha[6-1,0] - (k[6] * \alpha[6-1,(6-1)-0])$$
$$= \alpha[5,0] - (k[6] * \alpha[5,5])$$
$$= -0.1824 - (-0.0013 * 0.0155)$$
$$= -0.1824$$

$$e[7] = (1 - k[6]^2) * e[6] = (1 - (-0.0013)^2) * 5882.5441 = 5882.5341$$

Segmen-0, p = 7, untuk iterasi pada persamaan 2.6 :

m = 0 s/d 6: $tmp = 0$

$$tmp = 0 + (\alpha[7-1,0] * ak[0,|7-0|]) + (\alpha[7-1,1] * ak[0,|7-1|]) + (\alpha[7-1,2] * ak[0,|7-2|]) + (\alpha[7-1,3] * ak[0,|7-3|]) + (\alpha[7-1,4] * ak[0,|7-4|]) + (\alpha[7-1,5] * ak[0,|7-5|]) + (\alpha[7-1,6] * ak[0,|7-6|])$$

$$= 0 + (\alpha[6,0] * ak[0,7]) + (\alpha[6,1] * ak[0,6]) + (\alpha[6,2] * ak[0,5]) + (\alpha[5,3] * ak[0,4]) + (\alpha[6,4] * ak[0,3]) + (\alpha[5,5] * ak[0,2]) + (\alpha[6,5] * ak[0,1])$$

$$= 0 + (-0.1824 * -11.8944) + (-0.1754 * 47.0211) + (-0.0575 * -163.3645) + (0.0299 * 407.3562) + (-0.0172 * -47.3145) + (0.0152 * -888.2244) + (-0.0013 * -932.6881)$$

$$tmp = 0 + 2.1695 + -8.2475 + 9.3934 + 12.1799 + 0.8138 + -13.501 + 1.2124$$

$$tmp = 4.0205$$

$$k[7] = \frac{ak[0,p+1] - tmp}{e[p]} = \frac{ak[0,8] - (4.0205)}{e[7]}$$

$$= \frac{0.17 - 4.0205}{5882.5341}$$

$$= \frac{-3.8505}{5882.5341}$$

$$k[7] = -0.0006$$

$$\alpha[7,7] = k[7] = -0.0006$$

Iterasi untuk menghitung

$\alpha[7,6], \alpha[7,5], \alpha[7,4], \alpha[7,3], \alpha[7,2], \alpha[7,1]\ dan\ \alpha[7,0]$

l = p-1 = 7-1 = 6 hingga l >= 0

l = 6 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,6] = \alpha[7-1,6] - (k[7] * \alpha[7-1,(7-1)-6])$

$= \alpha[6,6] - (k[7] * \alpha[6,0])$

$= -0.0013 - (-0.0006 * -0.1824)$

$= -0.0014$

l = 5 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,5] = \alpha[7-1,5] - (k[7] * \alpha[7-1,(7-1)-5])$

$= \alpha[6,5] - (k[7] * \alpha[6,1])$

$= 0.0152 - (-0.0006 * -0.1754)$

$= 0.0150$

l = 4 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,4] = \alpha[7-1,4] - (k[7] * \alpha[7-1,(7-1)-4])$

$= \alpha[6,4] - (k[7] * \alpha[6,2])$

$= -0.0172 - (-0.0006 * -0.0575)$

$= -0.0172$

l = 3 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,3] = \alpha[7-1,3] - (k[7] * \alpha[7-1,(7-1)-3])$

$= \alpha[6,3] - (k[7] * \alpha[6,3])$

$= 0.0299 - (-0.0006 * 0.0299)$

$= 0.0299$

l = 2 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,2] = \alpha[7-1,2] - (k[7] * \alpha[7-1,(7-1)-2])$

$= \alpha[6,2] - (k[7] * \alpha[6,4])$

$= -0.0575 - (-0.0006 * -0.0172)$

$= -0.0575$

l = 1 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$\alpha[7,1] = \alpha[7-1,1] - (k[7] * \alpha[7-1,(7-1)-1])$

$$= \alpha[6,1] - (k[7] * \alpha[6,5])$$
$$= -0.1754 - (-0.0006 * 0.0152)$$
$$= -0.1754$$

l = 0 : $\alpha[p,l] = \alpha[p-1,l] - (k[p] * \alpha[p-1,(p-1)-l])$

$$\alpha[6,0] = \alpha[7-1,0] - (k[7] * \alpha[7-1,(7-1)-0])$$
$$= \alpha[6,0] - (k[7] * \alpha[6,6])$$
$$= -0.1824 - (-0.0006 * -0.0013)$$
$$= -0.1824$$

Jika array $\alpha[m-1,\dots]$ sudah terbentuk, maka untuk membentuk koefisien LPC dari sampel tersebut yaitu dengan iterasi n < p(ordeLPC) karena m yang digunakan adalah 8 maka batas iterasi yang digunakan adalah n < 8, berikut contoh pembentukan koefisien LPC segmen-0 :

$n = 0$ $koefLPC[segmen,n] = \alpha[p-1,n]$

$koefLPC[0,0] = \alpha[8-1,0]$

$koefLPC[0,0] = \alpha[7,0] = -0.1824$

$n = 1$ $koefLPC[0,1] = \alpha[8-1,1]$

$koefLPC[0,1] = \alpha[7,1] = -0.1754$

$n = 2$ $koefLPC[0,2] = \alpha[8-1,2]$

$koefLPC[0,2] = \alpha[7,2] = -0.0575$

$n = 3$ $koefLPC[0,3] = \alpha[8-1,3]$

$koefLPC[0,3] = \alpha[7,3] = 0.0299$

$n = 4$ $koefLPC[0,4] = \alpha[8-1,4]$

$koefLPC[0,4] = \alpha[7,4] = -0.0172$

$n = 5$ $koefLPC[0,5] = \alpha[8-1,5]$

$koefLPC[0,5] = \alpha[7,5] = 0.0150$

$n = 6$ $koefLPC[0,6] = \alpha[8-1,6]$

$koefLPC[0,6] = \alpha[7,6] = -0.0014$

$n = 7$ $koefLPC[0,7] = \alpha[8-1,7]$

$koefLPC[0,7] = \alpha[7,7] = -0.0006$

Untuk segmen yang lainnya, perhitungannya sama dengan diatas hanya sampel yang digunakan adalah sampel dalam segmen tersebut. Data hasil analisis LPC ditunjukkan pada tabel.

Tabel 3.12 Tabel hasil analisis LPC

Indeks (p)	Segmen-0	Segmen-1
0	-0.1824	-0.9785
1	-0.1754	-1.0049
2	-0.0575	-0.9975
3	0.0299	-0.8333
4	-0.0172	-0.6625
5	0.0150	-0.4743
6	-0.0014	-0.2915
7	-0.0006	-0.1363

3.2.3.7 Proses Parameter *Hidden Markov Model*

Pada proses ini hasil ekstraksi fitur dari file wav akan disimpan dalam database. Selain sequence dari koefisien LPC yang akan disimpan adalah label dan orde. Label yang digunakan ada 20, setiap label mewakili satu aksara. Berikut ini contoh dari proses parameter HMM dengan jumlah *state* 2. Pada Tabel 3.12 adalah contoh data *dummy training*.

Tabel 3.13 Data dummy training

Pola	Label
-0.1824, -0.1754, -0.0575	0

Diketahui :

Pola : sequence data

Label : 0

State : 2

T : 3

Matriks HMM yang dibentuk :

Tabel 3.14 Matriks inisial

	0	1
0	0	-infinity

Tabel 3.15 Matriks transisi

	0	1
0	-0.69314	-0.69314
1	-infinity	0

Tabel 3.16 Matriks emisi

	0	1
0	N(x, mean:0, stddev: 1) lnconstant = -ln((sqrtpi(2)*1)) = -0.9189385	N(x, mean:0, stddev: 1) lnconstant = -0.9189385

a) Menghitung nilai matriks *forward*

Dalam menentukan nilai matriks *forward* maka proses akan dibagi menjadi dua yaitu inisialisasi dan induksi. Contoh perhitungannya adalah sebagai berikut :

a Inisialisasi : Melakukan iterasi i dari 0 hingga *i<state*

lnFwd[0, i] = logPi[i] + logB[i].LogProbabilityFunction(observations[0]);

LogPdf = lnconstant – ((-0.1824-mean)/stddev) x ((-0.1824-mean)/stddev) x 0.5

= -0.9189385 – (-0.1824) x (-0.1824) x 0.5

= -0.93557

lnfwd[0,0] = LogPi[0] + logB[0].LogPdf(-0.1824)

= 0 + -0.93557

= -0.93557

Setelah proses inisialisasi selesai hingga *state*-1, maka proses selanjutnya adalah induksi.

b. Induksi : Melakukan iterasi t=1 hingga t<T kemudian i=0 hingga i<state dan j=0 hingga j < state.

```
double[] x = observations[t]
sum = Double.NegativeInfinity
sum = Special.LogSum(sum, lnFwd[t - 1, j] + logA[j, i])
lnFwd[t, i] = sum + logB[i].LogProbabilityFunction(x)
```

t=0,i=0 maka melakukan perulangan j untuk mendapat nilai sum baru

j=0, sum = logsum(-infinity, lnfwd[0,0]+logA[0,0])

= logsum(-infinity, -0.93557+-0.69314)

sum = -1.62872

j=1, sum = logsum(-1,62872, lnfwd[0,1]+logA[1,0])

= logsum(-1.62872, -infinity+-infinity) = -1,62872

Jika sudah memperoleh nilai sum yang baru maka dapat dihitung nilai matriks *forward* indeks ke t,i

lnfwd[0,0] = sum + logB[i].logPdf(-0.1754)

= -1.62872 + -0.93432

lnfwd[0,0] = -2.56304

Lakukan perhitungan diatas hingga semua perulangan selesai dilakukan dan mendapatkan semua nilai matriks *forward*.

b) Menghitung nilai matriks *backward*

Dalam menentukan nilai matriks *backward* maka proses akan dibagi menjadi dua yaitu inisialisasi dan induksi. Contoh perhitungannya adalah sebagai berikut :

a. Inisialisasi : melakukan iterasi i = 0 sampai i<state

T = panjang observasi = 3

lnbwd[T-1, i] = 0

lnbwd[2, 0] = 0

Setelah proses inisialisasi selesai hingga *state*-1, maka proses selanjutnya adalah induksi.

b. Induksi : Melakukan iterasi t=T-2 hingga t>=0 kemudian i=0 hingga i<state dan j=0 hingga j < state.

```
sum = Double.NegativeInfinity
sum = Special.LogSum(sum, lnBwd[t+1, j] + logA[i, j] +
        logB[j].logpdf(t+1))
lnbwd[t, i] = sum
```

t=1,i=0 maka melakukan perulangan j untuk mendapat nilai sum baru

j=0, sum = logsum(-infinity, lnbwd[2,0]+logA[0,0]+logB[0].logpdf(o[2]= -0.0575)

= logsum(-infinity, 0+-0.69314+(-0.92))

sum = -1.6137

j=1, sum = logsum(-1,6137, lnbwd[2,1]+logA[0,1]+logB[1].logpdf(o[2]= -0.0575)

= logsum(-1.6137, 0 +-0.69314+-0.92) = -0.9206

Jika sudah memperoleh nilai sum yang baru maka dapat dihitung nilai matriks *backward* indeks ke t,i

lnbwd[1,0] = sum

= -0.9206

Lakukan perhitungan diatas hingga semua perulangan selesai dilakukan dan mendapatkan semua nilai matriks *backward*.

c) **Menentukan dan menghitung nilai matriks *gamma***

Matriks *gamma* digunakan pada waktu re-estimasi parameter HMM.

Contoh perhitungan matriks *gamma* adalah sebagai berikut :

Melakukan iterasi dari t=0 hingga t<T, kemudian iterasi k=0 hingga k<*state*

```
lnsum = -infinity
loggamma[t,k] = lnfwd[t,k] + lnbwd[t,k] + w
lnsum = logsum(lnsum, loggamma[t,k]
```

jika t=0, k=0, maka

loggamma[0,0] = lnfwd[0,0] + lnbwd[0,0] + w
= -0.93557 + -1.8549 + 0
= -2.789

lnsum = logsum(lnsum, loggamma[0,0]
= logsum(-infinity, -2.789)
= -2.789

t=0, k=1 maka

loggamma[0,1] = lnfwd[0,1] + lnbwd[0,1] + w
= -infinity + -2.5481 + 0
= -infinity

lnsum = logsum(lnsum, loggamma[0,1]
= logsum(2.789, -infinity)
= -2.789

Setelah ditemukan nilai dari matriks *gamma* maka akan dinormalisasi dengan perulangan i=0 hingga i<*state* sebagai berikut.

i=0, loggamma[0,0] = loggamma[0,0] – lnsum
= -2.789 - -2.789
= 0

Lakukan perhitungan diatas hingga semua perulangan selesai dilakukan dan mendapatkan semua nilai matriks *gamma*.

d) Menentukan dan menghitung nilai matriks *Ksi*

Sama halnya dengan matriks *gamma*, matriks *ksi* juga digunakan pada waktu re-estimasi parameter HMM. Berikut ini adalah contoh perhitungan matriks *ksi*.

Melakukan iterasi dari t=0 hingga t<T-1, kemudian iterasi i=0 hingga i<*state* dan iterasi j=0 hingga j<*state*.

```
lnsum = -infinity
double[] x = sequence[t+1]
b = B[j].logProbabilityFunction(x)
logkasi[t][i,j] = lnfwd[t,i]+lnbwd[t+1,j]+logA[i,j]+b+w
lnsum = logsum(lnsum, logksi[t][i,j]
```

Untuk t=0, lnsum=-*infinity*, x = o[1] = -0.1754, i=0

j=0, b = B[0].logProbabilityFunction(-0.1754)

= -0.9343

logKsi[0][0,0] = lnfwd[0,0] + lnbwd[1,0] + logA[0,0] +b + w

= -0.93557 + -0.9206 + -0.69315 + -0.9342 + 0

= -3.4836

lnsum = logsum(-infinity, -3.4836) = -3.4836

j=1, b = B[1].logProbabilityFunction(-0.1754)

= -0.9343

logKsi[0][0,1] = lnfwd[0,0] + lnbwd[1,1] + logA[0,1] +b + w

= -0.93557 + -0.9206 + -0.69315 + -0.9342 + 0

= -3.4836

lnsum = logsum(-3.4836, -3.4836) = -2.7904

Setelah ditemukan nilai dari matriks *ksi* maka akan dinormalisasi dengan perulangan i=0 hingga i<*state* dan j=0 hingga j<*state* sebagai berikut.

Untuk i=0, j=0, maka

logksi[t][i,j] = logksi[t][i,j] – lnsum

= logksi[0][0,0] – (-2.7904)

= -3.4836 – (-2.7904)

= -0.69315

Lakukan perhitungan diatas hingga semua perulangan selesai dilakukan dan mendapatkan semua nilai matriks *ksi*.

e) Menentukan dan menghitung nilai *newlikelihood*

Menentukan dan menghitung nilai *loglikelihood* dari *sequence* yang ada kemudian menetapkan nilai *newlikelihood*. Berikut ini adalah contoh untuk menghitung *log-likelihood* dari *sequence*.

Melakukan iterasi j=0 hingga j<*state*.

```
Newloglikelihood = -infinity
Newloglikelihood = logsum(newloglikelihood, lnfwd[T-1,j])
```

Untuk j=0, newloglikelihood = logsum(newloglikelihood, lnfwd[2,0])
= logsum(-infinity, -4.1767)
= -4.1767

Untuk j=1, newloglikelihood = logsum(newloglikelihood, lnfwd[2, 1])
= logsum(-4.1767, -3.0781)
= -2.7904

f) Cek konvergensi

Mengecek jumlah iterasi sesuai dengan iterasi yang diberikan atau tidak, mengecek *oldlikelihood* sama dengan *newlikelihood* atau *newlikelihood* memiliki *error* yang kecil terhadap *oldlikelihood*. Jika semua benar maka proses dihentikan jika tidak maka akan di reestimasi kembali parameter HMM.

g) Re-estimasi Matriks Inisial

Matriks inisial digunakan untuk menghitung nilai matriks *forward* setelah parameter HMM terestimasi kembali. Adapun contoh perhitungannya adalah sebagai berikut :

Melakukan iterasi i=0 hingga i < panjang matriks inisial kemudian k=0 hingga k < panjang matriks *gamma*.

```
lnsum = -infinity
lnsum = logsum(lnsum, loggamma[k][0,i]
logP[i] = lnsum - logN
```

Untuk i=0, lnsum = -*infinity*, k=0

lnsum = logsum(lnsum, loggamma[0][0,0])

= logsum(-infinity, 0) = 0

logPi[0] = lnsum – logN

= 0 – log(1) = 0 – 0 = 0

Untuk i=1, lnsum = -*infinity*, k=0,

lnsum = logsum(lnsum, loggamma[0][0,1])

= logsum(-*infinity*, -*infinity*) = -*infinity*

logPi[0] = lnsum – logN

= -*infinity* – 0 = -*infinity*

h) Re-estimasi Matriks Transisi

Sama halnya dengan matriks inisial, matriks transisi juga digunakan untuk menghitung matriks *forward* dan *backward* setelah parameter HMM terestimasi. Adapun contoh perhitungannya adalah sebagai berikut :

Melakukan perulangan i=0 hingga i<*state* kemudian j=0 hingga j<*state* setelah itu menginisialisasi lnnum dan lnden. Kemudian melakukan perulangan lagi k=0 hingga k<panjang matriks *gamma* dan t=0 hingga t<T-1.

```
lnnum = -infinity
lnden = -infinity
T = panjang observasi = 3
lnnum = logsum(lnnum, logksi[k][t][i,j]
lnden = logsum(lnden, loggamma[k][t,i])
logA[i,j] = (lnnum == lnden) ? 0 : lnnum - lnden
```

Untuk i=0, j=0,

lnnum = -*infinity*, lnden = -*infinity*

k=0, 0<t<2,

t=0, lnnum = logsum(lnnum, logksi[0][0][0,0])

= logsum(-*infinity*, -0.69315) = -0.69315

lnden = logsum(lnden, loggamma[0][0,0])

= logsum(-*infinity*, 0) = 0

t=1, lnnum = logsum(lnnum, logksi[0][1][0,0])

= logsum(-0.69315, -1.3862) = -0.2876

lnden = logsum(lnden, loggamma[0][0,0])

= logsum(0, -0.69) = 0.41

Karena lnnum ≠ lnden maka

logA[0,0] = lnnum – lnden

= -0.2876-0.41 = -0.69315

Lakukan perhitungan diatas hingga semua perulangan selesai dilakukan dan didapatkan matriks transisi yang terestimasi.

i) Re-estimasi Matriks Emisi

Sama halnya seperti matriks inisial dan matriks transisi, matriks emisi juga digunakan untuk menghitung nilai matriks *forward* dan *backward* setelah parameter HMM terestimasi kembali. Adapun contoh perhitungannya adalah sebagai berikut :

Melakukan perulangan dari i=0 hingga i< panjang matriks emisi. Kemudian melakukan k=0 hingga k<jumlah observasi dan perulangan t=0 hingga t<T didalamnya.

```
lnsum = -infinity
T = jumlah observasi = 3
Weights[w] = loggamma[k][t,i]
lnsum = logsum(lnsum, weights[w])
weights[w] = weights[w] – lnsum
konversi ke probabilitas
p = exp(weights[w])
weights[w] = ((p=NaN or p=infinity) then 0, else p)
B[i].Fit(sample, weights)
```

Untuk i=0, lnsum = *-infinity*

k=0, 0<t<2,

t=0, weights[0] = loggamma[0][0,0] = 0

lnsum = logsum(lnsum, weights[0]) = 0

t=1, weights[1] = loggamma[0][1,0] = -0.69

lnsum = logsum(0, -0.69) = 0.41

t=2, weights[2] = loggamma[0][2,0] = -1.39

lnsum = logsum(0.41, -1.39) = 0.56

if insum ≠ *-infinity*

w=0, weights[0] = weights[0] – lnsum = 0 – 0.56 = -0.56

w=1, weights[1] = weights[1] – lnsum = -0.69– 0.56 = -1.25

w=2, weights[2] = weights[1] – lnsum = -1.39 – 0.56 = -1.95

konversi ke probabilitas :

w=0, p = exp(weights[0]) = exp(-0.56) = 0.5714

weights[0] = 0.5714

w=1, p = exp(weights[1]) = exp(-1.25) = 0.2857

weights[1] = 0.2857

w=2, p = exp(weights[2]) = exp(-1.95) = 0.1428

weights[2] = 0.1428

B[0] *weighted mean*

0	weights[0]	=	0.57142857		
	o[0]	=	-0.1824		
	sum	=	Weights[0] x o[0]	=	-0.1042286
1	weights[1]	=	0.28571429		
	0[1]	=	-0.1754		
	sum	=	Sum + Weights[1] x o[1]	=	-0.1543429
2	weights[2]	=	0.14285714		
	o[2]	=	-0.0575		
	sum	=	Sum + Weights[2] x o[2]	=	-0.1625571

W = weights[0]+weights[1]+weights[2]

=0.57142857 + 0.28571429 + 0.14285714 = 1

Sum/w = -0.16255/1 = -0.16255

B[0] *weighted variance*

sum	= 0		
squaresum	= 0		
weightsum	= 0		
i			
0	o[0]	=	-0.1824
	mean	=	-0.1626
	z	=	O[0]-mean = -0.0198
	w[0]	=	0.5714

	sum	=	w * (z*z)	= 0.0002	
	weightsum	=	0.5714		
	squaresum	=	0.3265		
1	o[1]	=	-0.1754		
	mean	=	-0.1626		
	z	=	O[1]-mean	= -0.0128	
	w[1]	=	0.2857		
	sum	=	Sum+w * (z*z)	=	0.0003
	weightsum	=	0.8571		
	squaresum	=	0.4082		
2	o[2]	=	-0.0575		
	mean	=	-0.1626		
	z	=	O[2]-mean	= 0.1051	
	w[2]	=	0.1429		
	sum	=	Sum+w * (z*z)	=	0.0018
	weightsum	=	w[0]+w[1]+w[2]	=	1.0000
	squaresum	=	$(w[0])^2+(w[1])^2+(w[2])^2$	=	0.4286

Sum/(weightsum-(squaresum/weightsum)) = 0.0018 / (1-(0.4285/1))
= 0.0018 / (1-0.4285)
= 0.0018 / 0.5715 = 0.00315

j) Evaluasi

Evaluasi digunakan ketika melakukan proses pengenalan suara dengan data *testing* setelah model-model sudah dibentuk dalam HMM. Langkah-langkah dalam perhitungan evaluasi sama dengan langkah-langkah pada perhitungan matriks *forward* dan perhitungan *newloglikelihood* hanya saja observasi yang digunakan adalah data *testing.*

BAB IV
IMPLEMENTASI

Pada bab implementasi akan dibahas mengenai implementasi sistem yang dibuat baik itu perangkat keras maupun perangkat lunak, batasan-batasan implementasi, implementasi program dari *Linear Predictive Coding* (*preemphasys, frame blocking, windowing*, autokorelasi, dan analisis LPC) sampai dengan *Hidden Markov Model*, implementasi *database* dan implementasi *interface.*

4.1 Implementasi Sistem

Perangkat lunak pengenalan suara untuk identifikasi aksara jawa dikembangkan dalam lingkungan implementasi yang terdiri dari perangkat keras dan perangkat lunak.

4.1.1 Implementasi Perangkat Keras

Implementasi perangkat keras yang dipakai dalam proses pembuatan sisitem dijelaskan pada tabel berikut ini :

Tabel 4.1 Implementasi perangkat keras komputer

Notebook ASUS K42JZ	
Nama *Hardware*	Spesifikasi
Processor	Intel (R) Core (TM) i3 M380 2.53 GHz
Memory (RAM)	4 GB
Harddisk	Western Digital WD3200BPVT SATA 320 GB
Motherboard	ASUSTeK Computer Inc.
Monitor	13.3" Widescreen LED Backlit Display
Recording Device	Eksternal Mic

4.1.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak yang dipakai dalam proses pembuatan sistem dijelaskan pada tabel berikut ini :

Tabel 4.2 Implementasi perangkat lunak komputer

Notebook ASUS K42JZ	
Nama *software*	Spesifikasi
Sistem operasi	Microsoft Windows 8 Pro 64-bit
Bahasa pemrograman	C#
Integrated Development Environment	Microsoft Visual Studio 2012
Database Management System	MySQL XAMPP 1.8.1

4.2 Batasan – Batasan Implementasi

Beberapa batasan dalam mengimplementasikan perangkat lunak pengenalan suara untuk identifikasi aksara jawa adalah sebagai berikut :

1. Perangkat lunak pengenalan suara dirancang dan dijalankan dengan menggunakan *Desktop Application*.
2. *File* yang digunakan memiliki format **.wav* dengan *sample rate* 8000Hz 8 bit mono.
3. *Database Management System* yang digunakan adalah MySQL.
4. Proses pemodelan *Hidden Markov Model* menggunakan *library* Accord.NET.
5. *File* wav yang akan di uji akan dipotong menggunakan WavePad Sound Editor.

4.3 Implementasi Program

Sistem pengenalan suara untuk aksara jawa ini terdiri dari beberapa proses, diantaranya proses *sampling*, proses ekstraksi fitur menggunakan *Linear Predictive Coding* (*preemphasis, frame blocking, windowing*, autokorelasi, dan analisis LPC), dan proses pengenalan menggunakan *Hidden Markov Model*.

4.3.1 Implementasi *Sampling File*

Proses *sampling file* ini bertujuan untuk membentuk sampel dari file yang digunakan. Proses implementasi *sampling* file seperti yang ditunjukkan pada Sourcecode 4.1 di bawah ini :

Sourcecode 4.1 Implementasi *Sampling File*

```
1  OpenFileDialog open = new OpenFileDialog();
2  open.Filter = "Wave File (*.wav|*.wav;";
3  if (open.ShowDialog() != DialogResult.OK) return;
4  wave = new NAudio.Wave.WaveFileReader(open.FileName);
5  int bts = wave.WaveFormat.BitsPerSample;
6  int sr = wave.WaveFormat.SampleRate;
7  label2.Text = System.IO.Path.GetFileName(open.FileName) +
8  Environment.NewLine + "Bit/Sample : " + bts +
9  ", Sample Rate : " + sr + Environment.NewLine +
10 Length :  + (int)wave.Length;
11
12 byte[] audData = new byte[wave.Length];
13 wave.Read(audData, 0, (int)wave.Length);
14
15 /* Sampling */
16 for (int i = 0; i < wave.Length; i++)
17 {
18   Array.Resize(ref fx, fx.Length + 1);
19   fx[fx.Length - 1] = audData[i];
20 }
```

Penjelasan dari *Sourcecode* 4.1 proses *sampling file* sebagai berikut ini :

Baris 1-10 adalah proses untuk membuka dan mengekstrak atribut dari file wav yang dipilih.

Baris 12-13 adalah proses membaca sinyal dari file wav.

Baris 16-20 adalah proses untuk membentuk sampel dari file wav yang dipilih.

4.3.2 Implementasi *Preemphasis*

Proses *preemphasis* bertujuan untuk mengkonversi sampel sinyal ucapan menjadi sinyal digital. Input dari proses ini berupa sampel dari hasil proses *sampling file*. Output yang dihasilkan adalah sampel sinyal dalam bentuk digital.

Proses implementasi *preemphasis* seperti yang ditunjukkan pada Sourcecode 4.2 di bawah ini :

Sourcecode 4.2 Implementasi *Preemphasis*

```
1  double[] sinyalRata = new double[1];
2  sinyalRata[0] = fx[0];
3  for (int i = 1; i < fx.Length; i++)
4  {
5      Array.Resize(ref sinyalRata, sinyalRata.Length + 1);
6      sinyalRata[i] = fx[i] - (0.9375 * fx[i - 1]);
7  }
8  return sinyalRata;
```

Penjelasan dari *Sourcecode* 4.2 proses *preemphasis* sebagai berikut ini :
Baris 1-2 adalah inisialisasi variabel untuk menyimpan sampel hasil *preemphasis.*
Baris 3-8 adalah proses untuk membentuk sampel dari konversi sampel sinyal hasil sampling menjadi sinyal digital.

4.3.3 Implementasi *Frame Blocking*

Proses *frame blocking* bertujuan untuk membagi hasil *preemphasis* ke dalam *frame-frame*. Input dari proses ini berupa sampel hasil proses *preemphasis*. Output yang dihasilkan adalah *frame-frame* yang berisikan sampel sebanyak N. Proses implementasi *frame blocking* seperti yang ditunjukkan pada Sourcecode 4.3 di bawah ini :

Sourcecode 4.3 Implementasi *frame blocking*

```
1  int totalsegmen = (sinyalRata.Length - (N - m)) / m;
2  double[,] segmen2Sinyal = new double[totalsegmen, N];
3  for (int i = 0; i < totalsegmen; i++)
4  {
5   for (int j = 0; j < N; j++)
6   {
7    int temp;
8    temp = (i * m) + j;
9    segmen2Sinyal[i, j] = sinyalRata[temp];
10  }
11 }
12 return segmen2Sinyal;
```

Penjelasan dari *Sourcecode* 4.3 proses *frame blocking* sebagai berikut ini :

Baris 1 adalah inisialisasi jumlah *frame* yang akan dihasilkan.

Baris 2 adalah inisialisasi variabel untuk menyimpan *frame-frame* yang dihasilkan.

Baris 2-11 adalah proses untuk membagi hasil *preemphasis* menjadi *frame-frame* yang masing-masing berisikan sampel sejumlah N dan dipisahkan sejauh m sampel.

4.3.4 Implementasi *Windowing*

Proses *windowing* bertujuan meminimalisasi sinyal tak kontinyu pada awal dan akhir masing-masing *frame*. Input dari proses ini berupa *frame-frame* yang telah dibuat sebelumnya. Output yang dihasilkan adalah *frame* yang berisikan sampel dengan nilai baru hasil *windowing*. Proses implementasi *windowing* seperti yang ditunjukkan pada Sourcecode 4.4 di bawah ini :

Sourcecode 4.4 Implementasi *windowing*

```
1  double w;
2  double[,] segmensinyalout = new double[totalsegmen, N];
3  for (int i = 0; i < totalsegmen; i++)
4  {
5   for (int j = 0; j < N; j++)
6   {
7    w = (0.54 - 0.46 * Math.Cos(2 * Math.PI * j / (N - 1)));
8    segmensinyalout[i, j] = segmen2Sinyal[i, j] * w;
9   }
10 }
11 return segmensinyalout;
```

Penjelasan dari *Sourcecode* 4.4 proses *windowing* sebagai berikut ini :
Baris 1-2 adalah inisialisasi variabel untuk menyimpan sampel hasil *windowing*.
Baris 3-10 adalah proses untuk menghitung nilai *windowing* setiap *frame* dari hasil *frame blocking*.

4.3.5 Implementasi Autokorelasi

Proses autokorelasi bertujuan untuk . Input dari proses ini berupa *frame* dan sampel hasil proses *windowing*. Proses ini akan menghasilkan korelasi tiap sampel dalam *frame* sejumlah orde LPC yang digunakan ditambah satu (ordeLPC+1). Proses implementasi *autokorelasi* seperti yang ditunjukkan pada Sourcecode 4.5 di bawah ini :

Sourcecode 4.5 Implementasi autokorelasi

```
1  double[,] ak = new double[totalsegmen, (p + 1)];
2  for (int i = 0; i < totalsegmen; i++)
3  {
4   for (int j = 0; j < (p + 1); j++)
5   {
6    ak[i, j] = 0;
7    for (int k = 1; k < (N - j); k++)
8    {
9     ak[i, j] = ak[i, j] + segmensinyalout[i, k] *
10                segmensinyalout[i, k + j]);
11   }
12  }
13 }
14 return ak;
```

Penjelasan dari *Sourcecode* 4.5 proses autokorelasi sebagai berikut ini :
Baris 1 adalah inisialisasi variabel untuk menyimpan hasil autokorelasi.
Baris 2-13 adalah proses untuk menghitung autokorelasi dari setiap *frame*.

4.3.6 Implementasi Analisis LPC

Proses analisis LPC bertujuan untuk mengubah hasil autokorelasi menjadi koefisien LPC. Input dari proses ini berupa hasil autokorelasi yang telah dibuat sebelumnya. Output yang dihasilkan adalah koefisien LPC dari file wav yang di*input*kan. Proses implementasi analisis LPC seperti yang ditunjukkan pada *Sourcecode* 4.6 di bawah ini :

Sourcecode 4.6 Implementasi analisis LPC

```
1  double[,] koef = new double[totalsegmen, p];
2  for (int i = 0; i < totalsegmen; i++)
3  {
4   double[] error = new double[p];
5   error[0] = ak[i, 0];
6   double[] k = new double[p];
7   double[,] a = new double[p, p];
8   for (int j = 0; j < p; j++)
9   {
10   if (j == 0)
11   {
12    k[0] = ak[i, 1] / ak[i, 0];
13   }
14   else
15   {
16    double tmp = 0;
17    for (int l = 0; l < j; l++)
18    {
19     tmp = tmp + (a[j - 1, l] * ak[i, Math.Abs(j - l)]);
20     k[j] = (ak[i, j + 1] - tmp) / error[j];
21    }
22   }
23   a[j, j] = k[j];
24   if (j >= 1)
25   {
26    for (int l = j - 1; l >= 0; l--)
27    {
28     a[j, l] = a[j - 1, l] - (k[j] * a[j - 1, (j - 1) - l]);
29    }
30   }
31   if (j < (p - 1))
32   {
33    error[j + 1] = (1 - Math.Pow(k[j], 2)) * error[j];
34   }
35  }
36
37  for (int n = 0; n < p; n++)
38  {
39   koef[i, n] = a[p-1, n];
40  }
41 }
42 return koef;
```

Penjelasan dari *Sourcecode* 4.6 proses analisis LPC sebagai berikut ini :

Baris 1 adalah inisialisasi variabel untuk menyimpan koefisien LPC dan error indeks-0.

Baris 4-7 adalah inisialisasi variabel untuk menyimpan error (e), koefisien pantulan (k), dan koefisien prediksi (a).

Baris 10-13 adalah menghitung nilai koefisien pantulan indeks-0.

Baris 14-22 adalah menghitung nilai koefisien pantulan indeks-1 dst.

Baris 23-30 adalah menghitung nilai koefisien prediksi.

Baris 31-34 adalah menghitung nilai error indeks-1 dst.

Baris 37-40 adalah menyimpan koefisisen LPC dari hasil koefisien prediksi.

4.3.7 Implementasi Parameter HMM

Dalam proses parameter HMM meliputi matriks transisi awal, matriks inisial awal, dan matriks emisi awal.

4.3.7.1 Implementasi Matriks Transisi dan Inisial Awal

Matriks transisi dan inisial digunakan untuk menjalankan algoritma *forward* dan *backward* baik proses inisialisasi maupun induksi.

Sourcecode 4.7 Implementasi *Matriks Transisi* dan *Inisial* awal

```
1  public Ergodic(int states, bool random)
2          {
3              if (states <= 0)
4              {
5                  throw new ArgumentOutOfRangeException(
                       "states", "Number of states should be higher
6  than zero.");
7              }
8              int m = System.Math.Min(States, Deepness);
9              this.states = states;
10             this.random = random;
11             this.pi = new double[states];
12             pi[0] = 1.0;
13         }
   public int Create(bool logarithm, out double[,] transitionMatrix,
14 out double[] initialState)
15       {
16             double[,] A = new double[States, States];
17                 // Create A using equal uniform probabilities,
18                 for (int i = 0; i < states; i++)
19                     double d = 1.0 / Math.Min(m, states - i);
20                     for (int j = i; j < states && (j - i) < m; j++)
21                         A[i, j] = d;
22                  }
23             if (logarithm)
24             {
25                 transitionMatrix = Matrix.Log(A);
26                 initialState = Matrix.Log(pi);
27             }
28             else
29             {
30                 transitionMatrix = A;
31                 initialState = (double[])pi.Clone();
32             }
33
```

```
34              return States;
35          }
```

Penjelasan implementasi matriks transisi dan inisial sebagai berikut :

Baris 8-11 adalah menentukan *state* dan pi.

Baris 16-21 adalah menghitung nilai matriks transisi.

Baris 23-27 adalah menentukan nilai matriks transisi dan inisial jika menggunakan logaritmik.

4.3.7.2 Implementasi Matriks Emisi Awal

Matriks emisi juga digunakan untuk menjalankan algoritma *forward* dan *backward* baik proses inisialisasi maupun induksi.

Sourcecode 4.8 Implementasi *Matriks emisi* awal

```
    public HiddenMarkovModel(ITopology topology, TDistribution
1   emissions)
2               : base(topology)
3           {
4               if (emissions == null)
5               {
6                   throw new ArgumentNullException("emissions");
7               }
8
9               // Initialize B using the initial distribution
10              B = new TDistribution[States];
11
12              for (int i = 0; i < B.Length; i++)
13                  B[i] = (TDistribution)emissions.Clone();
14          }
15
16   // Distribution parameters
17          private double mean = 0;   // mean μ
18          private double stdDev = 1; // standard deviation σ
19
20  public override object Clone()
21          {
22              return new NormalDistribution(mean, stdDev);
23          }
24
    public NormalDistribution([Real] double mean, [Positive] double
25  stdDev)
26          {
27              if (stdDev <= 0)
28              {
29                  throw new ArgumentOutOfRangeException("stdDev",
30                      "Standard deviation must be positive.");
31              }
32
33              initialize(mean, stdDev, stdDev * stdDev);
34          }
35
```

```
36  private void initialize(double mu, double dev, double var)
37          {
38              this.mean = mu;
39              this.stdDev = dev;
40              this.variance = var;
41
42              // Compute derived values
43              this.lnconstant = -Math.Log(Constants.Sqrt2PI * dev);
44          }
```

Penjelasan implementasi matriks emisi sebagai berikut :

Baris 1-14 adalah kostruktor dari fungsi.

Baris 16-18 adalah inisialisasi nilai *mean* dan *standard deviasi* yang digunakan dalam distribusi normal.

Baris 36-44 adalah inisialisasi nilai *mean*, *standard deviasi* dan *variance* yang digunakan dalam matriks emisi.

4.3.7.3 Implementasi Algoritma Forward

Algoritma *forward* digunakan untuk membentuk matriks *forward* yang digunakan dalam menghitung matriks *ksi* dan *gamma*.

Sourcecode 4.9 Implementasi algoritma *forward*

```
1   public static void LogForward<TDistribution>
2   (HiddenMarkovModel<TDistribution> model, double[][] observations,
3   double[,] lnFwd)where TDistribution : IDistribution
4   {
5               int states = model.States;
6               var logA = model.Transitions;
7               var logB = model.Emissions;
8               var logPi = model.Probabilities;
9               int T = observations.Length;
10              // Ensures minimum requirements
11            System.Diagnostics.Debug.Assert(lnFwd.GetLength(0) >= T);
12        System.Diagnostics.Debug.Assert(lnFwd.GetLength(1) == states);
13                Array.Clear(lnFwd, 0, lnFwd.Length);
14
15              // 1. Initialization
16              for (int i = 0; i < states; i++)
17                  lnFwd[0, i] = logPi[i] +
18                     logB[i].LogProbabilityFunction(observations[0]);
19              // 2. Induction
20              for (int t = 1; t < T; t++)
21              {
22               double[] x = observations[t];
23                for (int i = 0; i < states; i++)
24                {
25                  double sum = Double.NegativeInfinity;
26                  for (int j = 0; j < states; j++)
27                      sum = Special.LogSum(sum, lnFwd[t - 1, j] +
28                            logA[j, i]);
29                      lnFwd[t, i] = sum +
```

```
30                              logB[i].LogProbabilityFunction(x);
31                          }
32                      }
33                  System.Diagnostics.Debug.Assert(!lnFwd.HasNaN());
34              }
```

Penjelasan implementasi algoritma *forward* sebagai berikut :

Baris 3-9 adalah menentukan batasan dan variable yang dipakai dalam algoritma *forward*.

Baris 17-20 adalah proses inisialisasi dari algoritma *forward*.

Baris 21-33 adalah proses induksi dari algoritma *forward*.

4.3.7.4 Implementasi Algoritma Backward

Algoritma *backward* juga digunakan untuk menghitung nilai dari matriks *gamma* dan *ksi*.

Sourcecode 4.10 Implementasi algoritma *backward*

```
1   public static void LogBackward<TDistribution>
2   (HiddenMarkovModel<TDistribution> model, double[][] observations,
3   double[,] lnBwd) where TDistribution : IDistribution
4   {
5       int states = model.States;
6       var logA = model.Transitions;
7       var logB = model.Emissions;
8       var logPi = model.Probabilities;
9
10      int T = observations.Length;
11
12      // Ensures minimum requirements
13      System.Diagnostics.Debug.Assert(lnBwd.GetLength(0) >= T);
14      System.Diagnostics.Debug.Assert(lnBwd.GetLength(1) == states);
15      Array.Clear(lnBwd, 0, lnBwd.Length);
16
17              // 1. Initialization
18              for (int i = 0; i < states; i++)
19                  lnBwd[T - 1, i] = 0;
20
21              // 2. Induction
22              for (int t = T - 2; t >= 0; t--)
23              {
24                  for (int i = 0; i < states; i++)
25                  {
26                      double sum = Double.NegativeInfinity;
27                      for (int j = 0; j < states; j++)
28                          sum = Special.LogSum(sum, lnBwd[t + 1, j] +
29  logA[i, j] + logB[j].LogProbabilityFunction(observations[t + 1]));
30                      lnBwd[t, i] = sum;
31                  }
32              }
33              System.Diagnostics.Debug.Assert(!lnBwd.HasNaN());
34          }
```

Penjelasan implementasi algoritma *backward* sebagai berikut :

Baris 3-9 adalah menentukan batasan dan variabel yang dipakai dalam algoritma *backward*.

Baris 17-20 adalah proses inisialisasi dari algoritma *backward*.

Baris 21-33 adalah proses induksi dari algoritma *backward*.

4.3.7.5 Implementasi Matriks Gamma

Nilai matriks *gamma* digunakan untuk estimasi kembali parameter HMM.

Sourcecode 4.11 Implementasi *Matriks Gamma*

```
1  // Calculate gamma values for next computations
2     for (int t = 0; t < T; t++)
3     {
4         double lnsum = Double.NegativeInfinity;
5         for (int k = 0; k < states; k++)
6         {
7             logGamma[t, k] = lnFwd[t, k] + lnBwd[t, k] + w;
8             lnsum = Special.LogSum(lnsum, logGamma[t, k]);
9         }

10 System.Diagnostics.Debug.Assert(!Double.IsNaN(lnsum));
11
12        // Normalize if different from zero
13           if (lnsum != Double.NegativeInfinity)
14           for (int k = 0; k < states; k++)
15               logGamma[t, k] = logGamma[t, k] - lnsum;
16    }
```

Penjelasan implementasi matriks *gamma* sebagai berikut :

Baris 5-9 adalah proses perhitungan nilai matriks *gamma*.

Baris 13-17 adalah proses perhitungan normalisasi nilai matriks *gamma*.

4.3.7.6 Implementasi Matriks Ksi

Nilai matriks *ksi* juga digunakan untuk estimasi kembali parameter HMM.

Sourcecode 4.12 Implementasi *Matriks Ksi*

```
1  protected override void ComputeKsi(int index, double[,] lnFwd,
2  double[,] lnBwd)
3  {
4     int states = model.States;
5     double[,] logA = model.Transitions;
6     TDistribution[] B = model.Emissions;
7
8     var sequence = vectorObservations[index];
9
10    int T = sequence.Length;
11    var logKsi = LogKsi[index];
12    var w = LogWeights[index];
13
14    for (int t = 0; t < T - 1; t++)
15    {
```

```
16          double lnsum = Double.NegativeInfinity;
17          double[] x = sequence[t + 1];
18          for (int i = 0; i < states; i++)
19          {
20              for (int j = 0; j < states; j++)
21              {
22                  double b = B[j].LogProbabilityFunction(x);
23                  logKsi[t][i, j] = lnFwd[t, i] + lnBwd[t + 1, j] +
24                                    logA[i, j] + b + w;
25                  lnsum = Special.LogSum(lnsum, logKsi[t][i, j]);
26              }
27          }
28
29  System.Diagnostics.Debug.Assert(!Double.IsNaN(lnsum));
30
31          // Normalize if different from zero
32          if (lnsum != Double.NegativeInfinity)
33              for (int i = 0; i < states; i++)
34              for (int j = 0; j < states; j++)
35                  logKsi[t][i, j] = logKsi[t][i, j] - lnsum;
36      }
37  }
```

Penjelasan implementasi matriks *ksi* sebagai berikut :

Baris 3-11 adalah menentukan batasan dan variabel yang dipakai dalam pembentukan matriks *ksi*.

Baris 19-27 adalah proses perhitungan nilai matriks *ksi*.

Baris 31-37 adalah proses perhitungan normalisasi nilai matriks *ksi*.

4.3.8 Implementasi Re-estimasi Parameter HMM

4.3.8.1 Implementasi Re-estimasi Matriks Inisial

Reestimasi matriks inisial bertujuan untuk menghitung kembali nilai matriks *forward* dan *backward* yang belum mencapai kondisi konvergen.

Sourcecode 4.13 Implementasi *Re-estimasi matriks inisial*

```
1  // 3.1 Re-estimation of initial state probabilities
2  for (int i = 0; i < logP.Length; i++)
3  {
4      double lnsum = Double.NegativeInfinity;
5      for (int k = 0; k < LogGamma.Length; k++)
6          lnsum = Special.LogSum(lnsum, LogGamma[k][0, i]);
7      logP[i] = lnsum - logN;
8  }
```

Penjelasan implementasi reestimasi matriks inisial sebagai berikut :

Baris 4-7 adalah proses menghitung nilai matriks inisial.

4.3.8.2 Implementasi Re-estimasi Matriks Transisi

Matriks transisi juga diestimasi kembali untuk menghitung nilai matriks *forward* dan *backward* agar mencapai kondisi konvergen.

Sourcecode 4.14 Implementasi *Re-estimasi matriks transisi*

```
1  // 3.2 Re-estimation of transition probabilities
2  for (int i = 0; i < states; i++)
3  {
4      for (int j = 0; j < states; j++)
5      {
6          double lnnum = Double.NegativeInfinity;
7          double lnden = Double.NegativeInfinity;
8
9          for (int k = 0; k < LogGamma.Length; k++)
10         {
11             int T = observations[k].Length;
12
13             for (int t = 0; t < T - 1; t++)
14             {
15              lnnum = Special.LogSum(lnnum, LogKsi[k][t][i, j]);
16              lnden = Special.LogSum(lnden, LogGamma[k][t, i]);
17             }
18         }
19
20       logA[i, j] = (lnnum == lnden) ? 0 : lnnum - lnden;
21
22       System.Diagnostics.Debug.Assert(!Double.IsNaN(logA[i, j]));
23     }
24 }
```

Penjelasan implementasi reestimasi matriks transisi sebagai berikut :

Baris 6-7 adalah inisialisasi variabel yang digunakan untuk menghitung nilai matriks transisi.

Baris 9-18 adalah proses menghitung kembali nilai variabel dengan menggunakan matriks *ksi* dan *gamma*.

Baris 20 adalah menentukan kembali nilai matriks transisi.

4.3.8.3 Implementasi Re-estimasi Matriks Emisi

Sama halnya dengan matriks inisial dan transisi, matriks emisi diestimasi kembali untuk menghitung nilai matriks *forward* dan *backward* agar mencapai kondisi konvergen.

Sourcecode 4.15 Implementasi *Re-estimasi matriks emisi*

```
protected override void UpdateEmissions()
{
     var B = model.Emissions;

    // For each state i in the model
     for (int i = 0; i < B.Length; i++)
     {
        double lnsum = Double.NegativeInfinity;

        // For each observation sequence k
        for (int k = 0, w = 0; k < vectorObservations.Length; k++)
        {
           int T = vectorObservations[k].Length;

           // For each observation t in k
           for (int t = 0; t < T; t++, w++)
           {
                weights[w] = LogGamma[k][t, i];
                lnsum = Special.LogSum(lnsum, weights[w]);
           }
        }

System.Diagnostics.Debug.Assert(!Double.IsNaN(lnsum));

       if (lnsum != Double.NegativeInfinity)
       for (int w = 0; w < weights.Length; w++)
            weights[w] = weights[w] - lnsum;

       // Convert to probabilities
       for (int w = 0; w < weights.Length; w++)
       {
           double p = Math.Exp(weights[w]);
           weights[w] = (Double.IsNaN(p) || Double.IsInfinity(p)) ?
                        0.0 : p;
       }
       // Estimate the distribution for state i
          B[i].Fit(samples, weights, fittingOptions);
   }
}

public void Fit(double[] observations, double[] weights,
NormalOptions options)
{
    if (immutable)
    throw new
 InvalidOperationException("NormalDistribution.Standard is
```

```
47   immutable.");
48      double mu, var;
49      if (weights != null)
50      {
51  #if DEBUG
52        for (int i = 0; i < weights.Length; i++)
53        if (Double.IsNaN(weights[i])||Double.IsInfinity(weights[i]))
54        throw new ArgumentException("Invalid
55              numbers in the weight vector.", "weights");
56   #endif
57         // Compute weighted mean
58         mu = Statistics.Tools.WeightedMean(observations, weights);
59         // Compute weighted variance
60         var = Statistics.Tools.WeightedVariance
61               (observations, weights, mu);
62       }
63       else
64       {
65         // Compute weighted mean
66         mu = Statistics.Tools.Mean(observations);
67
68         // Compute weighted variance
69         var = Statistics.Tools.Variance(observations, mu);
70       }
71
72       if (options != null)
73       {
74         // Parse optional estimation options
75        double regularization = options.Regularization;
76
77        if (var == 0 || Double.IsNaN(var) || Double.IsInfinity(var))
78        var = regularization;
79       }
80
81       if (Double.IsNaN(var) || var <= 0)
82       {
83      throw new ArgumentException("Variance is zero. Try specifying "
84      + "a regularization constant in the fitting options.");
85     }
86
87     initialize(mu, Math.Sqrt(var), var);
88  }
```

Penjelasan implementasi reestimasi matriks emisi sebagai berikut :

Baris 3 untuk inisialisasi B dengan matriks emisi awal.

Baris 8 untuk menginisialisasi nilai *lnsum*.

Baris 18-19 menghitung nilai *weights* dan *lnsum* yang baru.

Baris 25-27 menghitung nilai baru *weights* jika *lnsum* tidak sama dengan *-infinity*.

Baris 31-35 mengkonversi nilai *weights* menjadi probabilitas.

Baris 42-88 menghitung nilai *mean* dan *variance* yang baru.

4.3.9 Implementasi Evaluasi HMM

Evaluasi HMM digunakan untuk menghitung likelihood dari data *testing* sehingga dapat dikenali sebagai sebuah kelas dalam model yang sudah dibuat.

Sourcecode 4.16 Implementasi *Evaluasi HMM*

```
public static int Evaluate(double[] sequence, out double max,
out string aksara)
{
    // Calculate the probability that the given
    //  sequences originated from the model
    double[] likelihood = new double[classes];;
    for (int j = 0; j < classes; j++)
    {
       likelihood[j] = classifier.Models[j].Evaluate(sequence);
    }
    int kelas;
    max = likelihood.Max(out kelas);
    aksara = koneksi.getAksara(kelas);
    return kelas;
}
public double Evaluate(Array observations)
{
    if (observations == null)
    throw new ArgumentNullException("observations");

    if (observations.Length == 0)
      return Double.NegativeInfinity;
      double[][] x = MarkovHelperMethods.checkAndConvert
                   (observations, dimension);

      // Forward algorithm
      double logLikelihood;

     // Compute forward probabilities
   ForwardBackwardAlgorithm.LogForward(this, x, out logLikelihood);

     // Return the sequence probability
      return logLikelihood;
}
public static double[,] LogForward<TDistribution>
(HiddenMarkovModel<TDistribution> model, double[][] observations,
out double logLikelihood)
where TDistribution : IDistribution
{
    int T = observations.Length;
    int states = model.States;
    double[,] lnFwd = new double[T, states];

    ForwardBackwardAlgorithm.LogForward<Tdistribution>
                        (model, observations, lnFwd);
    logLikelihood = Double.NegativeInfinity;
    for (int i = 0; i < states; i++)
        logLikelihood = Special.LogSum(logLikelihood,
                        lnFwd[observations.Length - 1, i]);
```

```
50      return lnFwd;
51  }
```

Penjelasan implementasi evaluasi HMM sebagai berikut :

Baris 5 untuk menginisialisasi array *likelihood*.

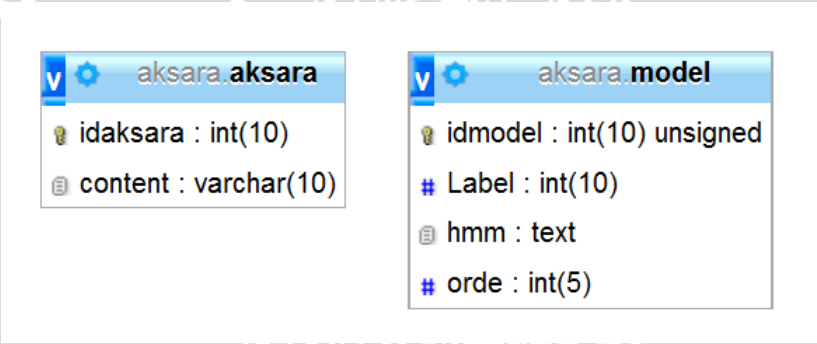
Baris 6-9 untuk menghitung *likelihood* data *testing* dengan masing-masing model yang sudah dibentuk.

Baris 30 untuk memanggil algoritma *forward*.

Baris 36-52 untuk menghitung matriks *forward* dan *likelihood* dari data *testing* dengan setiap model.

4.4 Implementasi Tabel dalam Sistem Pengenalan

Untuk implementasi database yang digunakan dalam sistem pengenalan suara ditunjukkan pada Gambar 4.1 sebagai berikut :



Gambar 4.1 Implementasi Tabel Database

4.5 Implementasi Interface

Implementasi interface terdiri dari 3 bagian utama, yaitu :

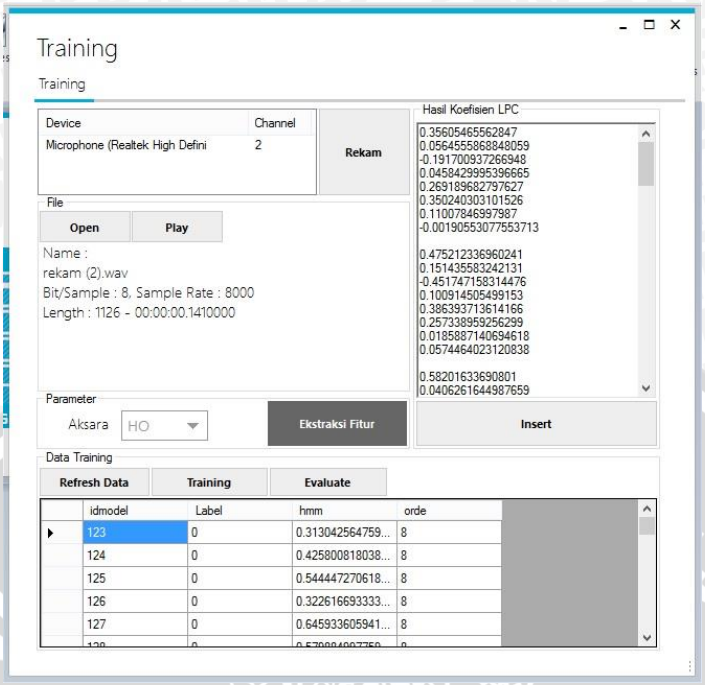
a. Form Utama

Form ini menampilkan *menu* utama yang ada pada pengenalan suara. Adapun *menu* dalam form ini antara lain : form untuk memilih orde LPC yang digunakan, *menu training* yang digunakan untuk menginput dan mentraining data dan *menu testing* digunakan untuk mengetahui hasil pengenalan suara. Adapun tampilannya ditunjukkan pada Gambar 4.2 berikut ini :

Gambar 4.2 Implementasi Form Utama

c. Form *Training*

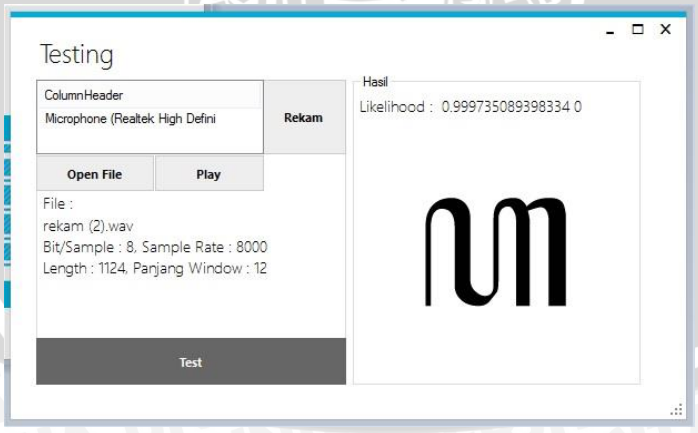
Form ini berisi *menu* untuk merekam suara, membuka *file* wav, memutar *file* wav, menampilkan *properties file* wav, memilih aksara, mengekstraksi fitur dan menampilkan hasil ekstraksi fitur, menampilkan data *training* dari *database*, serta *menu* untuk *training*. Adapun tampilan form *training* ditunjukkan pada Gambar 4.3 berikut ini :



Gambar 4.3 Implementasi Form *Training*

d. Form *Testing*

Form ini memasukkan *file* wav yang akan diuji dan hasil pengenalannya sama dengan id dikelasnya atau tidak. *Menu* yang terdapat pada form ini adalah rekam suara, *open file*, *play*, *test* serta form untuk menampilkan likelihood dan hasil aksara. Adapun tampilan form ditunjukkan pada Gambar 4.4 berikut ini :



Gambar 4.4 Implementasi Form *Testing*

BAB V
PENGUJIAN DAN ANALISIS

Pada bab pengujian dan analisis akan dibahas mengenai pengujian terhadap sistem dan analisis yang didapatkan setelah pengujian. Pengujian akan dijelaskan pada subbab 5.1 sedangkan analisis hasil pengujian akan dijelaskan pada subbab 5.2.

5.1 Pengujian

Pada pengujian sistem pengenalan suara akan diambil file wav hasil rekam sebanyak 4 file dari masing-masing kelas (20 kelas huruf aksara jawa). Sedangkan data *training* berupa file wav hasil rekam yang digunakan adalah sebanyak 6 file dari masing-masing kelas. Dari 80 file wav untuk pengujian kemudian akan dicoba pada sistem pengenalan suara untuk identifikasi aksara jawa menggunakan metode *Linear Predictive Coding* dan *Hidden Markov Model* yang sudah dibuat untuk dikenali oleh data *training* yang sudah tersedia.

Pengujian dilakukan dengan membuat 4 skenario. Skenario pertama dilakukan untuk menguji fungsionalitas metode yang telah digunakan. Skenario kedua dilakukan untuk menguji akurasi sistem terhadap jumlah kelas yang digunakan. Skenario ketiga dilakukan untuk menghitung akurasi sistem sesuai sinyal yang digunakan. Sedangkan skenario keempat adalah untuk menghitung akurasi menggunakan metode *Linear Predictive Coding* dengan menggunakan orde LPC yang berbeda dalam setiap pengujian yaitu 8, 10, 12, 14, dan 16. Orde LPC digunakan untuk proses ekstraksi fitur ketika memasukkan data *training* dan data *testing*. Pengujian dilakukan untuk mengetahui pengaruh potongan sinyal dan orde LPC terhadap akurasi sistem. Pengujian dilakukan dengan menghitung data suara yang teridentifikasi dengan tepat dari 80 data *testing* yang digunakan.

5.2 Analisis Hasil Pengujian

Dari hasil pengujian yang telah dilakukan persen akurasi sesuai skenario yang diujikan. Persen akurasi didapat dengan cara membagi jumlah aksara jawa yang dikenali dengan jumlah seluruh data yang digunakan sebagai *testing*.

5.2.1 Skenario 1

Pengujian dalam skenario 1 bertujuan untuk menguji fungsionalitas metode pengenalan yang digunakan. Pengujian dilakukan untuk mengetahui apakah sistem yang telah dibuat sudah memenuhi dalam mengimplementasikan metode dalam pengenalan. Pada skenario 1 sistem akan diuji menggunakan 1 kelas aksara dan menggunakan sampel 8 aksara yaitu Ha, Na, Da, Ta, Pa, Dha, Ma, dan Ga.

Tabel 5.1 Hasil pengujian skenario 1

Aksara	Benar	Salah	Total
Ha	8	0	8
Na	8	0	8
Da	8	0	8
Ta	8	0	8
Pa	8	0	8
Dha	8	0	8
Ma	8	0	8
Ga	8	0	8

Berdasarkan hasil pengujian dengan beberapa sampel aksara diatas, sistem mampu mengenali data yang diuji dengan benar. Hal tersebut memperlihatkan bahwa sistem yang telah dibuat sudah sesuai dan memenuhi dalam mengimplementasikan metode dalam pengenalan.

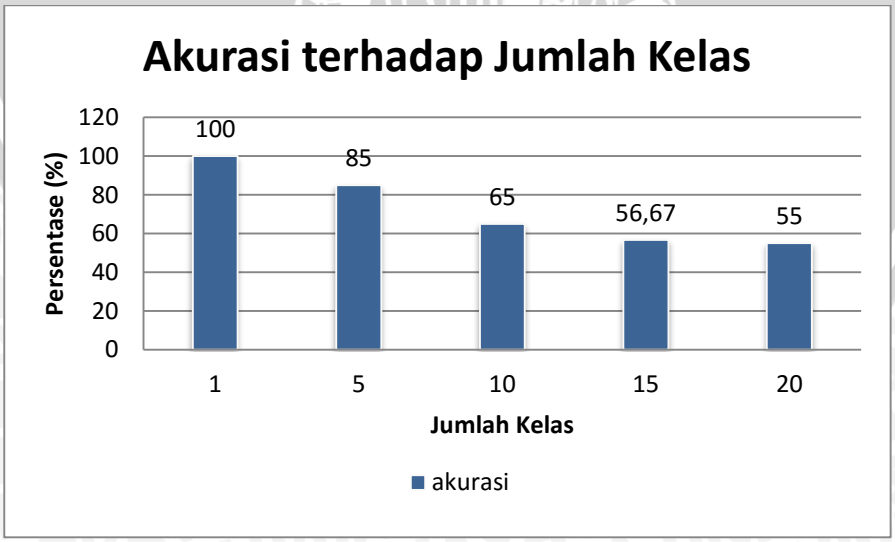
5.2.2 Skenario 2

Pengujian dalam skenario 2 bertujuan untuk menguji akurasi sistem terhadap jumlah kelas yang digunakan dalam proses pembentukan model sedangkan data-data training yang digunakan memiliki fitur yang hampir sama. Pada skenario 2 sistem akan diuji menggunakan parameter jumlah kelas aksara yang digunakan yaitu 1 kelas, 5 kelas, 10 kelas, 15 kelas, dan 20 kelas.

Tabel 5.2 Hasil pengujian skenario 2

Jumlah Kelas	Benar	Salah	Total Data
1	8	0	8
5	17	3	20
10	26	14	40
15	34	26	60
20	44	36	80

Berdasarkan Tabel 5.2 maka diperoleh tingkat keberhasilan pengenalan aksara pada skenario 2 dengan akurasi tertinggi 100% untuk pengenalan menggunakan 1 kelas aksara, 85% untuk 5 kelas, 65% untuk 10 kelas, 56.66% untuk 15 kelas, dan 55% untuk 20 kelas. (Hasil percobaan dapat dilihat pada halaman lampiran). Pengujian akurasi skenario 2 dapat digambarkan dalam Grafik 5.1 berikut ini.



Gambar 5.1 Grafik akurasi terhadap uji jumlah kelas.

Berdasarkan hasil pengujian dengan beberapa kelas diatas, dapat dihasilkan suatu analisis yaitu dengan bertambahnya data yang memiliki kemiripan fitur dan bertambahnya kelas yang dipakai maka akurasi pengenalannya semakin menurun. Hal ini terbukti ketika sistem menggunakan 5 kelas yang menghasilkan 85% akurasi kemudian menurun menjadi 65% ketika kelas yang digunakan adalah 10. Penyebab terjadinya penurunan akurasi dikarenakan hasil ekstraksi fitur LPC setiap data *training* hampir sama sehingga dapat mengakibatkan kesalahan dalam pemodelan ketika kelas aksara yang digunakan semakin banyak dan akurasi pengenalan akan menurun.

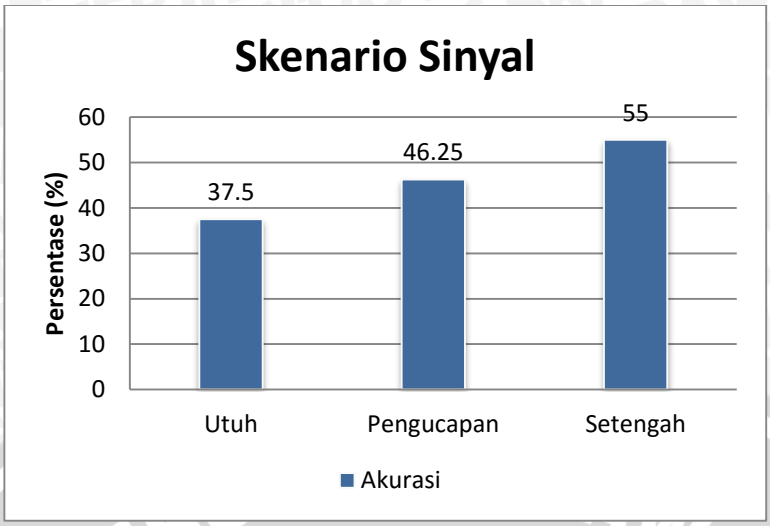
5.2.3 Skenario 3

Pengujian dalam skenario 3 bertujuan untuk menguji akurasi sistem terhadap sinyal yang digunakan. Pengujian dilakukan untuk mengetahui apakah sinyal yang digunakan mempengaruhi akurasi pengenalan sistem. Pada skenario 3 sistem akan diuji menggunakan parameter panjang sinyal yang digunakan yaitu sinyal yang utuh sepanjang 1.5 detik, sinyal yang sudah dipotong sesuai pengucapan aksara, dan sinyal yang dipotong setengah menjadi ucapan konsonan aksara. Sinyal dipotong menggunakan aplikasi WavePad Sound Editor Versi 6.18.

Tabel 5.3 Hasil pengujian skenario 3

Sinyal	Benar	Salah	Total
Utuh (1.5 detik)	30	50	80
Pengucapan (0.33)	37	43	80
Setengah (0.14)	44	36	80

Berdasarkan Tabel 5.3 maka diperoleh tingkat keberhasilan pengenalan aksara pada skenario 3 dengan akurasi tertinggi 55% untuk pengenalan menggunakan sinyal yang sudah dipotong menjadi setengah pengucapan, 46.25% untuk sinyal yang dipotong sesuai pengucapan, dan 37.5% untuk sinyal yang utuh sepanjang 1.5 detik (Hasil percobaan dapat dilihat pada halaman lampiran). Pengujian akurasi skenario 3 dapat digambarkan dalam Grafik 5.2 berikut ini.



Gambar 5.2 Grafik akurasi terhadap panjang sinyal.

Berdasarkan hasil pengujian sinyal yang digunakan, dapat dihasilkan beberapa analisis yaitu :

1. Pada pengujian pertama menggunakan sinyal utuh sepanjang 1.5 detik terdapat sinyal-sinyal yang tidak berpengaruh terhadap pengucapan aksara di awal dan akhir setiap data yang ikut terproses dalam LPC. Sinyal-sinyal di awal tersebut menghasilkan nilai fitur yang sama pada setiap data *training*. Hal tersebut menghasilkan akurasi pengenalan sistem terendah.

2. Pada pengujian kedua menggunakan sinyal yang sudah dipotong sesuai pengucapan aksara jawa terlebih dahulu menggunakan WavePad sehingga menghilangkan sinyal-sinyal diawal dan diakhir data yang tidak digunakan. Hal tersebut menghasilkan nilai fitur yang berbeda setiap data *training* sehingga dapat meningkatkan akurasi pengenalan sistem.

3. Pada pengujian ketiga menggunakan sinyal yang sudah dipotong meggunakan WavePad menjadi pengucapan konsonan awal aksara jawa sehingga menghilangkan sinyal yang tidak digunakan dan pengucapan vokal dari aksara jawa. Pemotongan dilakukan karena pengucapan vokal pada semua aksara memiliki kesamaan sehingga dapat menghasilkan sinyal yang hampir sama

juga. Dari data yang telah dipotong tersebut menghasilkan tingkat akurasi yang lebih tinggi daripada pengujian sebelumnya.

Dari beberapa anilisis diatas dapat disimpulkan bahwa sinyal-sinyal yang tidak berkaitan dengan pengucapan aksara jawa mempengaruhi hasil ekstraksi fitur dan menyebabkan penurunan dalam akurasi pengenalan pada sistem, begitu juga dengan pengucapan vokal yang sama.

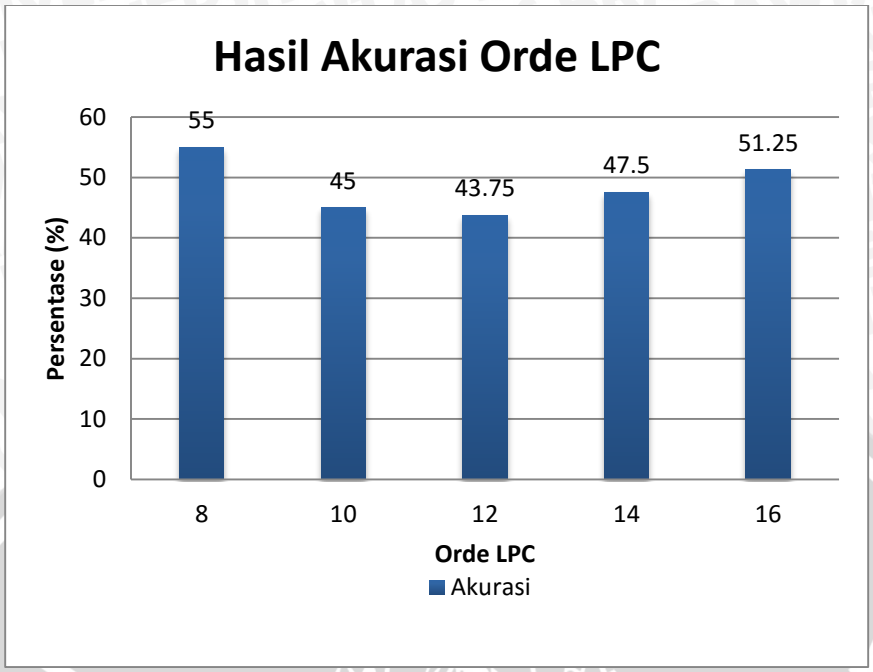
5.2.4 Skenario 4

Pengujian dalam skenario 4 bertujuan untuk menguji akurasi sistem terhadap orde LPC yang digunakan pada proses ekstraksi fitur. Pengujian dilakukan untuk mengetahui apakah orde LPC yang digunakan saat mengekstraksi fitur dari sinyal mempengaruhi akurasi pengenalan sistem. Pada skenario 4 sistem akan diuji menggunakan parameter orde LPC yang bernilai 8, 10, 12, 14, dan 16.

Tabel 5.4 Hasil pengujian skenario 4

Orde LPC	Benar	Salah	Total
8	44	36	80
10	36	44	80
12	35	45	80
14	38	42	80
16	41	39	80

Berdasarkan Tabel 5.4 maka diperoleh tingkat keberhasilan pengenalan aksara pada skenario 3 dengan akurasi tertinggi 55% untuk orde LPC 8, 45% untuk orde LPC 10, 43.75% untuk orde LPC 12, 47.5% untuk orde LPC 14, dan 51.25% untuk orde LPC 16 (Hasil percobaan dapat dilihat pada halaman lampiran). Pengujian akurasi skenario 4 dapat digambarkan dalam Grafik 5.3 berikut ini.



Gambar 5.3 Grafik akurasi terhadap orde LPC

Berdasarkan hasil pengujian terhadap orde LPC dengan beberapa nilai, dapat dihasilkan suatu analisis yaitu orde LPC yang dapat digunakan untuk memaksimalkan ekstraksi fitur dan pengenalan aksara jawa adalah 8

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan penelitian yang dilakukan dalam pembuatan aplikasi pengenalan suara untuk identifikasi aksara jawa menggunakan *Linear Predictive Coding* dan *Hidden Markov Model* dapat ditarik beberapa kesimpulan sebagai berikut :

1. Aplikasi pengenalan suara dalam penelitian ini mengimplementasikan metode *Linear Predictive Coding* sebagai metode untuk mengekstraksi fitur dari *file* wav yang akan melalui proses *preemphasis, frame blocking, windowing*, autokorelasi dan analisis LPC sehingga akan menghasilkan koefisien LPC. Dalam penelitian ini Orde LPC yang digunakan untuk memaksimalkan hasil ekstraksi fitur adalah 8. Sedangkan untuk pemodelan dan pencocokan pola dari data *training* dan data *testing* yang sudah disimpan menggunakan *Hidden Markov Model* dengan topologi *forward*.
2. Metode *Linear Predictive Coding* dapat diimplementasikan dalam pengenalan suara dengan objek seperti aksara jawa meskipun kurang maksimal, karena setiap aksara memiliki pengucapan yang hampir sama sehingga pemrosesan sinyal juga menghasilkan nilai-nilai fitur yang hampir sama.
3. Metode *Hidden Markov Model* berjalan dengan kurang baik untuk digunakan dalam pengenalan objek seperti aksara jawa. Hal tersebut disebabkan oleh jumlah aksara jawa yang banyak (20) dan hasil ekstraksi fitur yang hampir sama sehingga dapat membingungkan proses pembentukan model dalam HMM.
4. Sinyal yang tidak berhubungan dalam pengucapan suara dan pengucapan suara yang hampir sama dapat mempengaruhi hasil pengenalan suara.

6.2 Saran

Berdasarkan penelitian yang dilakukan dalam pembuatan aplikasi pengenalan suara untuk identifikasi aksara jawa menggunakan *Linear Predictive Coding* dan *Hidden Markov Model* dapat diberikan beberapa saran sebagai berikut :

1. Untuk studi kasus dengan data yang memiliki kemiripan seperti dalam penelitian ini diperlukan metode pengekstraksi fitur selain *Linear Predictive Coding* yang dapat menghasilkan fitur yang lebih baik.
2. Peningkatan dan pengoptimalan akurasi pada proses pencocokan suara menggunakan *Hidden Markov Model* dapat lebih ditingkatkan lagi.
3. Penggunaan *preprocessing* untuk mendeteksi sinyal yang dibutuhkan atau tidak dalam pemrosesan sehingga fitur yang dihasilkan akan lebih baik.
4. Penggunaan metode transformasi sinyal dalam *preprocessing* sehingga sinyal digital yang dihasilkan akan lebih baik.
5. Penggunaan teknik *filtering, noise reduction, end point detection* sehingga sinyal suara digital yang dihasilkan akan lebih baik dari segi kualitas.
6. Penggunaan alat-alat audio (mikrofon dan *soundcard*) yang lebih baik sehingga data *audio* yang diperoleh akan lebih baik kualitasnya.

DAFTAR PUSTAKA

[ALI-13] Aliyu, Asniar. 2013. *Pengaruh Panjang Runtun Ciri Lpccepstral Terhadap Tingkat Pengenalan Isyarat Tutur Model Markov Tersembunyi.* Sekolah Tinggi Tekologi Nasional Yogyakarta.

[ARD-03] Ardisasmita, M. Syamsa. 2003. *Sitem Kendali Peralatan dengan Perintah Suara menggunakan Model Hidden Markov Model dan Jaringan Syaraf Tiruan.* Risalah Lokakarya Komputasi dalam Sains dan Teknologi Nuklir XIV.

[ARI-05] Aribowo, Candra. 2005. *Perbedaan Prestasi Belajar Huruf Jawa antara Pembelajaran Menggunakan Program SWISH dengan Metode Konvensional pada siswa kelas VII Semester I DI SMP NEGERI 1 BRANGSONG KABUPATEN KENDAL*. Semarang, Universitas Negeri Semarang.

[BIC-03] Bicego, M., U. Castellani, and V. Murino. 2003. *Using Hidden Markov Models and wavelets for face recognition.* In Image Analysis and Processing, 2003. Proceedings. 12th International Conference on, pp. 52-56. IEEE.

[DWI-10] Hermanto, Dwi Kurniawan dan Firdaus Solihin. 2010. *Rancang Bangun Aplikasi Pangabdi Ajisaka Sebagai Solusi Konversi dan Pembelajaran Aksara Jawa Secara Online*. Bandung, Politeknik Telkom.

[HID-06] Hidayatno, Achmad, dan Sumardi. 2006. *Pengenalan Ucapan Kata Terisolasi dengan Metode Hidden Markov Model (HMM) melalui Ekstraksi Ciri Linear Predictive Coding (LPC)*. Semarang, Universitas Diponegoro.

[HLA-12] Hlavac, Vaclav. 2012. *Classifier Performance Evaluation*. Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Replubic.

[MUL-08] Mulyani, dkk. 2008. Penerapan Hidden Markov Model Dalam Clustering Sequence Protein Globin. Yogyakarta. Program Studi Ilmu Komputer, FMIPA, Universitas Gadjah mada.

[MUN-10] Munawar, Badri. 2010. *Pengidentifikasian Kata Dengan Menggunakan Metode Hidden Markov Model (HMM) Melalui Ekstraksi Ciri Linear Predictive Coding (LPC).* Bandung, Universitas Komputer Indonesia.

[NUG-11] Nugraha, Khrisna. 2011. *Aplikasi Perintah Suara Dengan Metode Fast Fourier Transform Dan Devide and Conque Pada Simulasi Rumah Pintar*. Unikom Jakarta.

[NUR-12] Nur Rohman, Sigit, Achmad Hidayatno dan Sumardi. 2012. *Aplikasi Pencirian dengan Linear Predictive Coding untuk Pembelajaran Pengucapan Nama Hewan dalam Bahasa Inggris menggunakan Jaringan Saraf Tiruan Propagasi Balik.* Semarang, Universitas Diponegoro.

[OKT-11] Oktarina, Elly. 2011. *Pembentukan Basis Data Ucapan dalam Bahasa Indonesia dan Pengkodeannya Berdasarkan Linear Predictive Coding (LPC).* Universitas Gunadarma.

[SEP-12] Sepritahara. 2012. *Sistem Pengenalan Wajah (Face Recognition) Menggunakan Hidden Markov Model (HMM).* Jurusan Teknik Elektro Universitas Indonesia, Depok.

[TEG-11] Tegar, Sinung dkk. 2011. *Aplikasi Pengenalan Ucapan Sebagai Pengaktif Peralatan Elektronik.* Semarang, Universitas Diponegoro.

[TON-11] Tono, Budhi Irawan, Astri Novianty. 2011.*Speech to Text Bahasa Inggris Menggunakan Metode Linear Predictive Coding dan Algoritma Genetika*. Institut Teknologi Telkom.

[WIK-14] WIKIPEDIA. http://upload.wikimedia.org/wikipedia/id/a/a9/Aj-ngtmbr.png (Diakses 20 Maret 2014)

LAMPIRAN

Tabel 5.5 Hasil Pengujian Skenario 1 (Aksara = Ha)

No	Kelas = 1	
	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Ha	Benar
6	Ha	Benar
7	Ha	Benar
8	Ha	Benar
Total Benar		**8**

Tabel 5.6 Hasil Pengujian Skenario 1 (Aksara = Na)

No	Kelas = 1	
	Label	Benar/Salah
1	Na	Benar
2	Na	Benar
3	Na	Benar
4	Na	Benar
5	Na	Benar
6	Na	Benar
7	Na	Benar
8	Na	Benar
Total Benar		**8**

Tabel 5.7 Hasil Pengujian Skenario 1 (Aksara = Da)

No	Kelas = 1	
	Label	Benar/Salah
1	Da	Benar
2	Da	Benar
3	Da	Benar
4	Da	Benar
5	Da	Benar
6	Da	Benar
7	Da	Benar
8	Da	Benar
Total Benar		**8**

Tabel 5.8 Hasil Pengujian Skenario 1 (Aksara = Ta)

No	Kelas = 1	
	Label	Benar/Salah
1	Ta	Benar
2	Ta	Benar
3	Ta	Benar
4	Ta	Benar
5	Ta	Benar
6	Ta	Benar
7	Ta	Benar
8	Ta	Benar
Total Benar		**8**

Tabel 5.9 Hasil Pengujian Skenario 1 (Aksara = Pa)

No	Kelas = 1	
	Label	Benar/Salah
1	Pa	Benar
2	Pa	Benar
3	Pa	Benar
4	Pa	Benar
5	Pa	Benar
6	Pa	Benar
7	Pa	Benar
8	Pa	Benar
Total Benar		**8**

Tabel 5.10 Hasil Pengujian Skenario 1 (Aksara = Dha)

No	Kelas = 1	
	Label	Benar/Salah
1	Dha	Benar
2	Dha	Benar
3	Dha	Benar
4	Dha	Benar
5	Dha	Benar
6	Dha	Benar
7	Dha	Benar
8	Dha	Benar
Total Benar		**8**

Tabel 5.11 Hasil Pengujian Skenario 1 (Aksara = Ma)

No	Kelas = 1	
	Label	Benar/Salah
1	Ma	Benar
2	Ma	Benar
3	Ma	Benar
4	Ma	Benar
5	Ma	Benar
6	Ma	Benar
7	Ma	Benar
8	Ma	Benar
Total Benar		**8**

Tabel 5.12 Hasil Pengujian Skenario 1 (Aksara = Ga)

No	Kelas = 1	
	Label	Benar/Salah
1	Ga	Benar
2	Ga	Benar
3	Ga	Benar
4	Ga	Benar
5	Ga	Benar
6	Ga	Benar
7	Ga	Benar
8	Ga	Benar
Total Benar		**8**

Tabel 5.13 Hasil Pengujian akurasi terhadap jumlah kelas (jumlah kelas = 1)

No	Kelas = 1	
	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Ha	Benar
6	Ha	Benar
7	Ha	Benar
8	Ha	Benar
Total Benar		**4**
Akurasi		**100%**

Tabel 5.14 Hasil Pengujian akurasi terhadap jumlah kelas (jumlah kelas = 5)

No	Kelas = 5	
	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Benar
6	Na	Benar
7	Na	Salah
8	Na	Benar
9	Ca	Benar
10	Ca	Benar
11	Ca	Benar
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Benar
16	Ra	Benar
17	Ka	Benar
18	Ka	Benar
19	Ka	Benar
20	Ka	Salah
Total Benar		**17**
Akurasi		**85%**

Tabel 5.15 Hasil Pengujian akurasi terhadap jumlah kelas (jumlah kelas = 10)

No	Kelas = 10	
	Label	**Benar/Salah**
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Benar
6	Na	Benar
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Benar
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah
28	Ta	Benar
29	Sa	Benar
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar

36	Wa	Salah
37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
Total Benar		**26**
Akurasi		**65%**

Tabel 5.16 Hasil Pengujian akurasi terhadap jumlah kelas (jumlah kelas = 15)

	Kelas = 15	
No	**Label**	**Benar/Salah**
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Benar
6	Na	Salah
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Salah
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah
28	Ta	Salah

29	Sa	Salah
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah
37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Benar
47	Dha	Benar
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
Total Benar		**34**
Akurasi		**56.67%**

Tabel 5.17 Hasil Pengujian akurasi terhadap jumlah kelas (jumlah kelas = 20)

No	Kelas = 20	
	Label	**Benar/Salah**
1	Ha	Benar
2	Ha	Benar
3	Ha	Salah
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Salah
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah
28	Ta	Salah
29	Sa	Salah
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar

36	Wa	Salah
37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Salah
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Benar
63	Ma	Benar
64	Ma	Benar
65	Ga	Salah
66	Ga	Benar
67	Ga	Benar
68	Ga	Benar
69	Ba	Benar
70	Ba	Salah
71	Ba	Salah
72	Ba	Benar
73	Tha	Salah
74	Tha	Benar

75	Tha	Benar
76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Salah
80	Nga	Benar
Total Benar		**44**
Akurasi		**55%**

Tabel 5.18 Hasil Pengujian akurasi sinyal yang digunakan (Utuh tanpa dipotong)

No	Label	Benar / Salah
1	Ha	Benar
2	Ha	Salah
3	Ha	Salah
4	Ha	Benar
5	Na	Benar
6	Na	Salah
7	Na	Salah
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Benar
12	Ca	Salah
13	Ra	Salah
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Benar
18	Ka	Salah
19	Ka	Salah
20	Ka	Salah
21	Da	Benar
22	Da	Salah
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Salah
27	Ta	Salah

28	Ta	Salah
29	Sa	Salah
30	Sa	Salah
31	Sa	Salah
32	Sa	Salah
33	Wa	Benar
34	Wa	Benar
35	Wa	Salah
36	Wa	Benar
37	La	Benar
38	La	Benar
39	La	Benar
40	La	Salah
41	Pa	Benar
42	Pa	Salah
43	Pa	Benar
44	Pa	Benar
45	Dha	Salah
46	Dha	Salah
47	Dha	Salah
48	Dha	Salah
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Benar
57	Nya	Salah
58	Nya	Salah
59	Nya	Salah
60	Nya	Salah
61	Ma	Salah
62	Ma	Salah
63	Ma	Salah
64	Ma	Salah
65	Ga	Salah
66	Ga	Salah

67	Ga	Salah
68	Ga	Salah
69	Ba	Salah
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Benar
74	Tha	Benar
75	Tha	Benar
76	Tha	Benar
77	Nga	Benar
78	Nga	Salah
79	Nga	Benar
80	Nga	Salah
Total Benar		**30**
Akurasi		**37.5%**

Tabel 5.19 Hasil Pengujian akurasi sinyal yang digunakan (Dipotong sesuai pengucapan)

No	Label	Benar/Salah
1	Ha	Salah
2	Ha	Salah
3	Ha	Salah
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Salah
12	Ca	Salah
13	Ra	Salah
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Benar
18	Ka	Salah
19	Ka	Salah
20	Ka	Salah
21	Da	Salah
22	Da	Salah
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Salah
27	Ta	Salah
28	Ta	Benar
29	Sa	Salah
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah

37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
41	Pa	Salah
42	Pa	Salah
43	Pa	Benar
44	Pa	Benar
45	Dha	Benar
46	Dha	Salah
47	Dha	Salah
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Salah
54	Ya	Salah
55	Ya	Salah
56	Ya	Benar
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Benar
63	Ma	Benar
64	Ma	Benar
65	Ga	Benar
66	Ga	Benar
67	Ga	Benar
68	Ga	Benar
69	Ba	Salah
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Salah
74	Tha	Salah
75	Tha	Salah

76	Tha	Benar
77	Nga	Benar
78	Nga	Benar
79	Nga	Salah
80	Nga	Benar
Total Benar		**37**
Akurasi		**46.25%**

Tabel 5.20 Hasil Pengujian akurasi sinyal yang digunakan (Dipotong sesuai pengucapan konsonan)

No	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Salah
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Salah
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah

28	Ta	Salah
29	Sa	Salah
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah
37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Salah
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Benar
63	Ma	Benar
64	Ma	Benar
65	Ga	Salah
66	Ga	Benar

67	Ga	Benar
68	Ga	Benar
69	Ba	Benar
70	Ba	Salah
71	Ba	Salah
72	Ba	Benar
73	Tha	Salah
74	Tha	Benar
75	Tha	Benar
76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Salah
80	Nga	Benar
Total Benar		**44**
Akurasi		**55%**

Tabel 5.21 Hasil Pengujian orde LPC = 8

No	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Salah
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Benar
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Salah
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah
28	Ta	Salah
29	Sa	Salah
30	Sa	Benar
31	Sa	Benar
32	Sa	Benar
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah

37	La	Benar
38	La	Benar
39	La	Benar
40	La	Benar
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Salah
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Benar
63	Ma	Benar
64	Ma	Benar
65	Ga	Salah
66	Ga	Benar
67	Ga	Benar
68	Ga	Benar
69	Ba	Benar
70	Ba	Salah
71	Ba	Salah
72	Ba	Benar
73	Tha	Salah
74	Tha	Benar
75	Tha	Benar

76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Salah
80	Nga	Benar
Total Benar		**44**
Akurasi		**55%**

Tabel 5.22 Hasil Pengujian orde LPC = 10

No	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Salah
8	Na	Salah
9	Ca	Benar
10	Ca	Salah
11	Ca	Benar
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Benar
17	Ka	Benar
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Benar
26	Ta	Benar
27	Ta	Salah
28	Ta	Salah
29	Sa	Salah
30	Sa	Salah
31	Sa	Salah
32	Sa	Salah
33	Wa	Salah
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah
37	La	Benar

38	La	Benar
39	La	Benar
40	La	Salah
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Salah
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Salah
52	Ja	Benar
53	Ya	Salah
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Salah
63	Ma	Benar
64	Ma	Salah
65	Ga	Salah
66	Ga	Benar
67	Ga	Benar
68	Ga	Salah
69	Ba	Salah
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Salah
74	Tha	Salah
75	Tha	Salah
76	Tha	Benar

77	Nga	Benar
78	Nga	Benar
79	Nga	Benar
80	Nga	Benar
Total Benar		**36**
Akurasi		**45%**

Tabel 5.23 Hasil Pengujian orde LPC = 12

No	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Salah
6	Na	Salah
7	Na	Salah
8	Na	Salah
9	Ca	Benar
10	Ca	Benar
11	Ca	Benar
12	Ca	Benar
13	Ra	Salah
14	Ra	Salah
15	Ra	Salah
16	Ra	Salah
17	Ka	Salah
18	Ka	Salah
19	Ka	Salah
20	Ka	Salah
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Benar
26	Ta	Benar
27	Ta	Salah
28	Ta	Benar
29	Sa	Salah
30	Sa	Benar
31	Sa	Salah
32	Sa	Salah
33	Wa	Benar
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah

37	La	Salah
38	La	Benar
39	La	Benar
40	La	Salah
41	Pa	Salah
42	Pa	Salah
43	Pa	Salah
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Benar
48	Dha	Benar
49	Ja	Benar
50	Ja	Benar
51	Ja	Benar
52	Ja	Salah
53	Ya	Benar
54	Ya	Benar
55	Ya	Salah
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Benar
60	Nya	Benar
61	Ma	Benar
62	Ma	Salah
63	Ma	Benar
64	Ma	Salah
65	Ga	Salah
66	Ga	Salah
67	Ga	Benar
68	Ga	Salah
69	Ba	Salah
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Salah
74	Tha	Salah
75	Tha	Benar

76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Benar
80	Nga	Benar
Total Benar		**35**
Akurasi		**43.7%**

Tabel 5.24 Hasil Pengujian orde LPC = 14

No	Label	Benar/Salah
1	Ha	Salah
2	Ha	Salah
3	Ha	Salah
4	Ha	Salah
5	Na	Salah
6	Na	Benar
7	Na	Salah
8	Na	Salah
9	Ca	Benar
10	Ca	Salah
11	Ca	Benar
12	Ca	Benar
13	Ra	Benar
14	Ra	Salah
15	Ra	Salah
16	Ra	Benar
17	Ka	Benar
18	Ka	Benar
19	Ka	Salah
20	Ka	Salah
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Salah
26	Ta	Benar
27	Ta	Salah
28	Ta	Benar
29	Sa	Salah
30	Sa	Salah
31	Sa	Benar
32	Sa	Salah
33	Wa	Benar
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah

37	La	Benar
38	La	Benar
39	La	Salah
40	La	Salah
41	Pa	Salah
42	Pa	Salah
43	Pa	Benar
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Benar
48	Dha	Salah
49	Ja	Benar
50	Ja	Salah
51	Ja	Benar
52	Ja	Benar
53	Ya	Benar
54	Ya	Benar
55	Ya	Benar
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Salah
60	Nya	Benar
61	Ma	Benar
62	Ma	Benar
63	Ma	Benar
64	Ma	Salah
65	Ga	Benar
66	Ga	Benar
67	Ga	Benar
68	Ga	Salah
69	Ba	Benar
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Salah
74	Tha	Salah
75	Tha	Benar

76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Benar
80	Nga	Benar
Total Benar		**38**
Akurasi		**47.5%**

Tabel 5.25 Hasil Pengujian orde LPC = 16

No	Label	Benar/Salah
1	Ha	Benar
2	Ha	Benar
3	Ha	Benar
4	Ha	Benar
5	Na	Salah
6	Na	Benar
7	Na	Salah
8	Na	Salah
9	Ca	Benar
10	Ca	Salah
11	Ca	Benar
12	Ca	Benar
13	Ra	Salah
14	Ra	Salah
15	Ra	Salah
16	Ra	Benar
17	Ka	Benar
18	Ka	Benar
19	Ka	Salah
20	Ka	Benar
21	Da	Salah
22	Da	Benar
23	Da	Salah
24	Da	Salah
25	Ta	Benar
26	Ta	Benar
27	Ta	Salah
28	Ta	Benar
29	Sa	Salah
30	Sa	Salah
31	Sa	Benar
32	Sa	Salah
33	Wa	Benar
34	Wa	Salah
35	Wa	Benar
36	Wa	Salah

37	La	Benar
38	La	Benar
39	La	Salah
40	La	Salah
41	Pa	Salah
42	Pa	Benar
43	Pa	Benar
44	Pa	Salah
45	Dha	Benar
46	Dha	Salah
47	Dha	Benar
48	Dha	Benar
49	Ja	Benar
50	Ja	Salah
51	Ja	Benar
52	Ja	Benar
53	Ya	Salah
54	Ya	Benar
55	Ya	Benar
56	Ya	Salah
57	Nya	Salah
58	Nya	Salah
59	Nya	Salah
60	Nya	Benar
61	Ma	Benar
62	Ma	Salah
63	Ma	Benar
64	Ma	Benar
65	Ga	Salah
66	Ga	Salah
67	Ga	Benar
68	Ga	Salah
69	Ba	Benar
70	Ba	Salah
71	Ba	Salah
72	Ba	Salah
73	Tha	Salah
74	Tha	Salah
75	Tha	Benar

76	Tha	Salah
77	Nga	Benar
78	Nga	Benar
79	Nga	Salah
80	Nga	Benar
Total Benar		**41**
Akurasi		**51.25%**