PENGENALAN *CHORD* OTOMATIS MENGGUNAKAN HAAR *WAVELET* DAN JARINGAN SYARAF TIRUAN HOPFIELD

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

oleh:
MUSTIKA MENTARI
0610963033

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENGENALAN *CHORD* OTOMATIS MENGGUNAKAN HAAR *WAVELET* DAN JARINGAN SYARAF TIRUAN HOPFIELD

oleh :

MUSTIKA MENTARI
0610963033 - 96

Telah dipertahankan di depan Majelis Penguji
pada tanggal 6 Juni 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Marji, M.T	**Candra Dewi, S.Kom., M.Sc**
NIP. 196708011992031001	**NIP. 197711142003122001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf A., MSc.
NIP. 196709071992031001

1.

2. LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Mustika Mentari**
NIM	**: 0610963033-96**
Jurusan	**: Matematika**
Program Studi	**: Ilmu Komputer**
Penulis skripsi berjudul	**: Pengenalan *Chord* Otomatis Menggunakan Haar *Wavelet* dan Jaringan Syaraf Tiruan Hopfield**

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 6 Juni 2011
Yang menyatakan,

(Mustika Mentari)
NIM. 0610963033-96

PENGENALAN *CHORD* OTOMATIS MENGGUNAKAN HAAR *WAVELET* DAN JARINGAN SYARAF TIRUAN HOPFIELD

ABSTRAK

Pengenalan *Chord* merupakan proses pengenalan chord secara otomatis berdasarkan kumpulan frekuensi nada tertentu yang dikemas dalam suatu file wav. Pengenalan *Chord* dapat dikategorikan kedalam sub-disiplin ilmu pada sistem cerdas yang merupakan bagian dari masalah pengenalan.

Tujuan penelitian ini adalah mengetahui tingkat akurasi pengenalan *chord* otomatis melalui metode Haar *Wavelet* dan Jaringan Syaraf Tiruan Hopfield.

Pengenalan ini diawali dengan merubah suatu sinyal suara dari domain waktu ke domain frekuensi menggunakan transformasi *wavelet* Haar setelah sebelumnya melalui pengolahan sinyal digital. Ini merupakan suatu cara dari transformasi *wavelet* untuk memproses sinyal tanpa ada batasan jumlah kapasitas menggunakan dekomposisi *Maximal Overlap Discrete Wavelet* (MODWT). Dengan hal ini masalah *down sampling* bisa teratasi. Pemetaan nada dilatih dengan Jaringan Syaraf Tiruan Hopfield yang menghasilkan karakteristik hasil pelatihan mendekati data asli dengan cara yang sederhana dan akurat sesuai dengan jumlah iterasi. Proses terakhir pada pengenalan *chord* adalah pencarian jarak terpendek dengan *Euclidean Distance* sesuai dengan karakteristik 24 jenis *chord* mayor dan minor

Berdasarkan pengujian yang dilakukan pada 30 buah data uji yang disimpan pada suatu *file wav* yang berisi satu jenis *chord* tiap filenya, menghasilkan tingkat akurasi 40% dan tingkat kesalahan 60%.

AUTOMATIC CHORD RECOGNITION USING HAAR WAVELET AND HOPFIELD ARTIFICIAL NEURAL NATWORK

ABSTRACT

Chord recognition is an automatically chord introduction process based on a group of certain frequency that is classified in a sub knowledge discipline in a smart system as a part of identification problem. The recognition is started with the changing of sound signal in a time domain to frequency domain using a kind of wavelet transformation family called "Haar Transformation" after passing digital signal processing.

The purpose of the research is to prove the level of accuracy of chord automatic recognition by using Haar Wavelet and Hopfield Artificial Neural Network methods.

It is the simplest kind of wavelet's that is able to process a signal in an unlimited length signal using decomposition of Maximal Overlap Discrete Wavelet (MODWT). As the result, the down sampling modus can be avoided. The signal pattern of tone is trained frequency signal using Artificial Neural Network Hopfield which produces a characteristics closely to the origin with a simple and accurate adaptation iteration. The final process to recognize chord is Euclidean Distance process to get the minimum distance between 24 major and minor chord characteristics.

Based on the test done on the 30 of data examination, that is kept in a wav file and contains of one file chord in each. It is resulted 40% of accurate data and 60% of fault data.

KATA PENGANTAR

Puji syukur Penulis panjatkan kepada Allah SWT atas segala limpahan rahmat dan hidayah-Nya, sehingga Penulis dapat menyelesaikan skripsi ini sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dalam bidang Ilmu Komputer.

Skripsi ini bertujuan untuk menerapkan sistem cerdas pengenalan *chord* musik dalam file wav sebagai solusi masalah pengidentifikasian *file* wav yang akan diuji.

Pada penyusunan skripsi ini, Penulis ingin mengucapkan terima kasih kepada:

1. Drs. Marji, MT selaku pembimbing utama dan Ketua Program Studi Ilmu Komputer, Jurusan Matematika, FMIPA Universitas Brawijaya atas segala arahan serta bimbingannya dalam penyusunan skripsi ini.
2. Candra Dewi, S.Kom., M.Sc selaku pembimbing pendamping atas segala arahan serta bimbingannya dalam penyusunan skripsi ini.
3. Dr. Abdul Rouf A., MSc selaku Ketua Jurusan Matematika, FMIPA Universitas Brawijaya.
4. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
5. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya yang telah banyak membantu Penulis dalam usaha memperlancar urusan administrasi untuk pelaksanaan penyusunan skrispsi ini.
6. Orang tua Penulis Dra. Tutik Warniati dan Mardjuki, Ir, MT atas segala dukungannya baik berupa materi, motivasi, arahan dan doa restunya kepada Penulis.
7. Kakak dan Kakak Ipar Penulis Marina Candra Dewi, Marlupi Berlianti, Novan Dwi Hartanto, dan Rany Hendryana Sugiarto yang tak henti-hentinya memberikan motivasi dan dukungan.
8. Keponakan pertama Nariva Helga Zulema, yang selalu memberikan senyuman fitrah sebagai penghilang kepenatan Penulis.

9. Seluruh keluarga, khususnya Tante Iin Takim W, dan keluarga yang tak henti-hentinya memberikan motivasi, teladan, dan doa restu.
10. Vierkury Metyopandi yang telah bersedia memberikan doa dan motivasi, serta mendampingi dan membantu dengan penuh kesabaran.
11. Rekan-rekan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan inspirasi, motivasi, dukungan, dan bantuannya kepada Penulis demi kelancaran pelaksanaan penyusunan skripsi ini khususnya Milyun, Siwi L, Christine D, Riezqi, Hermansyah, Ari R, Dwina S, I Gede A, Anisa W, Dinda F, Yanuar, Danang A, Vania, M. Azis, Yuita A (2007), Fahron (2005), Adit (2005), Akbar W, dan Putri F.M.
12. *Cientifico-Choir* dan Keluarga yang telah memberikan banyak pembelajaran dan pengalaman berharga.
13. Sahabat tercinta Yeni S (Statistika 2006), Inosensia A, Dewinta A, Richard, Puspita Rani, Rendiq R, Hardini P, Ceria W, Ria A, Inun, Leonora MM, Deni A, Mustaqbal, Kiki Z atas segala nasehat, dukungan, pengalaman, dan doa yang telah diberikan.
14. Para Pekerja dan tokoh-tokoh dunia yang banyak memberikan inspirasi
15. Semua pihak yang telah membantu dalam penyusunan skripsi ini yang tidak dapat Penulis sebutkan satu per satu.

Penulis sadari bahwa masih banyak kekurangan dalam penyusunan skripsi ini, oleh karena itu Penulis sangat menghargai saran dan kritik yang sifatnya membangun demi perbaikan penulisan dan mutu isi skripsi ini untuk kelanjutan penelitian serupa di masa mendatang.

Semoga skripsi ini dapat bermanfaat bagi semua pembaca. Amin.

Malang, Juni 2011

Penulis

DAFTAR ISI

HALAMAN JUDUL i
HALAMAN PENGESAHAN iii
HALAMAN PERNYATAAN v
ABSTRAK vii
KATA PENGANTAR xi
DAFTAR ISI xiii
DAFTAR GAMBAR xvii
DAFTAR TABEL xix
DAFTAR SOURCECODE xxi
DAFTAR RUMUS xxiii

BAB I PENDAHULUAN
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batas Permasalahan 3
1.4 Tujuan Penelitian 3
1.5 Manfaat Penelitian 3
1.6 Metode Penelitian 4
1.7 Sistematika Penulisan 4

BAB II TINJAUAN PUSTAKA
2.1 Musikologi 7
2.1.1 Tangga Nada 7
2.1.2 *Chord* 7
2.2 Pengolahan Sinyal Digital (*Preprocessing*) 8
2.2.1 KonversiSinyal Digital 8
2.2.2 Struktur File Wav 9
2.2.3 *Pre-emphasis* 13
2.2.4 *Frame Blocking* 14
2.2.5 *Windowing* 14
2.2.6 Transformasi *Wavelet* 15
2.2.6.1 Haar *Wavelet* 17
2.2.6.2 *Maximal Overlap Discrete Wavelet Transform* (MODWT) 20

2.3 *Pitch Class Profil* 23
2.4 Jaringan Syaraf Tiruan..... 24
2.4.1 Jaringan Syaraf Tiruan Hopfield 24
2.5 Pengenalan dengan *Euclidean Distance* 29
2.6 Evaluasi Kinerja 30

BAB III METODOLOGI DAN PERANCANGAN
3.1 Deskripsi Umum Sistem 31
3.2 Batasan Sistem 32
3.3 Pengolahan Sinyal Digital (*Preprocessing*)..... 32
3.3.1 Pembacaan *File* Wav 34
3.3.2 *Pre-emphasis*..... 34
3.3.3 *Frame Blocking* 35
3.3.4 *Windowing* 38
3.3.5 Transformasi MODWT Haar *Wavelet*..... 39
3.3.6 Pengelompokan Frekuensi 42
3.3.7 Pemetaan *Pitch Class Profil*..... 44
3.4 Pelatihan dan Pengenalan..... 46
3.4.1 Pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield..... 46
3.4.2 Pengenalan menggunakan Jarak Terdekat (*Euclidean Distance*)..... 49
3.5 Perhitungan Manual..... 50
3.5.1 Pembacaan *File* Wav 50
3.5.2 Perhitungan *Pre-emphasis*..... 52
3.5.3 *Windowing* dengan *Hamming Window* 54
3.5.4 Transformasi *Wavelet* dengan *Maximal Overlap Discrete Wavelet Transform* (MODWT)..... 57
3.5.5 Pemetaan *Pitch Class Profile* 62
3.5.6 Pelatihan dan Pengenalan..... 63
3.6 Rancangan *User InterFace* Sistem 66
3.6.1 *Form* Pembacaan *File* Wav..... 67

BAB IV HASIL DAN PEMBAHASAN
4.1 Lingkungan Implementasi 69
4.1.1 Lingkungan Perangkat Keras 69
4.1.2 Lingkungan Perangkat Lunak 69
4.2 Struktur Data 69

4.3 Implementasi Program 72
4.3.1 Pembacaan *File Wav* 72
4.3.2 Penggambaran Spektrum Sinyal 75
4.3.3 Pengolahan Sinyal Digital (*Preprocessing*) 79
4.3.3.1 *Pre-emphasis* 79
4.3.3.2 *Frame Blocking* 80
4.3.3.3 *Windowing* 80
4.3.3.4 Transformasi *Wavelet* 81
4.3.3.5 Pengelompokan Frekuensi 83
4.3.3.6 Pemetaan *Pitch Class Profile* 85
4.3.4 Pelatihan dan Pengenalan 87
4.3.4.1 Pelatihan 87
4.3.4.2 Pengenalan 88
4.4 Implementasi *Interface* (Antarmuka) 91
4.5 Pengujian dan Analisis Hasil 93
4.4.1 Pengujian 93
4.4.2 Hasil dan Analisis Pengujian 93

BAB V PENUTUP
5.1 Kesimpulan 97
5.2 Saran 97

DAFTAR PUSTAKA 99


UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

Gambar 2.1 Struktur *File* Wav 10
Gambar 2.2 Detail Struktur *File* Wav 13
Gambar 2.3 *Windowing* pada Sinyal 15
Gambar 2.4 Proses Dekomposisi *High Pass Filter* 17
Gambar 2.5 Haar *Wavelet* 18
Gambar 2.6 Ruang Vektor Dalam Bentuk Sinyal 19
Gambar 2.7 Skema MODWT 23
Gambar 2.8 Topologi JST Hopfield 25
Gambar 2.9 Arsitektur jaringan Hopfield 28
Gambar 2.10 *Euclidean* 30

Gambar 3.1 Diagram Alir Sistem Pengenalan *Chord* 31
Gambar 3.2 *Flowchart* Pengolahan Sinyal Digital 33
Gambar 3.3 *Flowchart Pre-emphasis* 35
Gambar 3.4 *Flowchart Frame Blocking* 37
Gambar 3.5 *Flowchart Windowing* dengan metode Hamming 38
Gambar 3.6 *Flowchart* Transformasi Haar *Wavelet* MODWT 40
Gambar 3.7 *Flowchart* Pembentukan Matriks Transformasi *Wavelet* 41
Gambar 3.8 *Flowchart* Pengelompokan Frekuensi 43
Gambar 3.9 Pitch Class Profile dari sinyal *chord* Am sebelum diskala 44
Gambar 3.10 *Flowchart* Pemetaan *Pitch Class Profile* 45
Gambar 3.11 JST Hopfield dengan 12 inputan 46
Gambar 3.12 *Flowchart* JST Hopfield 48
Gambar 3.13 *Flowchart* Jarak Terdekat (*Euclidean Distance*) 49
Gambar 3.14 *Chart* Hasil Pembacaan Wav (*left*) 51
Gambar 3.15 *Chart* Hasil Pembacaan Wav (*right*) 51
Gambar 3.16 *Chart* Hasil Perhitungan *Pre-emphasis* (*left*) ... 53
Gambar 3.17 *Chart* Hasil Perhitungan *Pre-emphasis* (*right*) 53
Gambar 3.18 *Chart* sesudah *Windowing* (*left*) 56
Gambar 3.19 *Chart* sesudah *Windowing* (*right*) 56
Gambar 3.20 *Chart* Hasil Dekomposisi Pada Koefisien

Skala (*left*)..... 59
Gambar 3.21 *Chart* Hasil Dekomposisi Pada Koefisien Skala (*right*) 59
Gambar 3.22 *Chart* Hasil Dekomposisi Pada Koef. *Wavelet* (*left*) 61
Gambar 3.23 *Chart* Hasil Dekomposisi Pada Koef. *Wavelet* (*right*)..... 62
Gambar 3.24 Pemetaan pada *Pitch Class Profile*..... 62
Gambar 3.25 Gambar User *Interface* Pembacaan File Wav . 67
Gambar 4.1 Aplikasi Pengenalan *Chord*..... 92

DAFTAR TABEL

Tabel 2.1 Frekuensi not (dalam Hz) 8
Tabel 2.2 Struktur Penyusun *File* Wav 11
Tabel 2.3 Ruang Vektor Dalam Bentuk Sinyal 20
Tabel 2.4 Banyak selang Nol pada Filter Sirkuler 22
Tabel 2.5 Matriks yang Menunjukkan Bobot Sinaptik *Neuron* 26
Tabel 3.1 Representasi Data *File* Wav 2 *channel* dalam Variabel 34
Tabel 3.2 *Pitch Class Profile* dari sinyal *chord* Am setelah diskala 44
Tabel 3.3 Daftar Kelas Pengenalan *Chord* 47
Tabel 3.4 Data Hasil Pembacaan wav 50
Tabel 3.5 Hasil Perhitungan *Pre-emphasis* 52
Tabel 3.6 Perhitungan *Window* dengan metode Hamming . 54
Tabel 3.7 Data Setelah *Windowing* 55
Tabel 3.8 Perhitungan koefisien skala level 1 57
Tabel 3.9 Perhitungan koefisien skala level 2 58
Tabel 3.10 Hasil Dekomposisi Koefisien Skala 58
Tabel 3.11 Perhitungan koefisien *wavelet* level 1..... 60
Tabel 3.12 Perhitungan koefisien *wavelet* level 2..... 60
Tabel 3.13 Hasil Dekomposisi Koefisien *Wavelet*..... 61
Tabel 3.14 Hasil Pemetaan PCP setelah diskala..... 63
Tabel 3.15 Matriks Asal pada *Chord* A# Mayor 64
Tabel 3.16 Hasil Perkalian Skala dengan Transposenya..... 64
Tabel 3.17 Hasil Pelatihan Jaringan Syaraf Tiruan Hopfield..... 65
Tabel 3.18 Hasil Pengenalan Menggunakan *Euclidean Distance*..... 66
Tabel 3.19 Hasil Uji Coba..... 68
Tabel 4.1 Rincian atribut Wave*Header*, *Channel*, dan Data..... 71
Tabel 4.2 Hasil uji coba tahap pertama 94
Tabel 4.3 Hasil uji coba tahap kedua 95
Tabel 4.4 Hasil uji coba tahap ketiga 96

UNIVERSITAS BRAWIJAYA

DAFTAR SOURCECODE

Sourcecode 4.1 Struktur Data Pembacaan File *Wav* 70
Sourcecode 4.2 Pembacaan File *Wav* 73
Sourcecode 4.3 Penggambaran Spektrum Sinyal 76
Sourcecode 4.4 *Pre emphasis* 79
Sourcecode 4.5 *Frame Blocking* 80
Sourcecode 4.6 *Windowing* 81
Sourcecode 4.7 Transformasi *Wavelet* 82
Sourcecode 4.8 Pengelompokan Frekuensi 84
Sourcecode 4.9 Pemetaan *Pitch Class Profile* 86
Sourcecode 4.10 Pelatihan dengan JST Hopfield 88
Sourcecode 4.11 Pengenalan dengan *Euclidean Distance* 89

UNIVERSITAS BRAWIJAYA

DAFTAR RUMUS

Rumus 2.1 Filter berordo rendah 13
Rumus 2.2 *Output Filter* berordo rendah 13
Rumus 2.3 Pencarian nilai awal *Frame*..... 14
Rumus 2.4 *Hamming Window*..... 14
Rumus 2.5 *Windowing* 15
Rumus 2.6 *Persamaan Wavelet*..... 15
Rumus 2.7 Integral kuagrat fungsi..... 16
Rumus 2.8 Persamaan Haar *Wavelet*..... 18
Rumus 2.9 Persamaan Filter Skala 18
Rumus 2.10 Persamaan Filter *Wavelet*..... 18
Rumus 2.11 Persamaan *Filter Wavelet* MODWT 20
Rumus 2.12 Persamaan Filet Skala MODWT..... 21
Rumus 2.13 Bentuk matriks *Filter Wavelet* 21
Rumus 2.14 Persamaan dalam matriks Orthogonal 21
Rumus 2.15 Deret *Filter Wavelet* 22
Rumus 2.16 Matriks level kedua*Filter WAvelet*..... 22
Rumus 2.17 Transformasi skala..... 23
Rumus 2.18 Prosedur MODWT 23
Rumus 2.19 Bobot matriks Hopfield 27
Rumus 2.20 Vektor Pola Hopfield..... 28
Rumus 2.21 Inisialisasi matrik Penyimpanan Hopfield 28
Rumus 2.22 Inisialisasi Pola *Input*..... 29
Rumus 2.23 Uji pola Konvergen 29
Rumus 2.24 *Determine Activation* (Sinyal *Output*) 29
Rumus 2.25 Jatak Terdekat (*Euclidean Distance*)..... 30
Rumus 2.26 Tingkat Akurasi..... 30
Rumus 2.27 Tingkat kesalahan..... 30


UNIVERSITAS BRAWIJAYA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Musik menjadi tidak pernah habis untuk dibahas. Istilah musik menjadi familiar di semua lapis masyarakat di seluruh dunia. Salah satu kasus pada bidang kesehatan menjelaskan bahwa jenis musik klasik dapat menjadi media penenang untuk bayi yang lahir secara prematur. Pengaruh ini dapat dilihat dari ekspresi wajah sang bayi yang tidak lagi terlihat menderita dan detak jantung yang berangsur-angsur normal (Anurogo, 2007). Kasus lain menerangkan bahwa musik mempunyai pengaruh baik pada perkembangan otak anak. Salah satu penelitian yang dilakukan oleh University of Toronto dengan melibatkan 144 anak berumur 6 tahun yang mengikuti kursus musik selama satu tahun. Hasilnya anak-anak tersebut mengalami perkembangan otak yang semakin baik, dan kemampuan matematika serta IQ secara keseluruhan yang semakin meningkat (Melinda, 2010). Berdasarkan kasus-kasus di atas cukup membuktikan bahwa musik dengan jenis yang beragam memberikan banyak mafaat pada berbagai bidang kehidupan.

Musik dibentuk oleh *chord* yang disusun secara harmoni dan selaras. Kombinasi *chord* tersebut telah melahirkan banyak jenis aliran lagu yang memiliki beragam kombinasi penyusunan sesuai dengan ide dan kreatifitas manusia yang membuatnya.

Teknologi yang semakin maju dan berkembang saat ini memungkinkan pengenalan *chord* dilakukan secara otomatis, sehingga kalangan yang awam akan musik pun bisa mengetahui *chord* tertentu tanpa harus rumit untuk mencari secara manual.

Sekarang ini sudah banyak dilakukan beberapa penelitian mengenai pengenalan *chord*. Salah satunya yaitu penelitian yang dilakukan oleh Drs. Miftahul Huda, dkk (2010), yaitu "Konversi nada-nada akustik menjadi chord menggunakan *Pitch Class Profile*". Penelitian tersebut menggunakan *Fast Fourier Transform* dan memberikan hasil rata-rata pengenalan *chord* sebesar 80-95%. Penelitian lain yang dilakukan oleh Andika Bandung Putra, dkk (2007), yaitu "*Speech Recognition* Menggunakan Gabor *Wavelet* dan

Jaringan Syaraf Tiruan *Backpropagation* untuk Sistem Keamanan Berbasis Suara". Penelitian tersebut menggunakan transformasi *Wavelet* yang merupakan perbaikan dari *fast fourier transform*. Hasil akhir penelitian tersebut menyimpulkan bahwa tingkat keberhasilan sistem untuk mengenali suara dengan benar adalah 85,8%.

Transformasi Fourier hanya memberikan informasi tentang frekuensi suatu sinyal sedangkan transformasi *Wavelet* memberikan informasi tentang kombinasi skala dan frekuensi. Hal ini yang menyebabkan transformasi Fourier hanya dapat menangkap informasi apakah suatu sinyal memiliki frekuensi tertentu ataukah tidak, tapi tidak dapat menangkap dimana frekuensi tersebut terjadi. *Wavelet* mempunyai ketelitian yang lebih baik dari FFT karena membawa informasi skala dan frekuensi sekaligus. Berdasarkan hasil penelitian tersebut maka dilakukan penelitian dengan judul "**Pengenalan *Chord* Otomatis Menggunakan Haar *Wavelet* Dan Jaringan Syaraf Tiruan Hopfield**". Pengenalan *chord* dilakukan dengan menggunakan transformasi *Wavelet* yang merupakan perbaikan dari FFT. Sedangkan untuk proses pelatihan digunakan metode Jaringan Syaraf Tiruan Hopfield karena JST ini mempunyai jenis pelatihan dengan pengawasan (*supervised)* dan mempunyai perhitungan dengan cara yang lebih sederhana, sesuai dengan arsitektur Hopfield yang menjelaskan *output* suatu *neuron* menjadi *input* bagi *neuron* yang lain.

1.2 Rumusan Masalah

Rumusan masalah yang akan dibahas pada skripsi ini adalah :

1. Bagaimana pengimplementasian pengolahan sinyal digital agar diperoleh data yang bisa diproses pada tahap pelatihan dan pengenalan.
2. Bagaimanakah proses transformasi Haar *Wavelet* agar diperoleh sinyal dalam domain frekuensi.

3. Bagaimana proses pelatihan mengunakan jaringan syaraf tiruan Hopfield disertai pengenalan menggunakan perhitungan jarak terdekat (*Euclidean Distance*).
4. Bagaimana tingkat kesesuaian pengenalan *chord* dengan transformasi *Wavelet* dan jaringan syaraf tiruan Hopfield.

1.3 Batas Permasalahan

Adapun batas permasalahan pada skripsi ini adalah sebagai berikut :

1. *Input* dari *user* hanya berupa audio yang dikemas dalam format wav.
2. Pengenalan dibatasi hanya untuk 24 tipe *chord*, yaitu 12 *chord* mayor dan 12 *chord* minor.
3. *Output* sistem berupa *chord* hasil pengenalan.
4. Waktu maksimal untuk setiap file wav adalah 90 detik.
5. Ketercapaian target luaran dilakukan dengan perhitungan prosentase kesesuaian tiap *frame* rangkaian *chord* hasil pengenalan dengan rangkaian *chord* yang sebenarnya pada sistem pengenalan.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai dalam skripsi ini adalah :

1. Mengimplementasikan teknik pengolahan sinyal digital untuk mengenali *chord* secara otomatis.
2. Mentransformasikan data wav dengan Haar *Wavelet*.
3. Melakukan pelatihan menggunakan jaringan syaraf tiruan Hopfield disertai perhitungan jarak terdekat (*Euclidean Distance*) sebagai cara pengenalan.
4. Menganalisa tingkat kesesuaian pada pengenalan *chord* menggunakan transformasi *Wavelet* dan jaringan syaraf tiruan Hopfield.

1.5 Manfaat Penelitian

Manfaat skirpsi ini mencakup beberapa bidang. Dalam bidang ilmu komputer dan sejenisnya bisa didapatkan implementasi ilmu secara nyata. Hal ini juga nyata terlihat dalam bidang seni musik karena mempermudah mengetahui *chord* musik.

Sebagai masyarakat umum ataupun masyarakat yang bisa dikatakan awam dengan dunia musik bisa didapatkan implementasi umum *chord* musik yang dikemas secara praktis dan mudah untuk diketahui.

1.6 Metode Penelitian

Metodologi yang digunakan dalam penelitian ini adalah sebagai berikut:

1. Studi Literatur
2. Mempelajari teori-teori
3. Pendefinisian dan Analisis Masalah
4. Perancangan dan Implementasi Sistem
5. Uji Coba dan Analisis Hasil Implementasi

1.7 Sistematika Penulisan

Sistematika penulisan pada laporan penelitian ini dibagi dalam lima bab sebagai berikut :

BAB I: PENDAHULUAN
Bab ini menjelaskan mengenai latar belakang penelitian, rumusan masalah, batas permasalahan, tujuan penelitian, manfaat penelitian, metode penelitian, dan sistematika penulisan laporan penelitian.

BAB II: TINJAUAN PUSTAKA
Bab ini memberikan gambaran mengenai dasar teori yang berkaitan dengan penelitian, diantaranya pengetahuan tentang

musikologi, pengolahan sinyal suara, transformasi Haar *Wavelet*, dan jaringan syaraf tiruan Hopfield.

BAB III: METODE DAN PERANCANGAN
Bab ini berisi metode-metode dan langkah-langkah yang dilakukan dalam penelitian. Beberapa hal yang dibahas adalah: deskripsi umum sistem, batasan sistem, pengolahan sinyal suara, pengenalan dengan jaringan syaraf tiruan Hopfield, contoh penerapan algoritma, dan rancangan antarmuka sistem.

BAB IV: HASIL DAN PEMBAHASAN
Bab ini menjelaskan secara lebih detail implementasi program dari rancangan yang telah dijelaskan pada Bab III dan hasil uji coba berupa persentase keakuratan yang didapat dari perbandingan jumlah data yang berhasil dideteksi dan dikenali dengan jumlah data yang gagal dideteksi dan dikenali.

BAB V: PENUTUP
Bab ini berisi kesimpulan-kesimpulan dari semua yang telah diuraikan dalam bab-bab sebelumnya dan saran-saran untuk perbaikan serta pengembangan penelitian selanjutnya.


UNIVERSITAS BRAWIJAYA

BAB II
TINJUAN PUSTAKA

2.1 Musikologi

2.1.1 Tangga Nada

Tangga nada merupakan susunan berjenjang dari nada-nada pokok suatu sistem nada, mulai dari salah satu nada dasar sampai dengan nada oktafnya, misalnya do, re, mi, fa, so, la, dan si yang biasanya dilambangkan dengan angka secara berurutan 1, 2, 3, 4 5 6, dan 7. Nada dasar tersebut biasa disebut dengan tangga nada mayor. Tangga nada mayor termasuk dalam skala diatonik yang disusun oleh oleh delapan not dengan interval berurutan adalah 1, 1, 1/2, 1, 1, 1, 1/2 (Pasaribu, 2010).

2.1.2 *Chord*

Chord/Akor merupakan tiga nada atau lebih yang dimainkan secara bersamaan maupun terpisah dan mewakili nada tersebut. Secara umum, *chord* dibagi kedalam 3 kelompok yaitu (Sambu, 2008):

a. *Chord* Mayor
   Chord mayor adalah kumpulan nada diatonik mayor yang dibunyikan secara bersamaan, dalam satu waktu yang sama. Nada diatonik mayor sendiri merupakan nada-nada dengan urutan jarak antar nada 1-1-1/2-1-1-1-1/2.

b. *Chord* Minor
   Chord minor adalah beberapa nada diatonik minor yang dibunyikan secara bersamaan, dalam satu waktu yang sama. Nada diatonik minor sendiri merupakan nada-nada dengan urutan jarak antar nada 1-1/2-1-1-1/2-1-1.

c. *Chord* Substitusi

Chord substitusi dapat dilihat berdasarkan ke interval atau jarak nada yang menyusunnya.

Semua not penyusun c*hord* mayor ataupun *chord* minor mempunyai frekuensi tertentu yang telah melalui proses pengukuran. Berikut ini adalah kumpulan frekuensi not yang ada pada tabel 2.1.

Tabel 2.1 Frekuensi not (dalam Hz)
(http://www.seventhstring.com/resources/notefrequencies.html)

Not	Oktaf								
	1	2	3	4	5	6	7	8	9
C	32.703	65.406	130.81	261.63	523.25	1046.5	2093	4186	8372
C#	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9	8869.8
D	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6	9397.3
D#	38.891	77.782	155.56	311.13	622.25	1244.5	2489	4978	9956.1
E	41.203	82.407	164.81	329.63	659.26	1318.5	2637	5274	10548
F	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7	11175
F#	46.249	92.499	185	369.99	739.99	1480	2960	5919.9	11840
G	48.999	97.999	196	392	783.99	1568	3136	6271.9	12544
G#	51.913	103.83	207.65	415.3	830.61	1661.2	3322.4	6644.9	13290
A	55	110	220	440	880	1760	3520	7040	14080
A#	58.27	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6	14917
B	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1	15804

2.2 Pengolahan Sinyal Digital (*Preprocessing*)

2.2.1 Sinyal Digital

Menurut pengertian umum, sinyal adalah suatu besaran yang merupakan fungsi waktu, ruang, atau beberapa variabel. Suatu sinyal mempunyai suatu informasi yang dapat dibaca dan diolah. Informasi tersebut diantaranya adalah amplitudo, frekuensi, perbedaan fase, dan gangguan akibat *noise* (Tanudjaja, 2007).

Sinyal dapat digolongkan menjadi dua jenis, yaitu sinyal kontinyu (*analogue*) dan sinyal diskrit (*digital*). Pengolahan sinyal merupakan suatu operasi matematika sehingga diperoleh suatu informasi yang berguna. Pengolahan sinyal dapat dilakukan secara kontinyu ataupun diskrit. Dewasa ini ternyata pengolahan sinyal secara *digital* lebih banyak dilakukan karena beberapa kelebihan sebagai berikut:

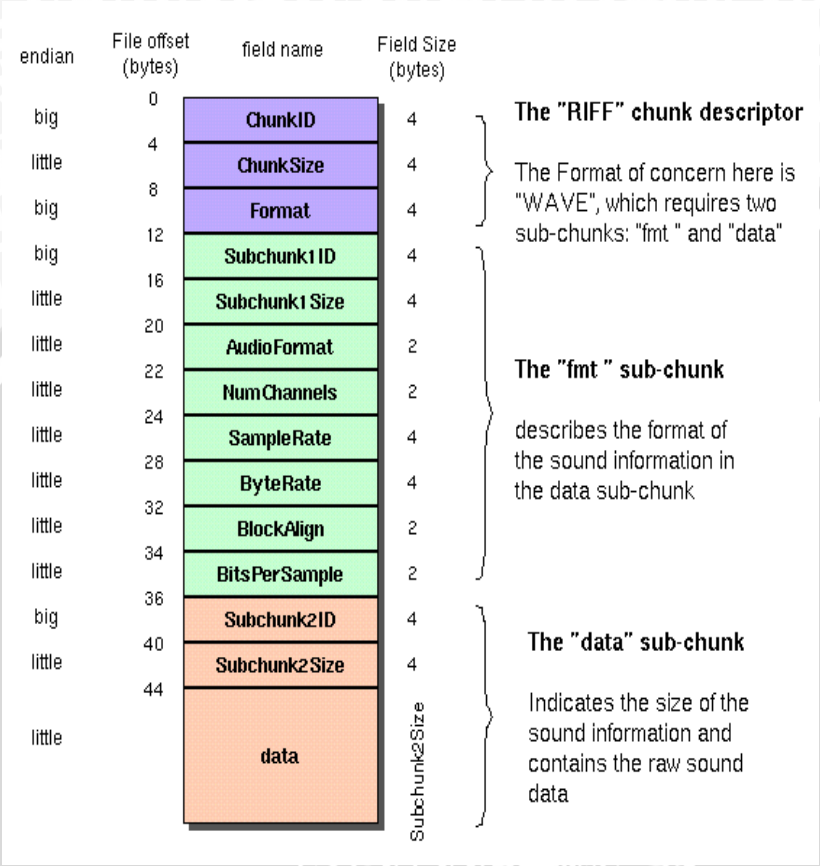
- Sinyal *digital* lebih mudah menyimpan banyak hasil pengolahan. Misalkan saja menggunakan media penyimpanan seperti *flash memory*, CD/DVD, ataupun *hard disk.*
- Sinyal *digital* lebih kebal terhadap *noise*, karena bekerja dengan level tegangan logika "1" dan "0".
- Sinyal *digital* lebih kebal terhadap perubahan *temperature*.
- Sinyal *digital* lebih mudah untuk diproses.

Meskipun mempunyai banyak kelebihan, tetapi sinyal *digital* juga mempunyai beberapa kelemahan sebagai berikut:

- Beberapa informasi akan hilang karena pembulatan kuantisasi dan *filtering* saat pengembalian bentuk menjadi sinyal analog.
- Pemrosesan sinyal *digital* memerlukan proses yang lebih lama dibandingkan dengan sinyal analog.

2.2.2 Struktur File Wav

File wav merupakan salah satu format audio data yang umum dipakai oleh berbagai aplikasi. Wav memiliki *header* berisi informasi yang berkaitan dengan data audio di dalamnya. *Header* ini diperlukan untuk proses pembacaan data audio pada aplikasi yang menjalankannya sehingga sangat penting untuk menjaga agar *header* ini tidak berubah. Struktur file wav ditunjukkan pada gambar 2.1 sedangkan struktur penyusunan file wav ditunjukkan pada tabel 2.2.



Gambar 2.1 Struktur File Wav
(https://ccrma.stanford.edu/courses/422/projects/WaveFormat/)

Tabel 2.2 Struktur Penyusun File Wav
(https://ccrma.stanford.edu/courses/422/projects/WaveFormat/)

Offset	Size	Name	Description
0	4	**ChunkID**	Terdiri dari kata "RIFF" yang berada pada kode ASCII (0x52494646 dalam bentuk *big-endian*)
4	4	**ChunkSize**	36 + SubChunk2Size, atau lebih tepatnya: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) ini merupakan besar seluruh file dalam byte dikurangi 8 byte untuk 2 field yang tidak termasuk dalam hitungan: ChunkID and ChunkSize
8	4	**Format**	Terdiri dari kata "WAVE" (0x57415645 dalam bentuk *big-endian*)
12	4	**Subchunk1ID**	Terdiri dari kata "fmt " (0x666d7420 dalam bentuk *big-endian*)
16	4	**Subchunk1Size**	16 untuk jenis PCM.
20	2	**AudioFormat**	PCM = 1 (*Linear quantization*) Nilai lebih dari 1 yang mengindikasika *file Wave* kompresi

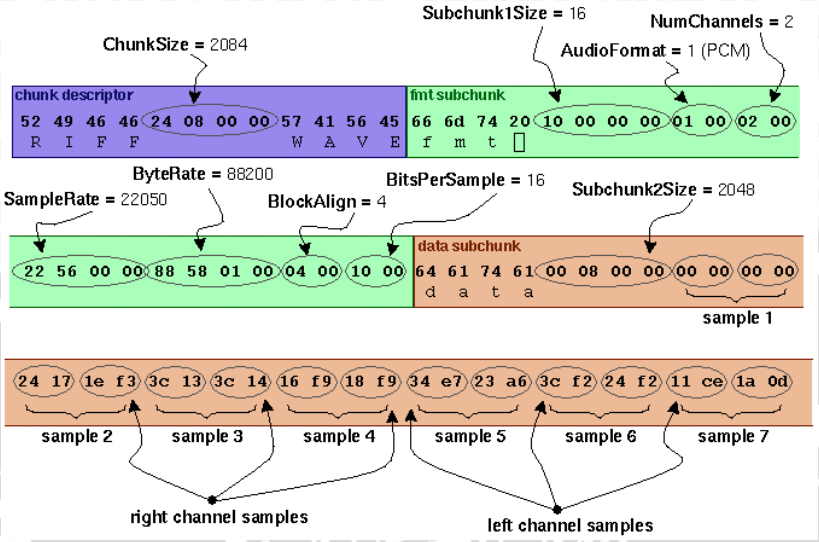
Tabel 2.2 Struktur Penyusun File Wav (Lanjutan)

Offset	Size	Name	Description
22	2	**NumChannels**	Mono = 1, Stereo = 2, dan seterusnya
24	4	**SampleRate**	8000, 44100, dan seterusnya.
28	4	**ByteRate**	== SampleRate * NumChannels * BitsPerSample/8
32	2	**BlockAlign**	== NumChannels * BitsPerSample/8 Jumlah *byte* untuk satu sampel termasuk semua *channel*.
34	2	**BitsPerSample**	8 bits = 8, 16 bits = 16, dan setersunya
36	4	**Subchunk2ID**	Terdiri dari kata "data" (0x64617461 dalam bentuk *big-endian*).
40	4	**Subchunk2Size**	== NumSamples * NumChannels * BitsPerSample/8
44	*	**Data**	*Data Sound* sebenarnya

Berikut ini adalah contoh dengan ukuran awal 72 *bytes* file wav yang terdiri dari bilangan hexadecimal:

52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00 22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00 00 24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d

(https://ccrma.stanford.edu/courses/422/projects/WaveFormat/)



Gambar 2.2 Detail Struktur File Wav

2.2.3 *Pre-emphasis*

Filter *Pre-emphasis* dilakukan setelah sinyal melalui proses *sampling*. Proses *sampling* dilalui dengan menentukan frekuensi maksimum. Selain itu penentuan *sample rate* didapat dari dua kali maksimum frekuensi.

Tujuan dilakukannya filter pada *pre-emphasis* adalah untuk mendapatkan bentuk *spectral* frekuensi sinyal yang lebih halus. Langkah *pre-emphasis* dari sinyal suara yang telah diubah menjadi sinyal *digital* pada proses pembacaan file wav, *s(n)* dilewatkan pada sebuah *filter* yang berorde rendah.

$$H(Z) = 1 - \tilde{a}.z^{-1} \qquad 0.9 \leq a \leq 1 \tag{2.1}$$

Dengan demikian *output* dari rangkaian *filter* tersebut, *s̃(n)* adalah sebagai berikut:

$$\tilde{s}(n) = s(n) - \tilde{a}\,.\,s\,(n-1) \tag{2.2}$$

2.2.4 *Frame Blocking*

Frame blocking merupakan proses yang membagi *voice* menjadi beberapa bagian yang mempercepat proses komputasi. Hasil dari *frame blocking* merupakan sinyal terpotong.

Masing-masing sinyal hasil dari *framing* adalah sinyal terpotong yang *discontinue*. Sinyal *discontinue* ini akan dilanjutkan dalam proses *Windowing* (Basuki, Huda, dan Amalia, 2006).

Pembagian sinyal dalam *frame-frame* ini dilakukan mulai dari awal sinyal dengan rumus 2.3. N adalah jumlah total data dengan l adalah posisi *frame* untuk mencari nilai n sebagai nilai awal *frame*.

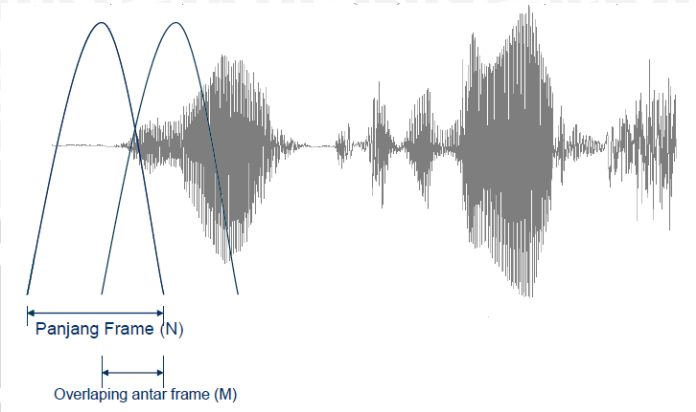
$$n = Nx\,(l - 1) + 1 \qquad (2.3)$$

2.2.5 *Windowing*

Sinyal terpotong yang *discontinue* setelah proses *frame blocking* dikalikan dengan fungsi *window* agar menjadi sinyal yang *continue*. Fungsi *windowing* yang digunakan dalam penelitian ini adalah *window* Hamming. Fungsi Hamming dapat membuat data pada awal *frame* dan akhir *frame* mendekati nilai 0 dengan baik. Setelah melalui proses ini sinyal akan menjadi *continue* (Basuki, Huda, dan Amalia, 2006). Berikut ini adalah gambar sinyal yang telah melalui proses *windowing*, ditunjukkan pada gambar 2.3.

Hamming *Window* = $w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$ (2.4)

dimana $0 \leq n \leq N - 1$



Gambar 2.3 *Windowing* pada Sinyal (http://file.upi.edu/Direktori/Presentasi_vq_rev.pdf)

Tujuan *windowing* adalah mengurangi kebocoran *spectral*. *Window w(n)* dan sinyal tiap bagian adalah *x(n)* maka sinyal hasil proses *windowing* ini adalah sebagai berikut:

$$\tilde{x}(n) = x(n).w(n) \quad (2.5)$$

$Dimana, \quad 0 \leq n \leq N-1$

2.2.6 Transformasi *Wavelet*

Wavelet merupakan sebutan untuk suatu gelombang kecil yang naik dan turun pada periode waktu tertentu (Percival dan Walden, 2000). Secara umum *wavelet* adalah fungsi-fungsi yang mempunyai sifat seperti persamaan:

$$\int_{-\infty}^{\infty} \Psi(u)\, du = 0 \quad (2.6)$$

Yaitu jika diintegralkan pada (-∞, ∞) hasilnya akan sama dengan nol dan integral dari kuadrat fungsi Ψ(.) akan sama dengan satu, seperti persamaan:

$$\int_{-\infty}^{\infty} \Psi^2(u)\, du = 1 \qquad (2.7)$$

Dalam tranformasi *wavelet* terdapat dua fungsi, yaitu fungsi skala (*father wavelet*) dan fungsi *wavelet* Ψ (*mother wavelet*) yang keduanya disebut *parent wavelets*. Semua keluarga *wavelet* tersebut dapat digunakan dalam analisis *wavelet*. Perbedaannya hanya dalam menentukan penskalaan sinyal dan bagaimana *wavelet* didefinisikan.

Terdapat dua macam transformasi *wavelet*, yaitu *Continues Wavelet Transform* (CWT) dan Discrete Wavelet Transform (DWT). CWT dirancang bekerja pada data deret waktu dengan waktu kontinyu (Percival dan Walden, 2000), pergesarannya dapat dilakukan secara kontinyu (Daubechies, 1992). Berbeda dengan CWT, DWT dirancang pada data diskrit atau integer, dengan perhitungan skala dan pergeserannya hanya dilakukan pada sekelompok skala tertentu.

Skala *wavelet* dengan frekuensi mempunyai hubungan sebagai berikut:

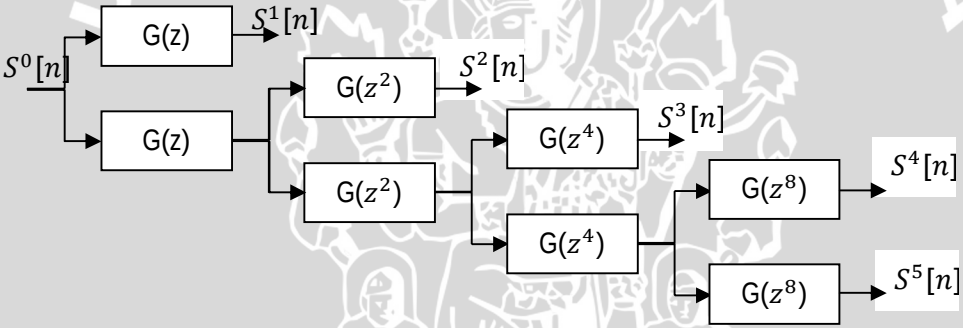
a. Nilai skala kecil (*compressed wavelet*) menyebabkan perubahan koefisien yang menyatakan frekuensi tinggi.
b. Nilai skala besar (*stretched wavelet*) menyebabkan perubahan koefisien yang menyatakan frekuensi rendah.

Berikut ini merupakan beberapa tahap untuk melakukan analisa *wavelet :*

a. Tahap pertama analisis *wavelet* adalah menentukan tipe *wavelet* atau *mother wavelet* yang akan digunakan karena mengingat banyaknya tipe wavelet yang bisa digunakan.
b. Tahap selanjutnya adalah membentuk basis *wavelet* yang akan digunakan untuk mentransformasikan sinyal.

Dibandingkan dengan jenis transformasi yang lain, *wavelet* memiliki kemampuan untuk menganalisis suatu data dalam domain waktu dan domain frekuensi sekaligus secara simultan.

Analisis data dengan jenis transformasi ini dilakukan dengan mendekomposisikan suatu sinyal ke dalam komponen-komponen frekuensi yang berbeda-beda untuk selanjutnya dapat dianalisis sesuai dengan skala resolusinya atau *level* dekomposisinya pada masing-masing komponen frekuensi tersebut.



Gambar 2.4 Proses Dekomposisi *High Pass Filter*

2.2.6.1 Haar *Wavelet*

Haar *Wavelet* pertama kali ditemukan olah Alfred Haar pada tahun 1910 (Chahyati, 2003). Haar *wavelet* didefinisikan dengan persamaan sebagai berikut:

\

$$h(x) = \begin{cases} 1 & for\ 0 < x \leq 1/2 \\ -1 & for\ 1/2 < x \leq 1 \\ 0 & for\ x \leq 0\ or\ 1 < x \end{cases} \quad (2.8)$$

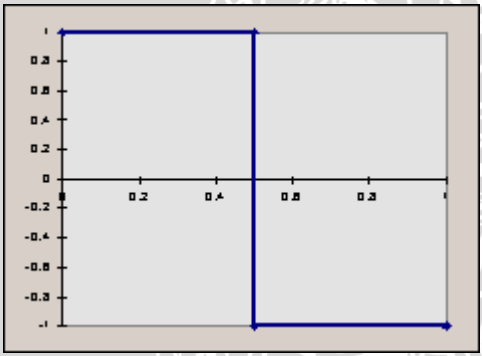
Persamaan filter skala untuk Haar dituliskan dengan :

$$\phi(t) = \sqrt{2}(g_0\phi(2t) + g_1\phi(2t-1)) \quad (2.9)$$

Persamaan filter *wavelet* untuk Haar dituliskan dengan :

$$\psi(t) = \sqrt{2}(g_0\phi(2t) - g_1\phi(2t+1)) \quad (2.10)$$

Penggambaran Haar *wavelet* dijelaskan pada gambar 2.5.



Gambar 2.5 Haar *Wavelet*

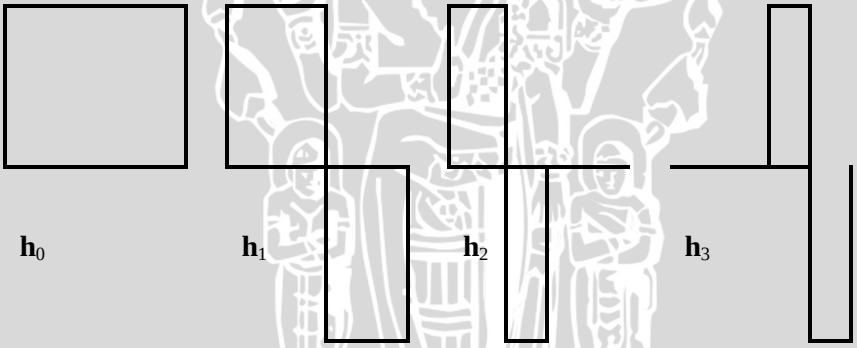
Untuk lebih jelas tentang Haar *wavelet*, permisalan sederhana adalah dengan menggunakan basis orthonormal sebagai berikut:

$$\mathbf{v}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{v}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Haar *wavelet* juga merentang ruang vektor 4 dimensi dengan vektor-vektor basis sebagai berikut:

$$\mathbf{h}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{h}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \mathbf{h}_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{h}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

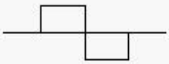
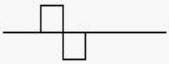
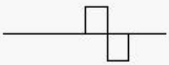
Penggambaran ruang vektor tersebut dalam bentuk sinyal dijelaskan pada gambar 2.6.



Gambar 2.6 Ruang Vektor Dalam Bentuk Sinyal

Untuk lebih jelasnya, berikut ini adalah ruang vektor dalam bentuk sinyal yang lebih detail pada tabel 2.3 :

Tabel 2.3 Ruang Vektor Dalam Bentuk Sinyal

Scaling Function		$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
Mother wavelet		$\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$
Mother wavelet yang didilasikan		$\begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$
Mother wavelet yang didilasikan dan digeser		$\begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$

2.2.6.2 Maximal Overlap Discrete Wavelet Transform (MODWT)

Salah satu hasil modifikasi transformasi *wavelet* diskrit adalah *Maximal Overlap Discrete Wavelet Transform* (MODWT). MODWT mempunyai keuntungan dapat mengeliminasi adanya pengurangan data menjadi setengah data asal. Hal ini memudahkan karena syarat DWT yang harus memenuhi kelipatan 2^n.

Jika terdapat suatu data N, maka transformasi MODWT akan menghasilkan vektor kolom $w_1, w_2, \ldots w_{J0}$ dan $v_1, v_2, \ldots v_{Jo}$ yang masing-masing berukuran N. Vektor W_J berisi koefisien *wavelet* MODWT, dan vektor V_J berisi koefisien skala MODWT. MODWT menggunakan algoritma piramida untuk efisiensi perhitungan. Sinyal X sebagai koefisien penghalus dan detil diperoleh secara *iterative* dengan mengalikan X dengan *filter* skala/*low pass* (g) dan *filter wavelet*/*high pass* (h) (Suhartono, 2009).

Suatu *filter wavelet* MODWT harus memenuhi persamaan:

$$\sum_{l=0}^{L-1} \tilde{h}_1 = 0 \qquad \sum_{l=0}^{L-1} \tilde{h}_{1^2} = \frac{1}{2} \qquad \sum_{l=-\infty}^{\infty} \tilde{h}_1 \tilde{h}_{1+2m} \quad (2.11)$$

Suatu filter skala MODWT harus memenuhi persamaan:

$$\sum_{l=0}^{L-1} \tilde{g}_1 = 0 \qquad \sum_{l=0}^{L-1} \tilde{g}_{1^2} = \frac{1}{2} \qquad \sum_{l=-\infty}^{\infty} \tilde{g}_1 \tilde{g}_{1+2m} = 0 \quad (2.12)$$

Formulasi MODWT ditujukan untuk mendefinisikan transformasi yang bersifat seperti DWT, akan tetapi tidak mengalami kesulitan dari sensitifitas DWT. Sensitifitas ini mengenai *downsampling* dari *output filter wavelet* dan *filter* skala pada masing-masing step dari algoritma piramida. Didefinisikan A merupakan matrik NxN yang berisi *filter* sirkuler $\tilde{g}_l$ untuk MODWT dan B merupakan matriks NxN yang berisi filter $\tilde{h}_l$.

Pada *level* pertama didefinisikan B_1 sebagai matriks NxN, sehingga didapatkan $w_1 = B_1 x$. Dengan *analogue* definisi untuk A_1, didapatkan $v_1 = A_1 x$. Bentuk matriks B_1 dapat dilihat pada persamaan 2.13 :

$$B_1 = \begin{bmatrix} \tilde{h}_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & \tilde{h}_2 & \tilde{h}_1 \\ \tilde{h}_1 & \tilde{h}_0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & \tilde{h}_2 \\ \tilde{h}_2 & \tilde{h}_1 & \tilde{h}_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & \tilde{h}_2 & \tilde{h}_1 & \tilde{h}_0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & \tilde{h}_3 & \tilde{h}_2 & \tilde{h}_1 & \tilde{h}_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & \tilde{h}_2 & \tilde{h}_1 & \tilde{h}_0 \end{bmatrix} \quad (2.13)$$

Pada MODWT langkah pertama ditulis pada persamaan dbawah ini dalam matriks orthogonal:

$$\begin{bmatrix} w_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} B_1 \\ A_1 \end{bmatrix} x = P_1 x \quad (2.14)$$

filter sirkuler $\{\tilde{h}_l : l = 0, \dots, L-1\}$ dan dengan lebar $2^{j-1}(L-1) + 1$ dengan deret,

$$\underbrace{\tilde{h}_0, 0, \ldots, 0}_{2^{j-1}-1\ zeroes}, \underbrace{\tilde{h}_1, 0, \ldots, 0}_{2^{j-1}-1\ zeroes}, \ldots, \underbrace{\tilde{h}_{L-2}, 0, \ldots, 0}_{2^{j-1}-1\ zeroes}, \tilde{h}_{L-1} \quad (2.15)$$

Sesuai dengan persamaan 2.15, banyaknya nilai nol pada selang *filter* $\tilde{g}$ untuk *filter* $\tilde{h}$ dihitung sesuai tabel 2.4.

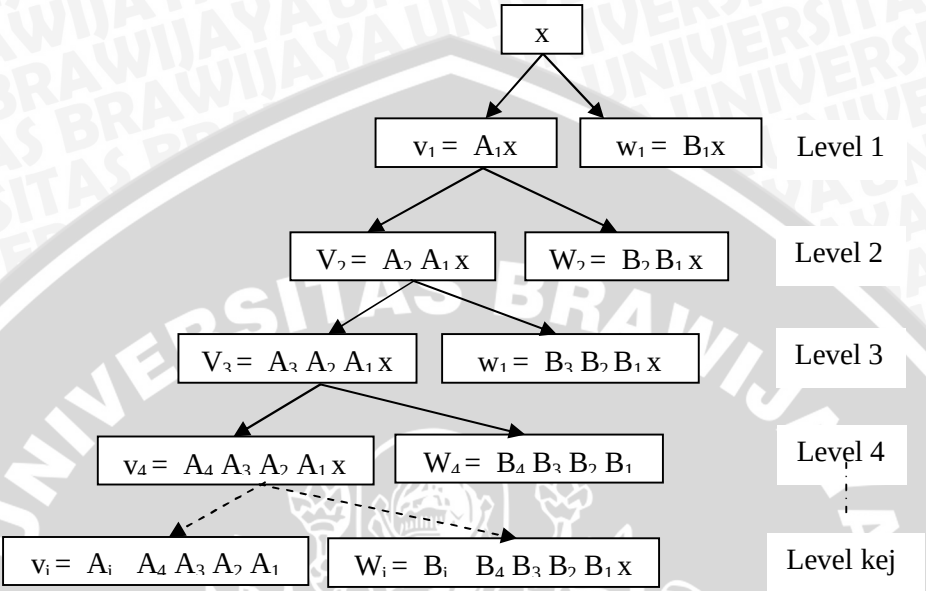
Tabel 2.4. Banyak selang Nol pada *Filter* Sirkuler.

Level (j)	$2^{j-1}-1$
1	0
2	1
3	3
4	7
…	…

Pada *level* kedua dari matriks B_2 sebagai matriks NxN dimisalkan N=12, maka didapatkan persamaan 2.16.

$$B_2 = \begin{matrix} \tilde{h}_0 & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_2 & 0 & \tilde{h}_1 & 0 \\ 0 & \tilde{h}_0 & 0 & \ldots & 0 & 0 & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_2 & 0 & \tilde{h}_1 \\ \tilde{h}_1 & 0 & \tilde{h}_0 & & 0 & 0 & 0 & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_2 & 0 \\ : & : & : & \ldots & : & : & : & : & : & : & : & : & : \\ : & : & : & \ldots & : & : & : & : & : & : & : & : & : \\ 0 & 0 & 0 & & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_1 & 0 & \tilde{h}_0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_1 & 0 & \tilde{h}_0 & 0 \\ 0 & 0 & 0 & & 0 & 0 & 0 & 0 & \tilde{h}_3 & 0 & \tilde{h}_1 & 0 & \tilde{h}_0 \end{matrix} \quad (2.16)$$

Algoritma piramida pada gambar 2.7 mengilustrasikan jika suatu data x didekomposisi dengan filter *wavelet* dan *filter* skala akan menghasilkan koefisien *wavelet* dan koefisien skala.



Gambar 2.7 Skema MODWT (Popoola, Ahmad,S, dan, K, 2007)

Skema MODWT ditulis dalam bentuk matriks, maka transformasi v_{j-1} dari w_j dan v_j menggunakan matriks B_j dan A_j berukuran NxN dapat ditulis :

2.3 *Pitch Class Profile*

Setelah sinyal melalui proses transformasi, maka nilai-nilai deteksi puncak telah diketahui dan kemudian diklasifikasikan melalui *Pitch Class Profile* (PCP). PCP terdiri dari 12 macam nada kromatik. Dari sini maka penggolongan sinyal berdasarkan frekuensi akan tampak.

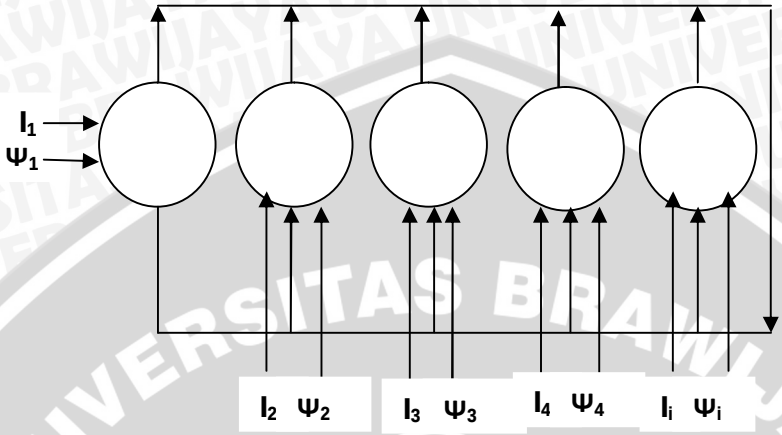
Dengan kata lain, PCP merupakan hasil pemetaan frekuensi yang bersesuaian dengan tiap nada pada tangga nada kromatik dari suatu data yang merupakan sinyal dalam domain frekuensi (Lee dan Slaney, 2006).

2.4 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan (*artificial neural networks*) atau disingkat JST sistem komputasi yang mempunyai arsitektur dan operasi dari pengetahuan tentang sel syaraf biologi di dalam otak. JST yang merupakan cabang ilmu multidisiplin yang mencoba meniru cara kerja otak makhluk hidup khususnya sel syaraf (*neuron*) digambarkan sebagai model matematis dan komputasi untuk aproksimasi nonlinear, klasifikasi data, *cluster* dan regresi parametik. Maka, dalam hal ini kecerdasan JST dianalogikan sama dengan kecerdasan manusia. JST dapat menyelesaikan persoalan yang rumit dengan menggunakan algoritma secara konvensional. Pada saat melakukan proses belajar, JST dapat memodifikasi tingkah laku sesuai keadaan lingkungan. Setelah melalui proses latihan menggunakan algoritma pelatihan pada JST, respon sebuah jaringan dapat menjadi tidak peka terhadap perubahan minor dari masukannya sampai suatu tingkat tertentu. (Kristanto, 2004)

2.4.1 Jaringan Syaraf Tiruan Hopfield

Jaringan Hopfield dibuat oleh John Hopfield dari California *Institute of Technology* pada tahun 1982. Dalam jaringan ini semua *neuron* saling berhubungan penuh. *Neuron* yang satu mengeluarkan *output* dan kemudian menjadi *input* bagi semua *neuron* yang lain. Proses penerimaan dan pengiriman sinyal antar *neuron* dilakukan secara *feedback* terus menerus sampai pada kondisi stabil. Dalam model diskritnya, jaringan Hopfield bobot sinaptiknya menggunakan vektor biner dimensi n atau $a\{0,1\}^n$. Lambang n merupakan *neuron* dan jaringannya terdiri dari n(n-1) interkoneksi dua jalur. Penjelasan secara otomatis bisa dilihat pada matriks simetris nxn dengan diagonal utama bernilai 0, jadi setiap *neuron* tidak memberi *input* kepada dirinya sendiri. Gambar topologi jaringan Hopfield dijelaskan pada gambar 2.8 sebagai berikut :



Gambar 2.8 Topologi JST Hopfield
(Kristanto, 2004)

Keterangan:
I_i = external *input*
Ψ_i = bias

Neuron pada jaringan syaraf tiruan hopfield mengeluarkan suatu *output* dan akhirnya dijadikan *input* untuk *neuron* lain. *Output* setiap simpul menjadi umpan balik ke *input* simpul lain melalui bobot koneksi W_{ij} yang tetap. Selanjutnya pola yang tidak dikenal dimasukkan ke dalam jaringan hanya satu kali pada saat waktu nol. Setelah itu, jaringan melakukan iterasi sampai pada kondisi *konvergen*, yaitu kondisi dimana *output* yang sudah tidak berubah lagi untuk iterasi selanjutnya. *Output* setelah jaringan *konvergen* inilah yang disebut dengan *output* JST.

Berikut ini adalah contoh penyelesaian menggunakan jaringan syaraf tiruan Hopfield :

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Matriks simetris 4x4 di atas menghasilkan jaringan Hopfield dengan 9 *neuron*, yaitu *neuron* A sampai I, dijabarkan pada tabel 2.5.

Tabel 2.5 Matriks yang menunjukkan bobot sinaptik *neuron* (Kristanto, 2004)

Dari \ Ke	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
A	0	-1	1	-1	1	1	1	-3	3
B	-1	0	-3	-1	1	1	-3	1	-1
C	1	-3	0	1	-1	-1	3	-1	1
D	-1	-1	1	0	-3	1	1	1	-1
E	1	1	-1	-3	0	-1	-1	-1	1
F	1	1	-1	1	-1	0	-1	-1	1
G	1	-3	3	1	-1	-1	0	-1	1
H	-3	1	-1	1	-1	-1	-1	0	-3
I	3	-1	1	-1	1	1	1	-3	0

Aplikasi matriks di atas dapat dijelaskan sebagai berikut :

Mulai	**A**	**b**
A=1	A=1	A=1
B=1	B==>0	B=0
C=1	C==>1	C=1
D=1	D==>0	D=0
E=1	E==>0	E==>1
F=1	F==>0	F=0
G=1	G==>1	G=1
H=1	H==>0	H=0
I=1	I==>1	I=1

Berdasarkan Hopfield, bobot sinaptik dari setiap *neuron* dibangkitkan melalui operasi logika dari vektor-vektor biner dalam jaringan tersebut.

α = 101 010 101
β = 110 011 001
γ = 010 010 010

Bentuk bipolar dari vektor dimensi di atas adalah :

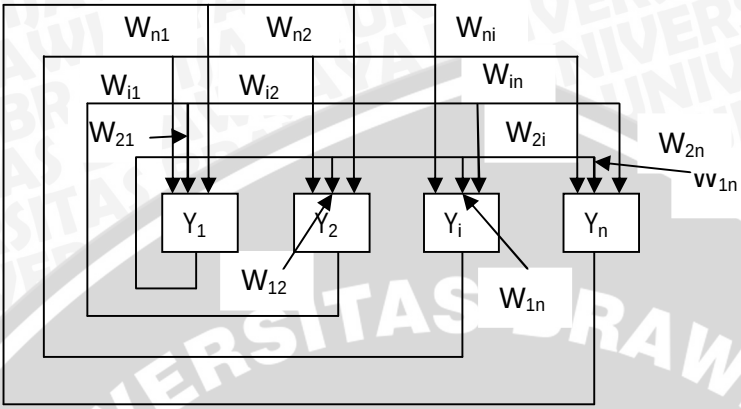
α' = 1-11 -11-1 1-11
β' = 11-1 -111 -1-11
γ' = -11-1 -11-1 -11-1

Perhitungan bobot matriks dilakukan dengan mengalikan setiap vektor dengan dirinya sendiri dan kemudian dijumlahkan :

$$W = \alpha' \alpha' + \beta' \beta' + \gamma' \gamma' \quad (2.17)$$

Bobot sinaptik dapat diperoleh setelah memberikan nilai 0 pada diagonal utama matriks W. Nilai dari setiap matriks merupakan bobot yang dikarenakan pada proses pengiriman sinyal dari sumbu vertikal ke sumbu horizontal. Pada kasus di atas diperoleh bahwa sinyal dari C ke B melewati bobot sinaptik -3 sedangkan sinyal dari F ke D melewati bobot +1. Kondisi setiap *neuron* dirubah menjadi 0 atau 1 sesuai ketentuan. Proses ini kemudian berlanjut sampai kondisi yang stabil yang telah disesuaikan.

Tampilan arsitektur JST Hopfield, dapat dilihat pada gambar 2.9 di bawah ini. *Output* setiap seimpul pada Hopfield diumpan balikkan ke *input* dari simpul lainnya melalui bobot koneksi W_{ij} yang tetap. Nilai W_{ij} mula-mula diinisialisasi menggunakan algoritma Hopfield untuk seluruh pola yang masuk. Pola yang tidak dikenal dimasukkan hanya satu kali pada saat kondisi waktu nol. Selanjutnya jaringan akan beriterasi sampai *output* konvergen.



Gambar 2.9 Arsitektur Jaringan Hopfield
(Kristanto, 2004)

Berikut ini adalah contoh hopfield yang diterapkan pada pola angka dari 0 sampai 9 :

1. Menciptakan vektor pola yang ke-p.

   $S(p) = ((S_1(p), S_2(p),\ldots,S_i(p))$ (2.18)

   Keterangan: p adalah jumlah pola. Dalam kasus di atas ada 10 buah pola dari angka 9 sampai 10.

2. Menciptakan matriks koneksi sebagai tempat pola contoh (angka 0-9) yang kemudian disimpan dalam suatu konstanta. Inisialisasi matriks W untuk menyimpan pola

$$W_{ij} = \begin{vmatrix} \sum_{p=1}^{10} S_i(p) * S_j(p) & i \neq j \\ 0 & i = j \end{vmatrix} \quad (2.19)$$

Ket :
p = jumlah pola
W_{ij} = bobot koneksi dari simpul i ke simpul j

S_i (p) = elemen in dari pola s yang hanya memiliki +1 atau -1
S_j (p) = elemen in dari pola s yang hanya memiliki +1 atau -1

3. Lakukan langkah ini sampai pola konvergen.
   a. Untuk tiap vektor x (x1, x2, …, xn),
   b. Menginisialisasi pola *input* yang tidak diketahui,

$$Y_i = X_i \qquad (i=1..n) \qquad (2.20)$$

   c. Kerjakan langkah b sampai semua pola diuji,
   d. Lakukanlah perhitungan sebagai berikut :

$$y\_in_i = x_i + \sum_{j=1}^{n} h_j(w_{ij}) \qquad (2.21)$$

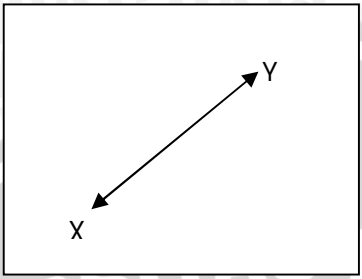
   e. *Determine Activation* (Sinyal *Output*),

$$y_i = \begin{Bmatrix} 1 \;\; jika \;\; y\_in_i \geq 0 \\ -1 \;\; jika \;\; y\_in_i < 0 \end{Bmatrix} \qquad (2.22)$$

   f. Hasil dari *output* Y_i berlaku untuk semua pola lain,
   g. Uji pola sampai konvergen.

2.5 Pengenalan dengan *Euclidean Distance*

Euclidean distance merupakan salah satu metode pengenalan dengan mengukur jarak terpendek atau paling kecil. Metode ini melakukan pengukuran jarak pada garis lurus (*straight line*) antara titik X (X1,X2,...Xn) dan titik Y (Y1,Y2,...Yn) yang ditunjukkan pada gambar 2.10.



Gambar 2.10. *Euclidean*

Rumus *Euclidean* berkembang sesuai keadaan ruang. Metode ini menghitung arah suatu data uji di hitung jaraknya dengan data asal dengan rumus 2.23 (Clifford Gower, John. 2006).

$$d_E(x, y) = \sqrt{(x_1 - y_1)^2} \qquad (2.23)$$

2.6 Evaluasi Kinerja

Akhir dari proses pegenalan *chord* adalah evaluasi kinerja untuk mengetahui keakuratan pengenalan yang telah dilakukan. Dalam penelitian ini tingkat akurasi digunakan untuk menghitung nilai keakuratan pengenalan *chord* setelah proses pengenalan dengan jaringan syaraf tiruan Hopfield selesai dilakukan. Persamaan yang digunakan untuk menghitung tingkat akurasi dan tingkat kesalahan dapat dilihat seperti pada persamaan 2.24 dan persamaan 2.25 (Wibisono, 2005).

$$Tingkat\ akurasi = \frac{jumlah\ pengenalan\ yang\ akurat}{jumlah\ total\ pengenalan} \times 100\% \qquad (2.24)$$

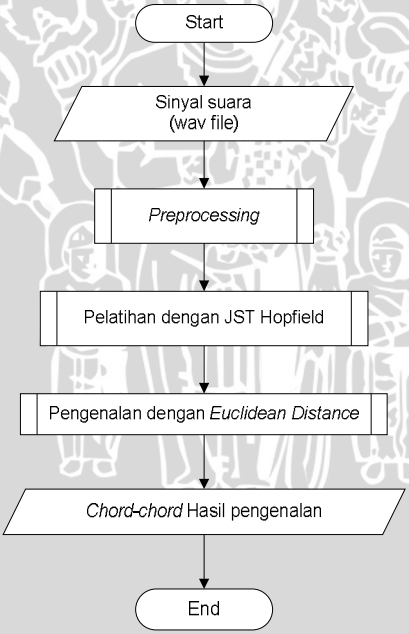
$$Tingkat\ kesalahan = 100\% - tingkat\ akurasi \qquad (2.25)$$

BAB III
METODOLOGI DAN PERANCANGAN

3.1 Deskripsi Umum Sistem

Dalam skripsi ini, penelitian yang dibuat adalah sistem pengenalan *chord* pada sebuah lagu baik pada jenis *mono* ataupun *stereo*. Setelah mengalami beberapa proses, maka *output* dari sistem ini adalah berupa *chord* hasil pengenalan.

Sistem ini mempunyai dua tahap proses. Tahap pertama pada penelitian ini dilakukan pengolahan sinyal digital (*preprocessing*) dengan masukan sinyal berformat wav yang diambil dalam bentuk pola yang akhirnya digunakan untuk proses pelatihan dengan metode jaringan syaraf tiruan Hopfield, dan kemudian diukur jarak terpendek dengan chord asli menggunakan *Euclidean Distance*.



Gambar 3.1 Diagram Alir Sistem Pengenalan *Chord*

3.2 Batasan Sistem

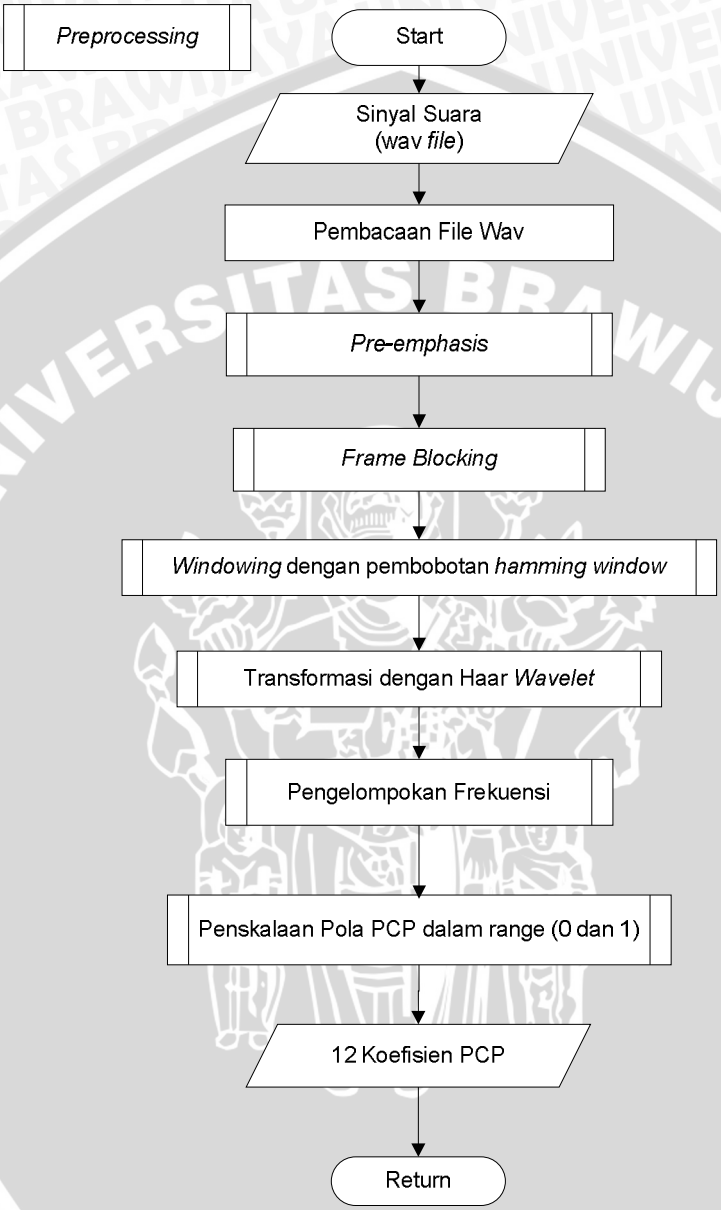
Sistem yang akan dibuat ini memiliki batasan-batasan sebagai berikut:

a. *Input* sistem hanya berupa file wav dengan format wav yang mempunyai *sample rate* 44100Hz.
b. Pengenalan dibatasi hanya untuk 12 *chord* mayor dan 12 *chord* minor.
c. Waktu maksimal untuk setiap file wav adalah 90 detik.
d. *Output* sistem berupa *chord* hasil pengenalan.

3.3 Pengolahan Sinyal Digital (*Preprocessing*)

Tahap pertama yaitu pengolahan sinyal digital yang memiliki beberapa tahap yang harus dilalui. Pengolahan sinyal digital awal dilakukan dengan sinyal suara dengan format wav. Kemudian data dan informasi pada file wav diambil melalui proses pembacaan. Data yang didapat disimpan dalam *array* yang kemudian diproses untuk tahap *pre-emphasis* untuk pengurangan *noise*, dan dilanjutkan dengan proses *frame blocking* serta *windowing* dengan metode pembobotan *hamming window.*

Setelah sinyal masukan sudah dibagi-bagi dalam bentuk *frame* kemudian proses dilanjutkan ke proses transformasi dengan menggunakan transformasi *wavelet*. *Output* dari proses transformasi selanjutnya di petakan dalam PCP. Setelah melalui proses pemetaaan dalam PCP, didapatkan 12 koefisien PCP dalam skala 0 dan 1, koefisien ini digunakan untuk proses penelitian pada tahap kedua. Proses pengolahan tersebut dapat dijelaskan pada gambar 3.2.



Gambar 3.2 *flowchart* Pengolahan Sinyal Digital

3.3.1 Pembacaan *File* Wav

Sinyal suara alam bentuk wav mempunyai kumpulan-kumpulan informasi yang telah disediakan pada beberapa bagian. Pada *header file* mengandung ketentuan-ketentuan yang diantaranya berupa *sample rate*, *bit* per *second*, *byte* per *sample*. Setelah melakukan pembacaan pada bagian-bagian *file* wav maka informasi-informasi tersebut disimpan pada variabel global yang akan digunakan pada proses selanjutnya.

Data yang ada pada penelitian ini adalah data suara *mono* yang berarti data dengan 1 *channel* suara ataupun data suara *stereo* yang berarti 2 *channel* suara. Jadi, untuk data suara *mono sample* suara dimasukkan ke dalam *array* tempData [i]. Sedangkan untuk sampel suara *stereo,* sampel suara ganjil dianggap sebagai *left channel* dan genap dianggap sebagai *right channel* yang disimpan dalam *array* tempDataLeft [i] dan tempDataRight [i]. Representasi data *file* wav dalam variabel dijelaskan pada tabel 3.1.

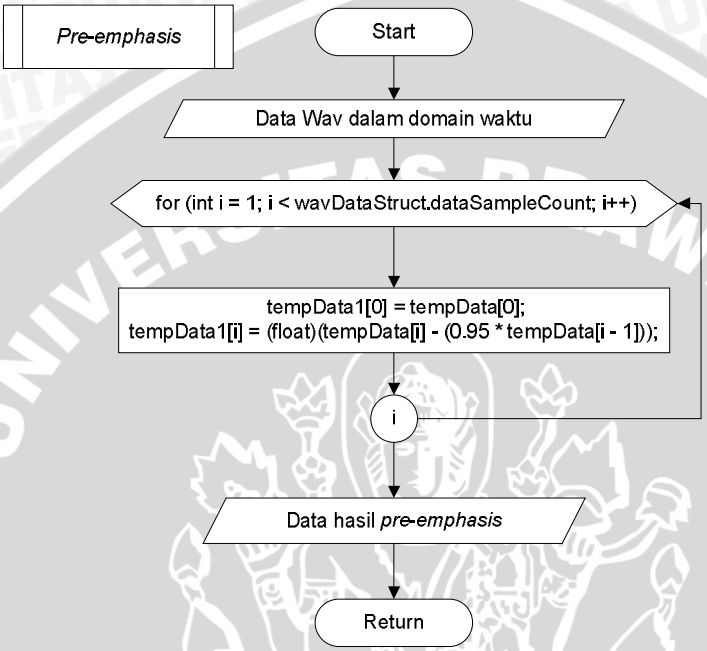
index	0	1	2	…	N
left sample	12	15	28	…	292
right sample	12	15	28	…	292

Tabel 3.1 Representasi Data *File* Wav 2 *channel* dalam Variabel

3.3.2 *Pre-emphasis*

Sinyal suara yang telah diubah menjadi sinyal digital pada proses pembacaan *file* wav, *s(n)* dilewatkan pada sebuah filter yang berorde rendah. Rangkaian *pre-emphasis* yang digunakan dalam penelitian ini adalah sebuah sistem orde satu yaitu a=0.95 ini dikarenakan pengaruh *pre-emphasis* yang tidak mengubah terlalu jauh keseluruhan data.

Proses yang terjadi pada saat *pre-emphasis* ditunjukkan ada gambar 3.3.



Gambar 3.3 *Flowchart Pre-emphasis*

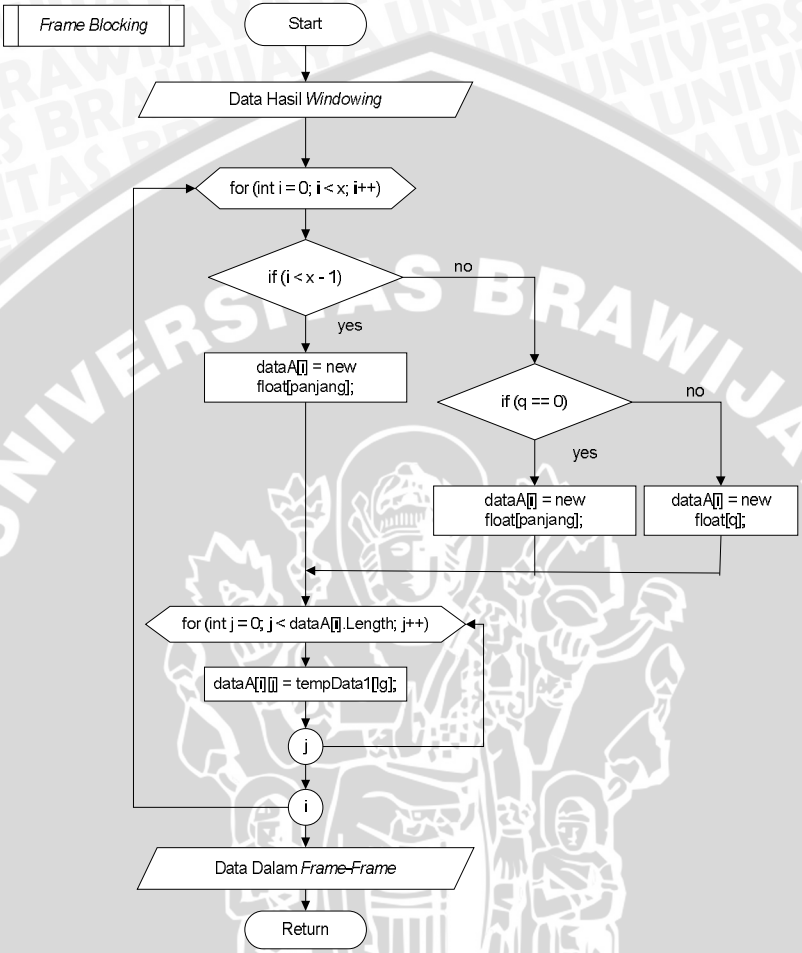
3.3.3 *Frame Blocking*

Setelah melalui proses *pre-emphasis* dilakukan suatu proses *frame blocking* yang membagi file berukuran panjang ke dalam suatu bagian-bagian tertentu sebagai upaya untuk mempercepat proses komputasi. Dalam penelitian ini sinyal masukan dibagi kurang lebih dalam 0.18 ms tiap *frame*, jadi ada 8192 data *sample* pada tiap *frame*, Perhitungan ini berdasarkan sampel *rate* 44100 Hz dan waktu 1 sekon (1000ms).

waktu tiap *frame* = (8192/44100) * 1000
= 0.18 ms

Pengambilan nilai panjang frame 8192 data didasarkan sesuai dengan panjang file wav yang sebesar yaitu maximal 1.5 menit.

Sinyal yang sudah dipotong tersebut merupakan sinyal *discontinue* yang kemudian dilanjutkan pada proses *windowing*. *Frame blocking* lebih jelas ditunjukkan pada gambar 3.4.

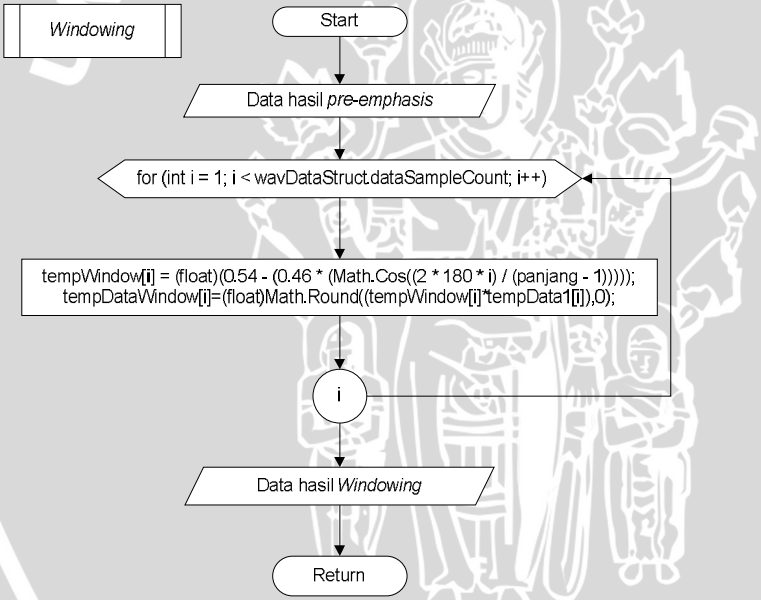


Gambar 3.4 *Flowchart Frame Blocking*

Variable x adalah jumlah total panjang data dibagi panjang *frame*, q adalah sisa pembagian jumlah total panjang data dengan panjang *frame*, dan *array* dataA[i] sebagai media penyimpanan data tiap-tiap *frame*.

3.3.4 *Windowing*

Proses *windowing* akan menghilangkan efek *discontinue* yang ada pada *frame blocking*. Dalam *windowing* diambil suatu bagian yang mewakili untuk kemudian diproses untuk dianalisa dengan kata lain *windowing* menghilangkan sinyal-sinyal yang tidak dibutuhkan. Jenis *windowing* yang digunakan dalam penelitian ini adalah *windowing* dengan metode pembobotan *hamming window*. Pada tahap ini fungsi *window* yang disimpan pada array tempWindow[i]. Kemudian tiap *frame* dikalikan dengan fungsi *window* dan akan menghasilkan suatu sinyal yang mendekati nol pada awal dan akhir *frame*. Penjelasan *windowing* dengan *hamming* dapat dilihat pada gambar 3.5 di bawah ini:



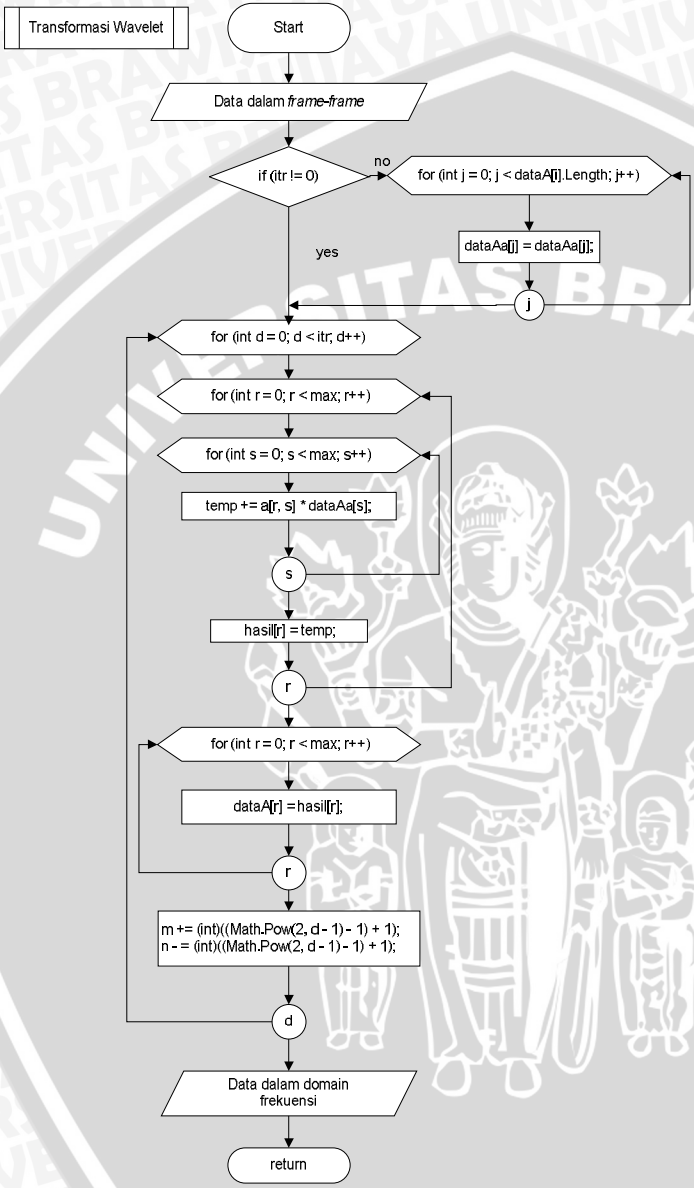
Gambar 3.5 *Flowchart Windowing* dengan metode Hamming

3.3.5 Transformasi MODWT Haar *Wavelet*

Proses transformasi dilakukan untuk mengubah koefisien dalam domain waktu pada proses *windowing* menjadi domain frekuensi. Hal ini dilakukan dengan proses dekomposisi menggunakan MODWT Haar *Wavelet*. Koefisian yang digunakan dalam penelitian ini adalah koefisien wavelet yang lebih peka terhadap frekuensi suara yang lebih tinggi. Penentuan jumlah iterasi dengan *wavelet* didapat dengan rumus:

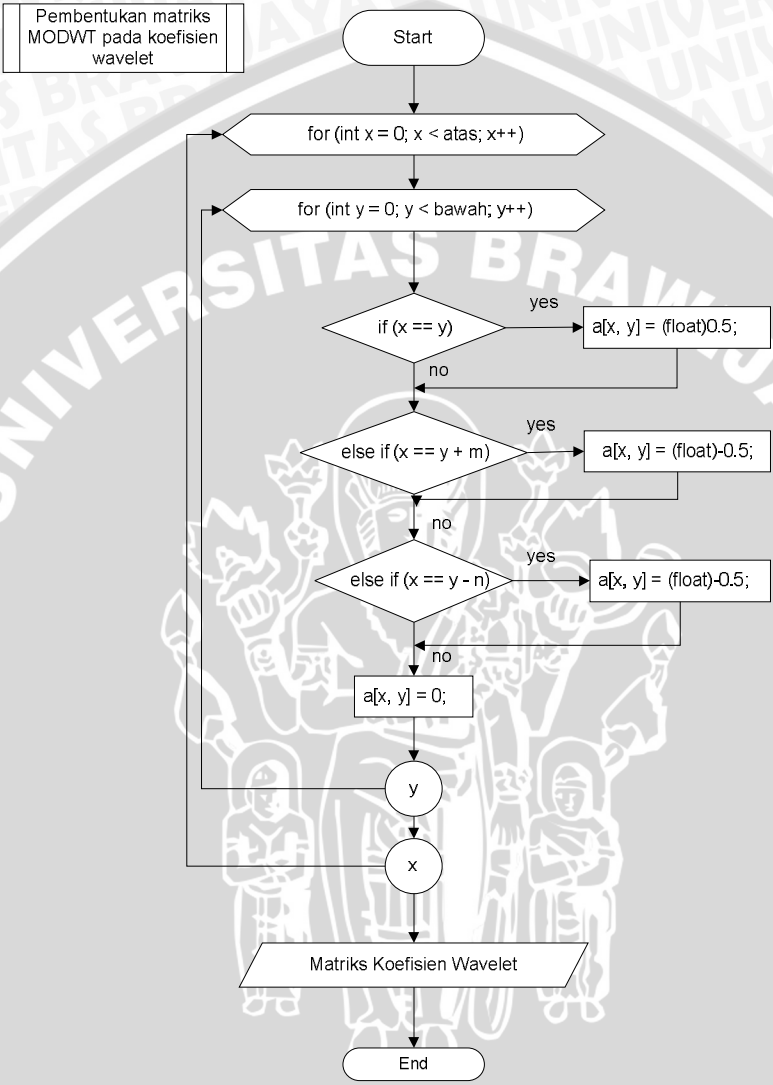
$$J < ln\left(\left(\frac{N}{L-1}\right)\right) + 1 \quad (3.1)$$

Sinyal yang telah di *windowing* dikalikan dengan matriks NxN dengan N adalah panjang data pada tiap *frame* utnuk kemudian diproses sesuai dengan *level* yang disesuaikan dengan perhitungan J. Pada transformasi ini dihasilkan data dalam domain frekuensi untuk kemudian diproses lebih lanjut pada pemetaan PCP, dengan a[r,s] adalah matriks koefisien *wavelet*, itr adalah jumlah iterasi menurut data yang berpengaruh pada jumlah level matriks yang ditentukan dengan m dan n. Gambar alur dekomposisi pada transformasi *wavelet* tersebut dijelaskan pada gambar 3.6.



Gambar 3.6 *Flowchart* Transformasi Haar *Wavelet* MODWT

Sedangkan *flowchart* pembentukan matriks transformasi *wavelet* terletak pada gambar 3.7.



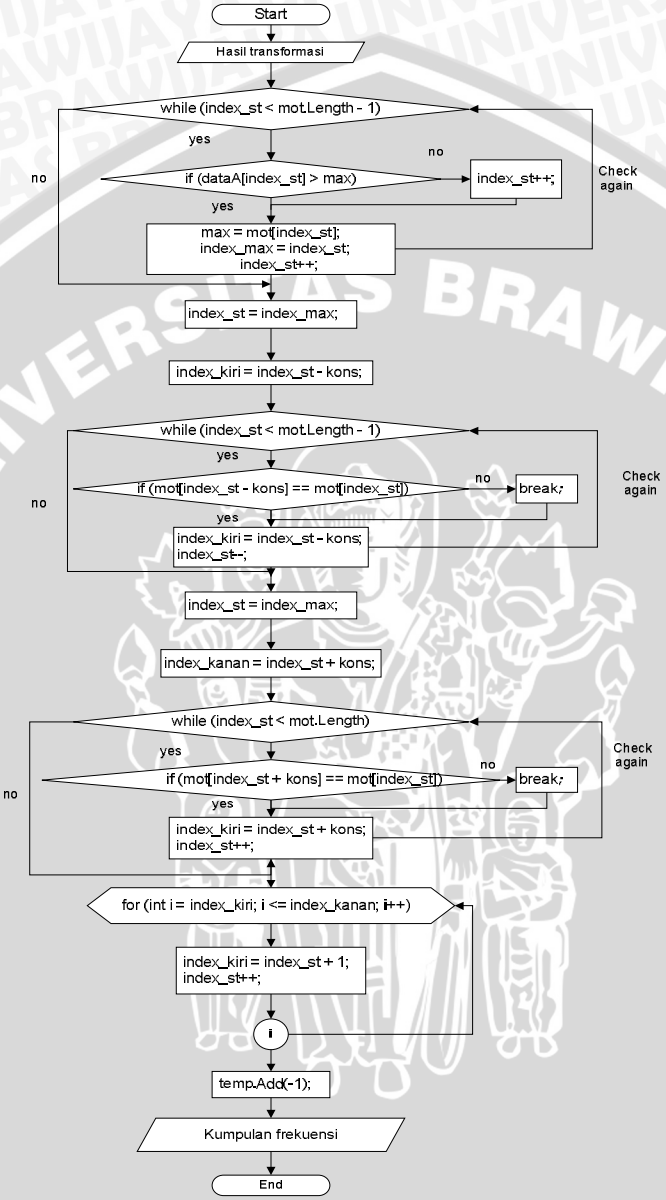
Gambar 3.7 *Flowchart* Pembentukan Matriks Transformasi *Wavelet*

Gambar 3.7 menyatakan proses pembentukan matriks koefisien *wavelet* dengan m dan n adalah perubahan pola pada setiap *level* dekomposisi.

3.3.6 Pengelompokan Frekuensi

Pengelompokan frekuensi merupakan proses lanjutan setelah proses transformasi. Indeks awal dan akhir pada kelompok frekuensi disimpan pada suatu *Arraylist* untuk kemudian dimasukkan kedalam array sesuai dengan bagian awal dan akhir pada indeks tersebut pada tiap bagian.

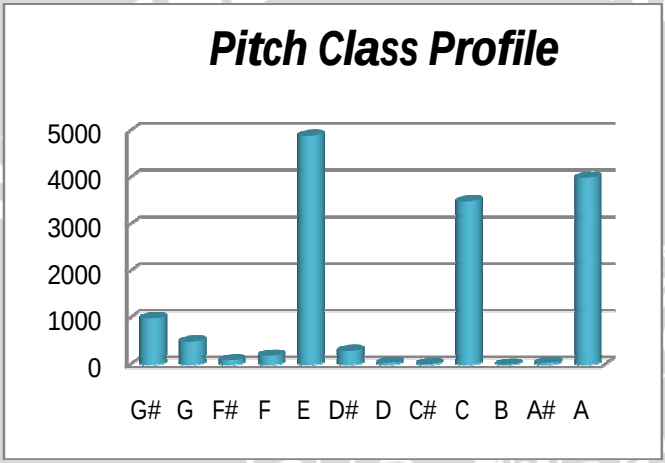
Kumpulan frekuensi pada tiap kelompok minimal terdiri dari dua titik yang berada diantara frekuensi 32.703 Hz sampai 15804 Hz. Pada proses ini dengan thr sebagai nilai pemisah pada sinyal, dan *array* indeks untuk menyimpan indeks awal dan akhir sinyal setiap puncak,. Proses lebih jelasnya dapat dilihat pada gambar 3.8.



Gambar 3.8 *Flowchart* Pengelompokan Frekuensi

3.3.7 Pemetaan *Pitch Class Profile*

Proses transfomasi menghasilkan koefisien pada tiap *frame* yang merupakan frekuensi nada-nada dalam berbagai oktaf yang kemudian dipetakan pada *pitch class profile* (PCP). PCP dipetakan dalam 12 koefisien dan diskala dalam *range* 0 dan 1 untuk selanjutnya diproses pada proses jaringan syaraf tiruan. Visualisasi proses PCP dapat dilihat pada gambar 3.9. Sedangkan gambar 3.10 menunjukkan proses pemetaan PCP.

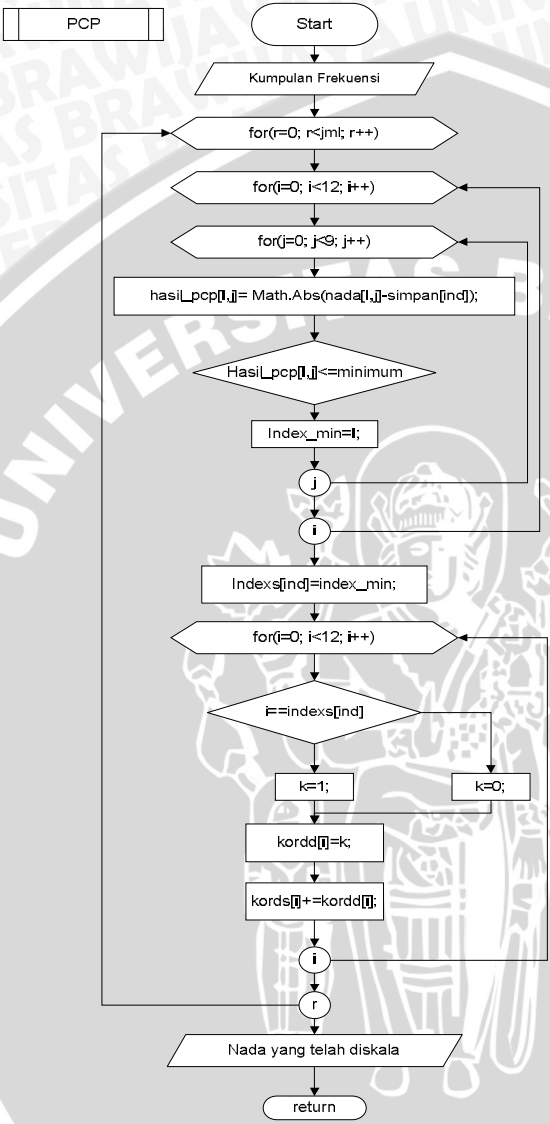


Gambar 3.9 Pitch Class Profile dari sinyal *chord* Am sebelum diskala

Tabel 3.2 Pitch Class Profile dari sinyal *chord* Am setelah diskala

Index	Skala
1	1
2	0
3	0
4	0
5	1
6	0
7	0

Index	Skala
8	0
9	0
10	1
11	0
12	0



Gambar 3.10 *Flowchart* Pemetaan *Pitch Class Profile*

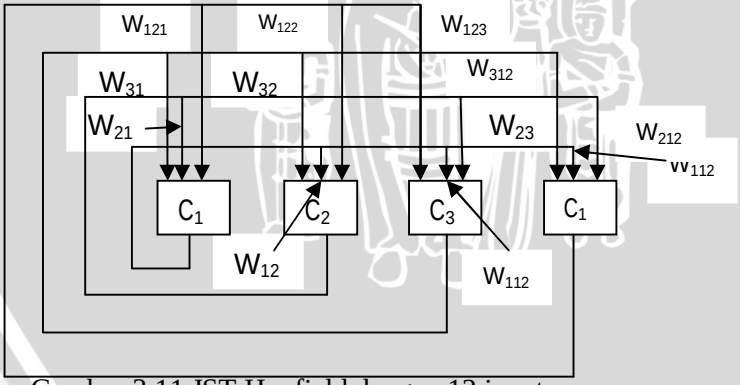
Data dalam domain frekuensi dipetakan satu persatu dengan mencari jarak paling minimal pada frekuensi sesuai dengan urutan 12 nada tanpa memperhatikan urutan oktaf. Nilai di bawah frekuensi minimum digunakan sebagai nilai pembanding pada tiap frekuensi nada untuk membentuk kords[i] sebagai *array* pola *chord*.

3.4 Pelatihan dan Pengenalan

Koefisien-koefisien yang telah diskalakan dan sudah berbentuk pola, akan diproses menjadi *input* pada metode jaringan syaraf tiruan melalui proses pembelajaran untuk tiap-tiap *chord*, dan kemudian proses pengenalan untuk mengenali jenis *chord*.

3.4.1 Pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield

Arsitektur jaringan syaraf tiruan yang digunakan pada penelitian ini adalah Hopfield yang merupakan jenis jaringan syaraf tiruan terawasi (*supervised*) yang terdiri atas 12 masukan sesuai dengan pemetaan *pitch class profile* (PCP). Di sini setiap bobot keluaran akan menjadi inputan pada perhitungan bobot selanjutnya. Proses Hopfield ditunjukkan pada gambar 3.11. *Neuron* utama adalah 12 jenis *chord* yang memberikan keluaran ke semua *neuron* lainnya secara bergantian.



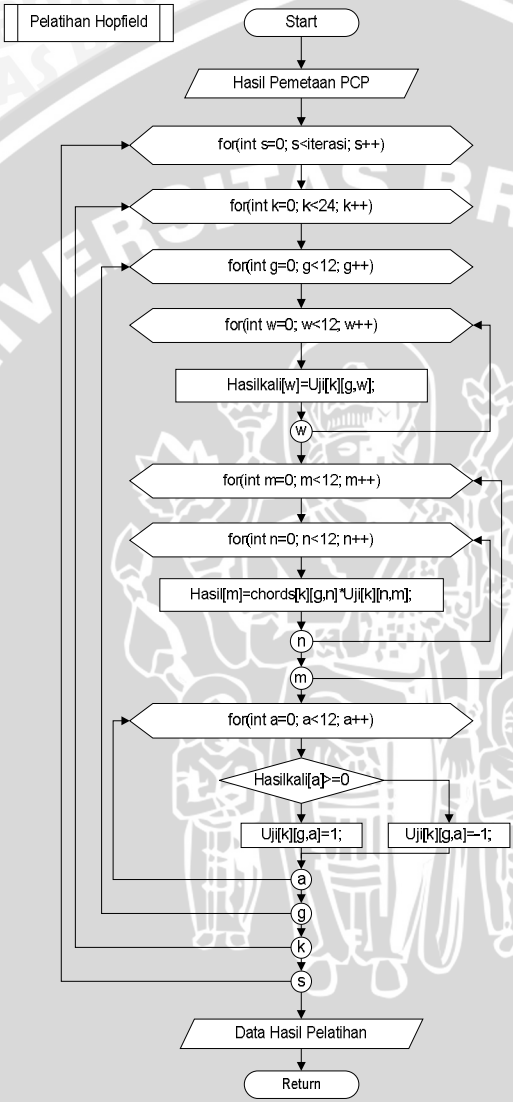
Gambar 3.11 JST Hopfield dengan 12 inputan

Kelas-kelas pengenalan yang akan dipakai dalam pengujian pola dapat dilihat pada tabel 3.3.

Tabel 3.3 Daftar Kelas Pengenalan *Chord*

Index	Kelas	Nada Penyusun
0	Am	A,C,E
1	A	A,C#,E
2	A#m	A#,C#,F
3	A#	A#,D,F
4	Bm	B,D,F#
5	B	B,D#,F#
6	Cm	C,D#,G
7	C	C,E,G
8	C#m	C#,E,G#
9	C#	C#,F,G#
10	Dm	D,F,A
11	D	D,F#,A
12	D#m	D#,F#,A#
13	D#	D#,G,A#
14	Em	E,G,B
15	E	F,G#,B
16	Fm	F,G#,C
17	F	F,A,C
18	F#m	F#,A,C#
19	F#	F#,A#,C#
20	Gm	G,A#,D
21	G	G,B,D
22	G#m	G#,B,D#
23	G#	G#,C,D#

Proses yang terjadi pada JST Hopfield terlihat pada gambar *flowchart* 3.12.

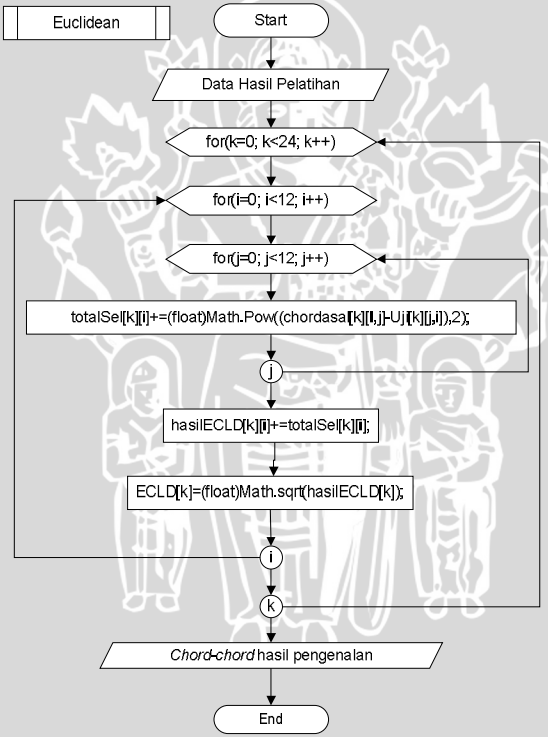


Gambar 3.12 *Flowchart* JST Hopfield

Array Uji[k][g,w] adalah matriks uji dan *array chords*[k][g,n] sebagai pola data asli yang diambil dari media penyimpanan pola berupa file .txt.

3.4.2 Pengenalan menggunakan Jarak Terdekat (*Euclidean Distance*)

Proses terakhir adalah tahap pemetaan dengan *Euclidean Distance.* Data Uji Yang telah dilatih dengan data asal pada tiap-tiap chord diproses lebih lanjut untuk mengetahui jarak yang paling dekat dengan chord asal. Proses pencarian jarak paling minimal ini dijelaskan pada gambar 3.13.



Gambar 3.13 *Flowchart* Jarak Terdekat (*Euclidean Distance*)

Dengan uji[k][j,i] sebagai matriks hasil pelatihan yang kemudian diukur jaraknya dengan *chord*asal[k][i,j] yang menghasilkan jarak yang disimpan pada *array* ECLD[k].

3.5 Perhitungan Manual

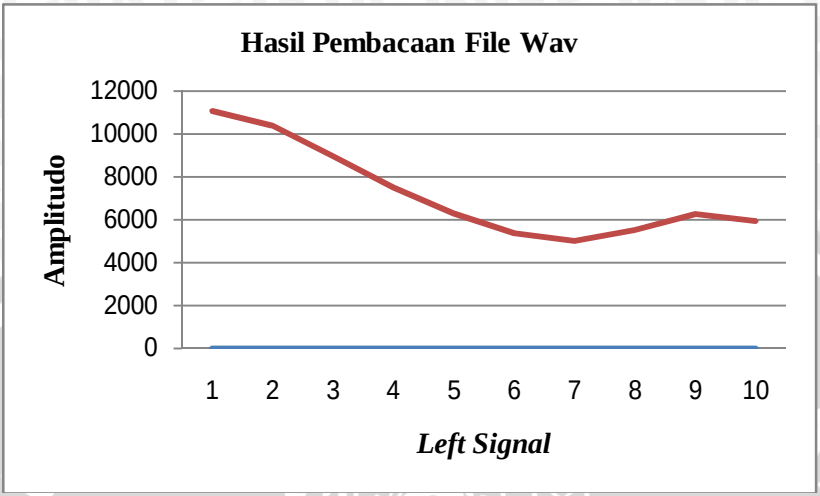
3.5.1 Pembacaan *File* Wav

Hasil pembacaan *file* wav diketahui berupa *chord* A# Mayor dan mempunyai data seperti pada tabel pada tabel 3.4.

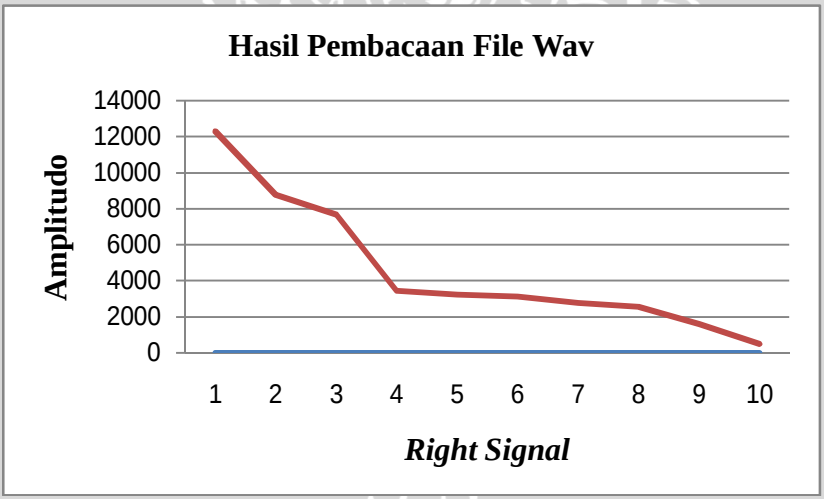
Tabel 3.4 Data Hasil Pembacaan wav

Index	***Left Sample*** **0**	***Right Sample*** **1**
0	11074.00	-244.04
1	10387.00	-1023.52
2	8948.00	-1083.08
3	7507.00	-897.72
4	6309.00	-679.64
5	5377.00	-138.92
6	5023.00	702.92
7	5525.00	957
8	6261.00	-65.56
9	5945.00	2.56

Chart hasil pembacaan potongan sinyal wav ditunjukkan pada gambar 3.14 dan 3.15.



Gambar 3.14 *Char*t Hasil Pembacaan Wav (*left*)



Gambar 3.15 *Chart* Hasil Pembacaan Wav (*right*)

3.5.2 **Perhitungan *Pre-Emphasis***

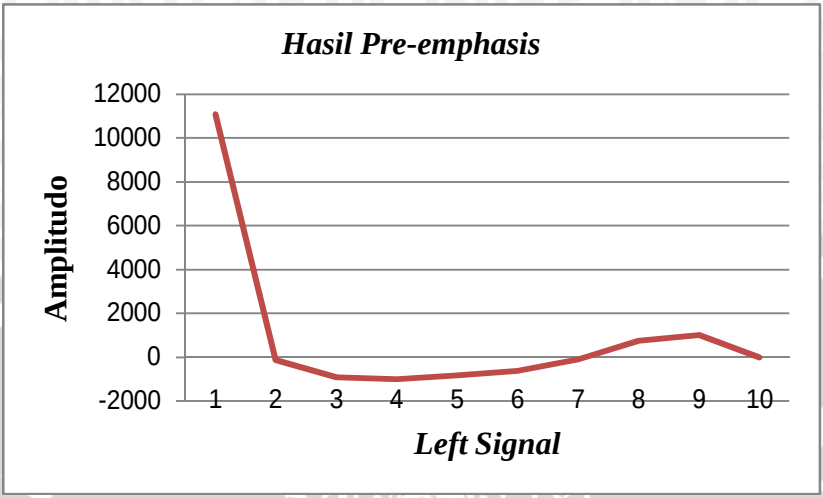
Data hasil pembacaan wav dilewatkan pada filter *pre-emphasis* dengan ã = 0,95 untuk memperhalus sinyal. Hasil perhitungan filter *pre-emphasis* ditunjukkan pada tabel 3.5. Perhitungan pre-emphasis untuk index ke-1 pada left *sample* menghasilkan nilai = -133.3.

$$\tilde{s}(n) = 10387 - (0.95\, x 11074) = -133.3$$

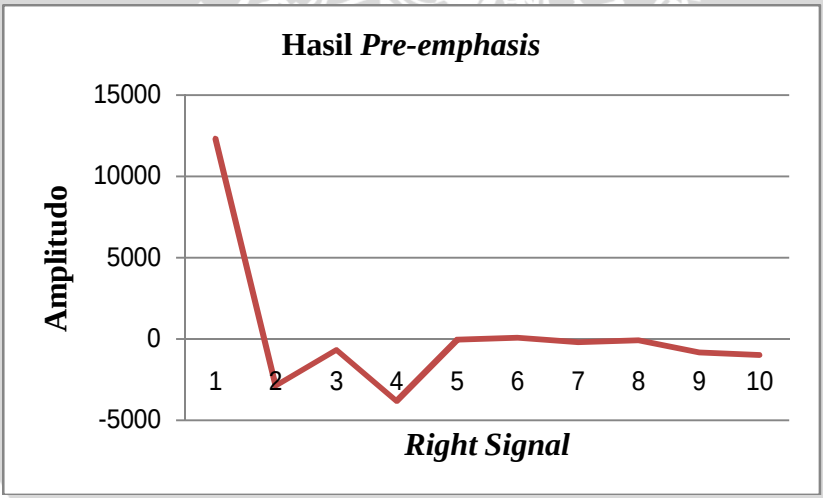
Tabel 3.5 Hasil Perhitungan *Pre-emphasis*

Index	**Left Sample**	**Right Sample**
	0	**1**
0	11074	-244.04
1	-133.3	-791.682
2	-919.65	-110.736
3	-993.6	131.206
4	-822.65	173.194
5	-616.55	506.738
6	-85.15	834.894
7	753.15	289.226
8	1012.25	-974.71
9	7.05	64.842

Chart hasil perhitungan pada filter *pre-emphasis* ditunjukkan pada gambar 3.16 untuk left sample dan 3.17 untuk right sample.



Gambar 3.16 *Chart* Hasil Perhitungan *Pre-emphasis* (*left*)



Gambar 3.17 *Chart* Hasil Perhitungan *Pre-emphasis* (*right*)

3.5.3 *Windowing* dengan *Hamming Window*

Berikut ini adalah hasil perhitungan metode *Hamming window* pada tabel 3.6. Perhitungan window dengan metode *hamming* untuk index ke-0 menghasilkan nilai 0.08 dengan data permisalan data total sejumlah 22 buah.

$$w(n) = 0.54 - 0.46\cos\left(\frac{2x180x0}{22-1}\right)$$

Tabel 3.6 Perhitungan *Window* dengan metode Hamming

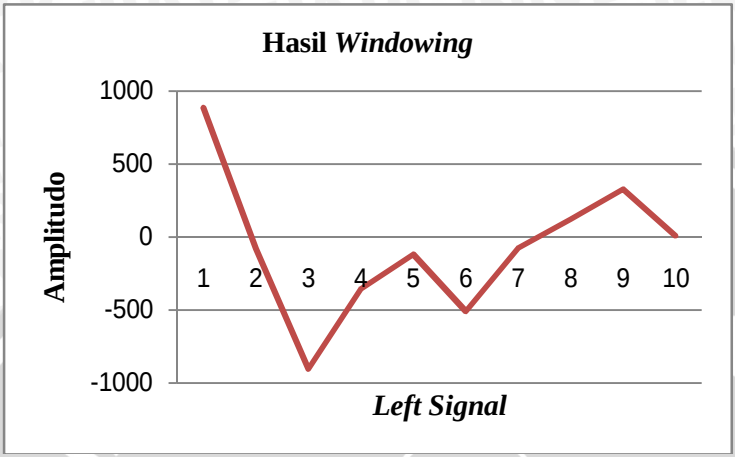
Index	x(n)
0	0.08
1	0.602323
2	0.983112
3	0.357607
4	0.14631
5	0.82907
6	0.855361
7	0.165477
8	0.326124
9	0.972477

Setelah mengalikan antara perhitungan *window* dengan data awal, akan diperoleh data pada tabel 3.7 sebagai berikut:

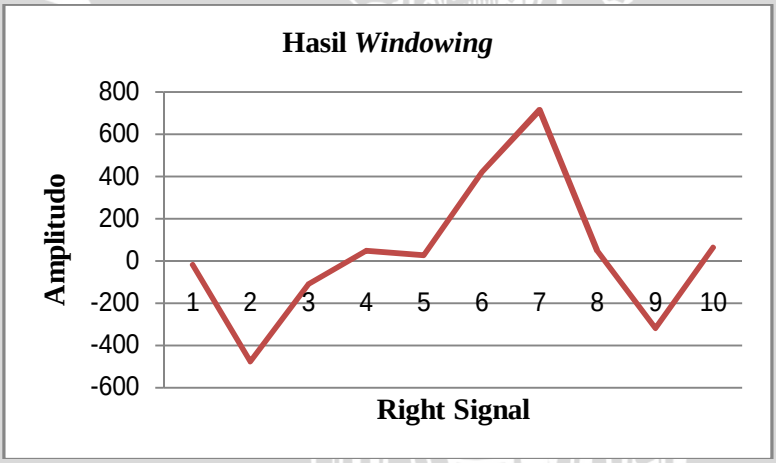
Tabel 3.7 Data Setelah *Windowing*

Index	Left Sample 0	Right Sample 1
0	885.92	-19.5232
1	-80.2896	-476.848
2	-904.119	-108.866
3	-355.319	46.92024
4	-120.362	25.34007
5	-511.163	420.1214
6	-72.834	714.1354
7	124.6288	47.86018
8	330.1186	-317.876
9	6.855964	63.05736

Chart hasil perhitungan *windowing* dengan metode Hamming, ditunjukkan pada gambar 3.18 dan 3.19.



Gambar 3.18 *Chart* sesudah *Windowing* (*left*)



Gambar 3.19 *Chart* sesudah *Windowing* (*right*)

3.5.4 Transformasi *Wavelet* dengan *Maximal Overlap Discrete Wavelet Transform* (MODWT)

a. Jumlah Iterasi

Penentuan Jumlah Iterasi dilakukan dengan rumus sebagai berikut:

$$j < \ln((N/L-1)+1)$$
$$j < \ln((10/2-1)+1) = 2$$

b. Koefisien Skala

Setelah Perhitungan 2 kali iterasi ditemukan koefisien skala pada tabel 3.10. Level pertama untuk *left channel* dihitung sesuai tabel 3.8 dimana matriks (warna biru) dikalikan dengan hasil *windowing* (warna hitam) sehingga menghasilkan nilai pada kolom terakhir (warna hitam tebal). Level kedua dari data yang sama dihitung sesuai dengan perhitungan pada tabel 3.9. Matriks level 2 (warna biru) berubah dari level pertama dengan selang nol yang bergeser dikalikan dengan hasil dekomposisi level pertama dan kemudian menghasilkan nilai pada kolom terakhir (warna hitam tebal).

Tabel 3.8 Perhitungan koefisien skala level 1

0.5	0	0	0	0	0	0	0	0	0.5	885.92	**446.388**
0.5	0.5	0	0	0	0	0	0	0	0	-80.2896	**402.8152**
0	0.5	0.5	0	0	0	0	0	0	0	-904.119	**-492.204**
0	0	0.5	0.5	0	0	0	0	0	0	-355.319	**-629.719**
0	0	0	0.5	0.5	0	0	0	0	0	-120.362	**-237.841**
0	0	0	0	0.5	0.5	0	0	0	0	-511.163	**-315.763**
0	0	0	0	0	0.5	0.5	0	0	0	-72.834	**-291.999**
0	0	0	0	0	0	0.5	0.5	0	0	124.6288	**25.8974**
0	0	0	0	0	0	0	0.5	0.5	0	330.1186	**227.3737**
0	0	0	0	0	0	0	0	0.5	0.5	6.855964	**168.4873**

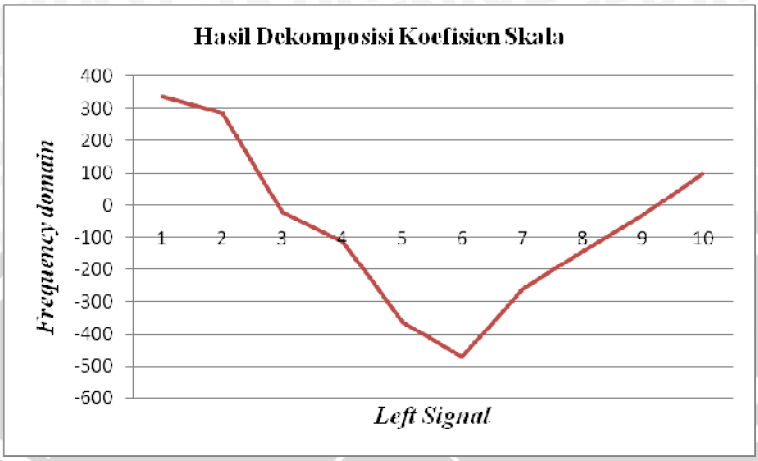
Tabel 3.9 Perhitungan koefisien skala level 2

0.5	0	0	0	0	0	0	0	0.5	0	446.388	**336.8808**
0	0.5	0	0	0	0	0	0	0	0.5	402.8152	**285.6512**
0.5	0	0.5	0	0	0	0	0	0	0	-492.204	**-22.9082**
0	0.5	0	0.5	0	0	0	0	0	0	-629.719	**-113.452**
0	0	0.5	0	0.5	0	0	0	0	0	-237.841	**-365.022**
0	0	0	0.5	0	0.5	0	0	0	0	-315.763	**-472.741**
0	0	0	0	0.5	0	0.5	0	0	0	-291.999	**-264.92**
0	0	0	0	0	0.5	0	0.5	0	0	25.8974	**-144.933**
0	0	0	0	0	0	0.5	0	0.5	0	227.3737	**-32.3124**
0	0	0	0	0	0	0	0.5	0	0.5	168.4873	**97.19234**

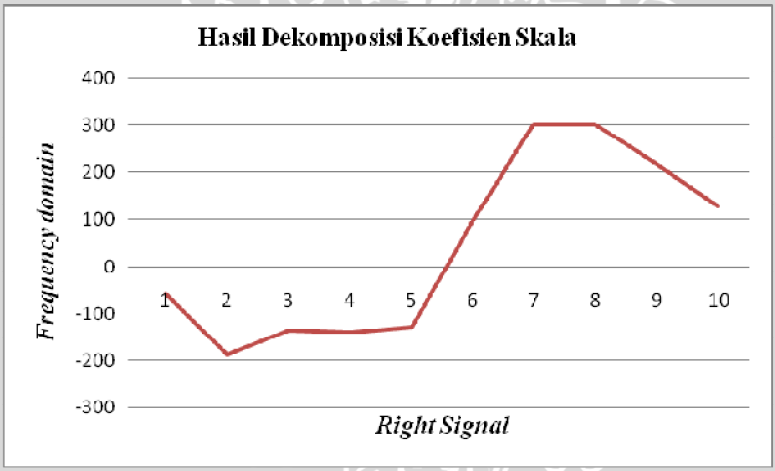
Tabel 3.10 Hasil Dekomposisi Koefisien Skala

Index	**Left Sample 0**	**Right Sample 1**
0	336.8808	-56.6204
1	285.6512	-187.797
2	-22.9082	-135.545
3	-113.452	-139.579
4	-365.022	-128.363
5	-472.741	95.87893
6	-264.92	301.6293
7	-144.933	301.8643
8	-32.3124	216.0602
9	97.19234	126.7942

Chart hasil dekomposisi pada koefisien skala ditunjukkan pada gambar 3.20 dan 3.21.



Gambar 3.20 *Chart* Hasil Dekomposisi Pada Koef. Skala (*left*)



Gambar 3.21 *Chart* Hasil Dekomposisi Pada Koef. Skala (*right*)

c. Koefisien Wavelet MODWT

Setelah Perhitungan dua kali iterasi ditemukan koefisien *wavelet* pada tabel 3.13. Level pertama untuk *left channel* dihitung sesuai tabel 3.11 dimana matriks (warna biru) dikalikan dengan hasil *windowing* (warna hitam) sehingga menghasilkan nilai pada kolom terakhir (warna hitam tebal). Level ke-2 dari data yang sama dihitung sesuai dengan perhitungan pada tabel 3.12. Matriks level 2 (warna biru) berubah dari level ke-1 dengan selang nol yang bergeser dikalikan dengan hasil dekomposisi level pertama, kemudian menghasilkan nilai pada kolom terakhir (warna hitam tebal).

Tabel 3.11 Perhitungan koefisien *wavelet* level 1

0.5	0	0	0	0	0	0	0	0	-0.5	885	**439**
-0.5	0.5	0	0	0	0	0	0	0	0	-80	**-483**
0	-0.5	0.5	0	0	0	0	0	0	0	-904	**-411**
0	0	-0.5	0.5	0	0	0	0	0	0	-355	**274**
0	0	0	-0.5	0.5	0	0	0	0	0	-120	**117**
0	0	0	0	-0.5	0.5	0	0	0	0	-511	**-195**
0	0	0	0	0	-0.5	0.5	0	0	0	-72	**219**
0	0	0	0	0	0	-0.5	0.5	0	0	124	**98**
0	0	0	0	0	0	0	-0.5	0.5	0	330	**102**
	0	0	0	0	0	0	0	-0.5	0.5	6	**-161**

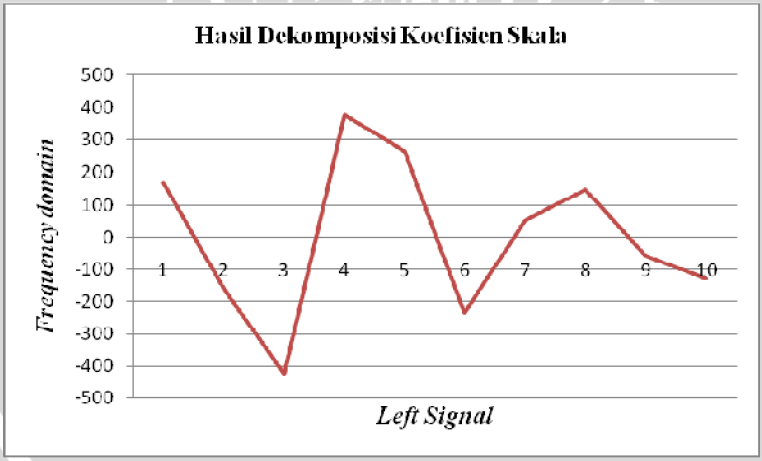
Tabel 3.12 Perhitungan koefisien *wavelet* level 2

0.5	0	0	0	0	0	0	0	-0.5	0	439	**168**
0	0.5	0	0	0	0	0	0	0	-0.5	-483	**-160**
-0.5	0	0.5	0	0	0	0	0	0	0	-411	**-425**
0	-0.5	0	0.5	0	0	0	0	0	0	274	**378**
0	0	-0.5	0	0.5	0	0	0	0	0	117	**264**
0	0	0	-0.5	0	0.5	0	0	0	0	-195	**-234**
0	0	0	0	-0.5	0	0.5	0	0	0	219	**50**
0	0	0	0	0	-0.5	0	0.5	0	0	98	**147**
0	0	0	0	0	0	-0.5	0	0.5	0	102	**-58**
0	0	0	0	0	0	0	-0.5	0	0.5	-161	**-130**

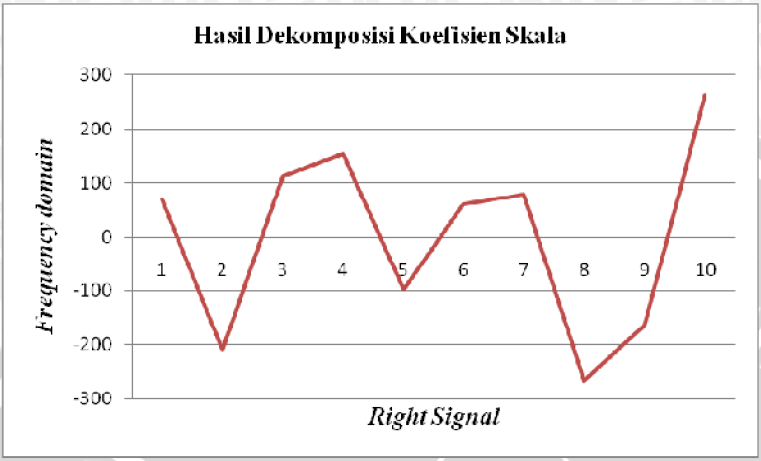
Tabel 3.13 Hasil Dekomposisi Koefisien *Wavelet*

Index	Left Sample 0	Right Sample 1
0	168.3936	70.78891
1	-160.737	-209.565
2	-425.723	112.6406
3	378.7524	153.2778
4	264.6966	-97.3905
5	-234.9	59.74877
6	50.843	78.89854
7	147.066	-265.264
8	-58.2098	-164.938
9	-130.181	261.8021

Chart hasil dekomposisi pada koefisien *wavelet* ditunjukkan pada gambar 3.22 dan 3.23.



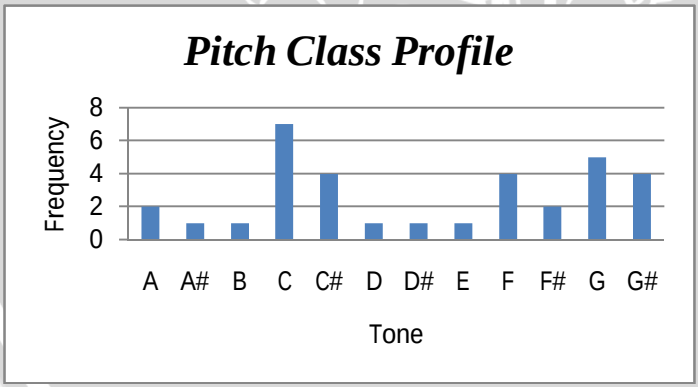
Gambar 3.22 *Chart* Hasil Dekomposisi Pada Koef. *Wavelet* (*left*)



Gambar 3.23 *Chart* Hasil Dekomposisi Pada Koef. *Wavelet* (*right*)

3.5.5 Pemetaan *Pitch Class Profile*

Dari hasil transformasi di atas, yang dipakai pada pemetaan adalah Koefisien *wavelet* setelah iterasi ke-2 dalam domain frekuensi, menghasilkan grafik pada gambar 3.24.



Gambar 3.24 Pemetaan pada *Pitch Class Profile*

Dari hasil grafik di atas, dilakukan penskalaan sebagai *input* jaringan syaraf tiruan, dijelaskan pada tabel 3.14.

Tabel 3.14 Hasil Pemetaan PCP setelah diskala

Index	Skala
1	0
2	0
3	0
4	1
5	0
6	0
7	1
8	0
9	0
10	0
11	1
12	0

3.5.6 Pelatihan dan Pengenalan

1. Pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield

a. Vektor A# Mayor
Vektor tersimpan untuk *chord* A# Mayor ditunjukkan pada tabel 3.15 dalam bentuk matriks dengan diagonal utama berupa angka nol. Pembentukan matriks ini diciptakan oleh pola *chord* dari hasil perkalian vector pola *chord* A# Mayor tersebut.

Tabel 3.15 Matriks asal dari chord A# Mayor

0	1	-1	1	1	-1	1	1	1	1	-1	1
1	0	-1	1	1	-1	1	1	1	1	-1	1
-1	-1	0	-1	-1	1	-1	-1	-1	-1	1	-1
1	1	-1	0	1	-1	1	1	1	1	-1	1
1	1	-1	1	0	-1	1	1	1	1	-1	1
-1	-1	1	-1	-1	0	-1	-1	-1	-1	1	-1
1	1	-1	1	1	-1	0	1	1	1	-1	1
1	1	-1	1	1	-1	1	0	1	1	-1	1
1	1	-1	1	1	-1	1	1	0	1	-1	1
1	1	-1	1	1	-1	1	1	1	0	-1	1
-1	-1	1	-1	-1	1	-1	-1	-1	-1	0	-1
1	1	-1	1	1	-1	1	1	1	1	-1	0

b. Hasil pemetaan PCP data uji yang sudah dikalikan dengan hasil transposenya, dijelaskan pada tabel 3.16.

Tabel 3.16 Hasil Perkalian Skala dengan Transposenya

1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	-1
1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	-1
1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	-1
1	1	1	-1	1	1	-1	1	1	1	-1	1

2. Pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield

a. Pada tahap sebelumya didapatkan hasil pemetaan data sesuai masukan, yang kemudian dikalikan dengan transposenya (hasil uji) dan data asli sebagai sumber data. Selanjutnya kolom data asli (seperti warna kuning pada Tabel 3.15) dikalikan dengan tiap-tiap kolom matriks uji(seperti tanda hijau pada Tabel 3.16), dan ditambahkan dengan nilai pada kolom yang diuji (seperti angka yang berwarna merah pada tabel 3.16) pada iterasi tertentu. Hasil didapat dari pelatihan sampai kondisi konvergen. Sehingga akhirnya didapatkan hasil pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield seperti pada tabel 3.17.

Matriks hasil (0,0) = 1+((0.1)+(1.1)+(-1.1)+(1.-1)+(1.1)+(-1.1)+(1.-1)+(1.1)+(1.1)+(1.1)+(-1.-1)+(1.1)) = 4

Nilai 4 merupakan angka positif, jadi matriks hasil (0,0) = 1 (warna ungu). Perhitungan ini terus berlanjut pada semua kolom sampai selesai pada tabel 3.16 kemudian dilanjutkan dengan baris kedua dan seterusnya pada tabel 3.15.

Tabel.3.17. Hasil Pelatihan Jaringan Syaraf Tiruan Hopfield

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	-1	1	1	-1	1	1	1	-1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	-1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

b. Pengenalan menggunakan jarak terdekat (*Euclidean Distance*)

Hasil pelatihan jaringan syaraf tiruan Hopfield akan dijadikan sebagai bahan uji selanjutnya pada *Euclidean Distance,* sehingga dihasilkan matriks sesuai data masukan yaitu A# Mayor. Data di atas, didapat dari rumus untuk mengetahui jarak terdekat (*Euclidean Distance*). Pengurangan dilakukan sesuai dengan kolom matriks uji, dengan baris matriks asli. Sehingga didapatkan 24 matriks hasil pengurangan. Salah satu matriks hasil proses ini terlihat pada tabel 3.18.

Tabel.3.18. Hasil Pengenalan Menggunakan *Euclidean Distance*

0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	4	0	0	4	0	0	0	0	4	0	
0	0	48	0	0	48	0	0	0	0	48	0	144

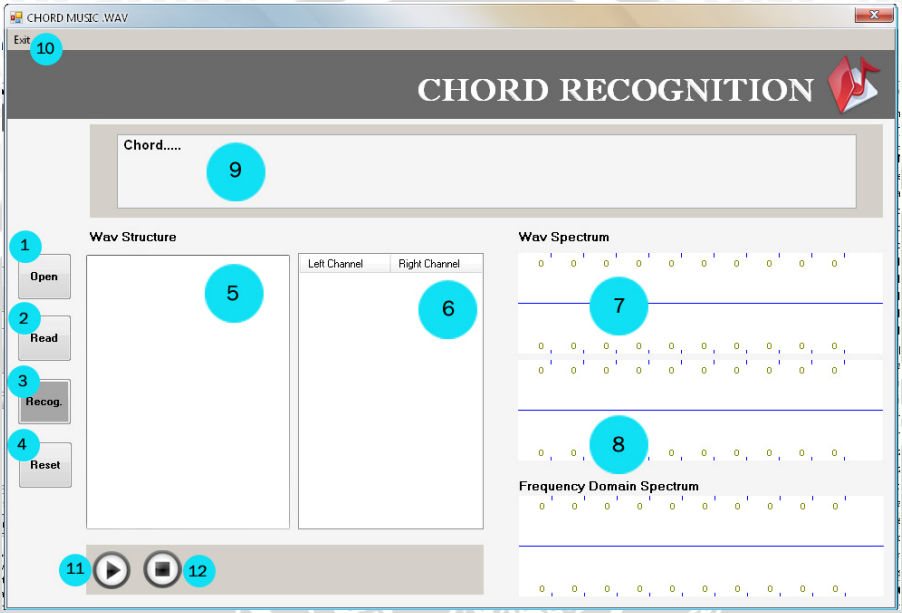
Matriks di atas dijumlah pada tiap-tiap kolom (warna hijau). Jumlah total tiap kolom tersebut (warna merah) diakar sehingga terbentuk nilai 12 sebagai penanda jarak kedekatan untuk pengenalan data uji dengan data asal A# Mayor.

3.6 Rancangan *User Interface* Sistem

Rancangan user interface sistem pada pengenalan *chord* ini terdiri dari 1 form *chord recognition.*

3.6.1 Form Pembacaan *File* Wav

Rancangan *user interface* untuk pembacaan *file* wav digambarkan pada gambar 3.25.



Gambar 3.25 Gambar User Interface Pembacaan File Wav

Keterangan :

1. *Button* untuk *open file* wav
2. *Button* untuk pambacaan struktur *file* wav
3. *Button* untuk mengetahui hasil pengenalan *chord*
4. *Button* untuk mengembalikan proses pada bentuk awal (*reset*)
5. *TextBox* untuk mengeluarkan hasil bacaan struktur *file* wav
6. *ListView* untuk mengeluarkan beberapa sinyal diskrit hasil pembacaan
7. *WavDrawingPanel* untuk mengeluarkan grafik sinyal wav dalam domain waktu
8. *WavDrawingPanel* untuk mengeluarkan grafik pada domain frekuensi
9. *TextBox* untuk kotak keluaran hasil pengenalan *chord*
10. *ToolScripMenuItem Exit* untuk keluar dari aplikasi

11. *Image Play* untuk memutar masukan *file* wav
12. *Image Stop* untuk menghentikan *file* wav yang sedang diputar

Hasil dari penelitian akan disimpan dalam tabel hasil uji coba, yang ditampilkan pada tabel 3.19.

Tabel 3.19 Hasil Uji Coba

No	**Nama *Chord***	**Jumlah *Chord***	**Jumlah *Chord* Benar**	**Akurasi %**

BAB IV
IMPLEMANTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi merupakan lingkungan yang terdiri dari lingkungan perangkat keras dan perangkat lunak. Sistem ini dibuat dengan menggunakan Visual Studio C# 2008 pada *operating system* Windows XP SP 2 dan Windows 7.

4.1.1 Lingkungan Perangkat Keras

Dalam penelitian *chord recognition* ini menggunakan beberapa perangkat keras sebagai berikut:

1. Prosesor Intel Dual Core CPU T2390 1,86GHz
2. Memori DDR2 1 Gb
3. *Harddisk* 120 Gb
4. *Monitor* 14”
5. *Keyboard*

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian *chord recognition* ini adalah :

1. Sistem Operasi Windows XP SP2
2. Sistem Operasi Wndows 7
3. Microsoft Visual Studio C# 2008 *Express Edition*
4. Microsoft Office Word 2007
5. Microsoft Office Excel 2007
6. Notepad

4.2 Struktur Data

Aplikasi pada *chord recognition* ini menggunakan array untuk penyimpanan data sinyal yang terkandung dalam suatu *chord*. Pembacaan *file* wav menggunakan acuan awal Wave *Header* yang terdiri dari bagian RIFF Chunk dan fmt sub-chunk, *Channel* pada bagian fmt sub-chunk, dan data pada bagian data sub-chunk yang ada

pada sebuah lagu untuk kemudian diolah dan selanjutnya diproses pada tahap pembelajaran dan pengenalan. *Sourcecode* 4.1 menunjukkan struktur data pembacaan *file* wav.

```
public class wavInfo
{
    public string fileName;
    public uint fileLength;
}
public class riffChunk
{
    public string riffID;
    public uint riffSize;
    public string riffFormat;
}
public class fmtChunk
{
    public string fmtID;
    public uint fmtSize;
    public ushort fmtAudioFormat;
    public ushort fmtChannels;
    public uint fmtSampleRate;
    public uint fmtByteRate;
    public ushort fmtBlockAlign;
    public uint fmtBitsPerSample;
}
public class dataChunk
{
    public string dataID;
    public int dataSize;
    public int dataSampleCount;
    public float[] wavFloat;
    public float[] wavLeft;
    public float[] wavRight; }
```

***Sourcecode* 4.1** Struktur Data Pembacaan File Wav

Wave *Header* merupakan serangkaian informasi pada suatu file wav. Sedangkan *channel* merupakan informasi jumlah *channel* pada file wav, yaitu 1 *channel* untuk *mono* dan 2 *channel* untuk *stereo*. Selanjutnya data pada tiap *channel* tersebut disimpan pada *array* satu dimensi. Rincian variabel yang digunakan dapat dilihat pada tabel 4.1.

Tabel 4.1 Rincian atribut Wave*Header*, *Channel*, dan Data

Nama variabel	Keterangan
RiffID	Atribut bertipe string berukuran 4 untuk menyimpan karakter 'RIFF'
RiffSize	Atribut bertipe uint untuk menyimpan ukuran chunk file wav
RiffFormat	Atribut bertipe string berukuran 4 untuk menyimpan karakter 'WAVE'
fmtID	Atribut bertipe string berukuran 4 untuk menyimpan karakter 'fmt '
fmtSize	Atribut bertipe uint untuk menyimpan ukuran *subchunk*
fmtAudioFormat	Atribut bertipe ushort untuk menyimpan format suara file wav, misalnya nilai 1 untuk format suara PCM
fmtChannels	Atribut bertipe ushort untuk menyimpan jumlah channel *file* suara, 1 untuk *mono* dan 2 untuk *stereo*
fmtSampleRate	Atribut bertipe uint untuk menyimpan nilai *Sample Rate*
fmtByteRate	Atribut bertipe uint untuk menyimpan jumlah byte tiap detik
fmtBlockAlign	Atribut bertipe ushort untuk menyimpan jumlah byte tiap sample suara
fmtBitsPerSample	Atribut bertipe uint untuk menyimpan jumlah bit tiap sample suara
dataID	Atribut bertipe string berukuran 4 untuk menyimpan karakter 'data'
dataSize	Atribut bertipe int untuk menyimpan Data size chunk
dataSampleCount	Atribut bertipe int untuk menyimpan Data sample count
WavFloat	Atribut array bertipe float untuk menyimpan data aktual suara untuk setiap *channel* suara
WavLeft dan WavRigth	Atribut array bertipe float untuk menyimpan data wav yang berupa *stereo* atau wav yang mempunyai 2 *channel*

4.3 Implementasi Program

Program pada penelitian ini, secara garis besar terdiri dari pembacaan file wav, penggambaran *spectrum* sinyal, pengolahan sinyal digital berupa *preemphasis*, *frame blocking*, *windowing*, pembagian sinyal berdasarkan kelompok frekuensi, transformasi *wavelet* untuk memperoleh data dalam domain frekuensi, dan pemetaan *pitch class profile* (PCP) yang menghasilkan 12 koefisien PCP, pembelajaran menggunakan algoritma Hopfield, dan pengenalan menggunakan *Euclidean Distance* sebagai pembanding jarak terkecil antar *chord* uji dan *chord* asli.

4.3.1 Pembacaan File Wav

Pembacaan file wav terdiri dari 2 tahap yaitu tahap pembacaan *Header* Wav sesuai dengan posisi pada struktur wav yang secara otomatis akan memberikan informasi mengenai jenis *channel* wav *mono* atau *stereo*. Selanjutnya tahap kedua yaitu pembacaan data wav pada posisi 44 yang akan mengeluarkan informasi amplitudo pada tiap *channel*. Informasi amplitudo ini akan dikeluarkan beberapa baris pada *listview* yang bisa langsung dilihat pada program. *Source code* 4.2 yang merupakan *source code* untuk pembacaan file wav dengan memanggil urutan posisi struktur wav yang sudah dijelaskan sebelumnya.

```
public class WAV
    {
   private wavInfo wavInfoStruct = new wavInfo();
   private riffChunk wavRiffStruct = new riffChunk();
   private fmtChunk wavFmtStruct = new fmtChunk();
   private dataChunk wavDataStruct = new dataChunk();
   private String FileName;

        public void setFileName(String fileName)
        {
            FileName = fileName;
            wavInfoStruct.fileName = FileName;
        }

        public wavInfo getWAVInfo()
        {
            return wavInfoStruct;
        }

        public riffChunk getRiffStruct()
        {
            return wavRiffStruct;
        }

        public fmtChunk getFmtStruct()
        {
            return wavFmtStruct;
        }

        public dataChunk getDataStruct()
        {
            return wavDataStruct;
        }

        public void readHeader()
        {
            String tmpString = null;
            FileStream fs = new FileStream(FileName,
FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            fs.Position = 0;
            tmpString = new string(br.ReadChars(4));
            wavRiffStruct.riffID = tmpString;
```

***Source code* 4.2** Pembacaan File Wav

```
fs.Position = 4;
wavRiffStruct.riffSize = (uint)br.ReadInt32();

tmpString = null;
fs.Position = 8;
tmpString = new string(br.ReadChars(4));
wavRiffStruct.riffFormat = tmpString;

wavInfoStruct.fileLength = (uint)fs.Length;

tmpString = null;
fs.Position = 12;
tmpString = new string(br.ReadChars(4));
wavFmtStruct.fmtID = tmpString;

fs.Position = 16;
wavFmtStruct.fmtSize = (uint)br.ReadInt32();

fs.Position = 20;
wavFmtStruct.fmtAudioFormat = (ushort)br.ReadInt16();

fs.Position = 22;
wavFmtStruct.fmtChannels = (ushort)br.ReadInt16();

fs.Position = 24;
wavFmtStruct.fmtSampleRate = (uint)br.ReadInt32();

fs.Position = 28;
wavFmtStruct.fmtByteRate = (uint)br.ReadInt32();

fs.Position = 32;
wavFmtStruct.fmtBlockAlign = (ushort)br.ReadInt16();

fs.Position = 34;
wavFmtStruct.fmtBitsPerSample = (uint)br.ReadInt16();

tmpString = null;
fs.Position = 36;
tmpString = new string(br.ReadChars(4));
wavDataStruct.dataID = tmpString;

fs.Position = 40;
wavDataStruct.dataSize = br.ReadInt32();
```

***Source code* 4.2** (lanjutan) Pembacaan File Wav

```
fs.Position = 44;
tempByte = new byte[wavDataStruct.dataSize];
tempByte = br.ReadBytes((int)wavDataStruct.dataSize);
wavDataStruct.wavBytes = tempByte;

fs.Position = 44;
wavDataStruct.dataSampleCount = wavDataStruct.dataSize /
2;

float[] tempData = new
float[wavDataStruct.dataSampleCount];
float[] tempDataLeft = new
float[wavDataStruct.dataSampleCount /2];
float[] tempDataRight = new
float[wavDataStruct.dataSampleCount /2];
int jumGanjil = 0;
int jumGenap = 0;
for (uint i = 1; i < wavDataStruct.dataSampleCount; i++)
{
float tempValue = ((short)(br.ReadInt16()));
tempData[i] = tempValue;

if(wavFmtStruct.fmtChannels==2)
{
if (i % 2 == 0)
{
tempDataLeft[jumGenap] = tempValue;
jumGenap++;
}
else
{
tempDataRight[jumGanjil] = tempValue;
jumGanjil++;
}}}
```

***Source code* 4.2** (lanjutan) Pembacaan File Wav

4.3.2 Penggambaran Spektrum Sinyal

Setelah proses pembacaan file wav, informasi amplitudo yang didapat dari proses pembacaan data wav yang kemudian dilanjutkan pada proses penggambaran spektrum sinyal. Penggambaran ini diawali dengan penambahan *toolbox* WAV*DrawingPanel*. *Source code* 4.3 merupakan *source code* penggambaran spektrum sinyal dalam domain waktu dan frekuensi untuk jenis *stereo* ataupun *mono*.

```
RectangleF range = this.Bounds;
float width, height;
width = this.Width;
height = this.Height;
Bitmap bmp = new Bitmap((int)width, (int)height,
PixelFormat.Format32bppArgb);
Graphics graph = Graphics.FromImage(bmp);
graph.Clear(Color.White);
Pen redPen = new Pen(Color.Red);
Pen blackPen = new Pen(Color.Black);
float xvalue = 0;
float yvalue = height / 2;
step = width / (dataStruct.dataSampleCount);
float maxValue = 0;

if (drawingType == WAVDrawingType.wavFloatType)
   {
 for (int i = 0; i < dataStruct.dataSampleCount; i++)
     {
       float value = (int)dataStruct.wavFloat[i];
       value = Math.Abs(value);
       if (value > maxValue)
          {
           maxValue = value;
          }
     }
       maxValue = maxValue + 50;
       float scale = (height / 2) / maxValue;
       for (int i = 0; i <
dataStruct.dataSampleCount; i++)
        {
       float value;
       value = dataStruct.wavFloat[i] * scale;
       graph.DrawLine(blackPen, xvalue, yvalue,
xvalue + step, (height / 2) + value);
       xvalue = xvalue + step;
       yvalue = (height / 2) + value;}}
```

***Source code* 4.3** Penggambaran Spektrum Sinyal

```
else if (drawingType == WAVDrawingType.wavLeftType  )
   {
   for (int i = 0; i < dataStruct.dataSampleCount/2;
i++)
   {float value = (int)dataStruct.wavLeft [i];
    value = Math.Abs(value);
    if (value > maxValue)
     {
       maxValue = value;
     }}
      maxValue = maxValue + 50;
       float scale = (height / 2) / maxValue;
       for (int i = 0; i <
dataStruct.dataSampleCount/2; i++)
       {
        float value;
        value = dataStruct.wavLeft [i] * scale;
        graph.DrawLine(blackPen, xvalue, yvalue,
xvalue + step, (height / 2) + value);
        xvalue = xvalue + step;
        yvalue = (height / 2) + value;
       }}
else if (drawingType == WAVDrawingType wavRightType)
       {
   for (int i = 0; i < dataStruct.dataSampleCount/2;
i++)
    {float value = (int)dataStruct.wavRight [i];
   value = Math.Abs(value);
   if (value > maxValue)
     {
       maxValue = value;
     }}
       maxValue = maxValue + 50;
       float scale = (height / 2) / maxValue;
       for (int i = 0; i <
dataStruct.dataSampleCount/2; i++){
       float value;
       value = dataStruct.wavRight [i] * scale;
       graph.DrawLine(blackPen, xvalue, yvalue,
xvalue + step, (height / 2) + value);
       xvalue = xvalue + step;
       yvalue = (height / 2) + value;
       }}
```

***Source code* 4.3** Penggambaran Spektrum Sinyal

```
       else if (drawingType ==
       WAVDrawingType.wavFrekuensi1 ) {
  for (int i = 0; i < dataStruct.dataSampleCount;
i++){
  float value = (int)dataStruct.Frek1 [i];
  value = Math.Abs(value);
  if (value > maxValue) {
     maxValue = value; }}
     maxValue = maxValue + 50;
     float scale = (height / 2) / maxValue;
     for (int i = 0; i < dataStruct.dataSampleCount;
i++){
     float value;
     value = dataStruct.Frek1 [i] * scale;
     graph.DrawLine(blackPen, xvalue, yvalue, xvalue
+ step, (height / 2) + value);
     xvalue = xvalue + step;
     yvalue = (height / 2) + value; }}

     else if (drawingType == WAVDrawingType.
wavFrekuensi2       )
     { for (int i = 0; i <
dataStruct.dataSampleCount; i++){
       float value = (int)dataStruct.Frek2 [i];
       value = Math.Abs(value);
       if (value > maxValue){ maxValue = value;}
         }
       maxValue = maxValue + 50;
       float scale = (height / 2) / maxValue;
       for (int i = 0; i <
dataStruct.dataSampleCount; i++){
       float value;
       value = dataStruct.Frek2 [i] * scale;
       graph.DrawLine(blackPen, xvalue, yvalue,
xvalue + step, (height / 2) + value);
       xvalue = xvalue + step;
       yvalue = (height / 2) + value;}}

       drawGraphCoordinates(graph, blackPen, range,
0, dataStruct.dataSampleCount);

       graph.Dispose();
       this.BackgroundImage = bmp;
```

***Source code* 4.3** Penggambaran Spektrum Sinyal

4.3.3 Pengolahan Sinyal Digital(*Preprocessing*)

Setelah melalui proses pembacaan file wav, data akan mengalami beberapa tahap pada pengolahan sinyal digital yang terdiri dari *pre-emphasis*, *frame blocking*, *windowing*, transformasi *wavelet*, pengelompokkan data berdasarkan frekuensi, dan pemetaan PCP (*Pitch Class Profile)*.

4.3.3.1 *Pre-emphasis*

Sinyal suara yang merepresentasikan amplitudo dilewatkan pada suatu filter yang dinamakan *pre-emphasis*. Setiap baris data *mono* ataupun *stereo* dikalikan dengan dirinya sendiri yang sudah dikurangi dengan suatu koefisien tertentu dengan hasil yang disimpan dalam *array* satu dimensi tempdata1[i] untuk jenis suara *mono*, serta tempdata2[i] dan tempdata3[i] untuk jenis *stereo*. Sinyal dalam bentuk yang lebih halus pun berhasil dilakukan setelah melalui proses *pre-emphasis*. Dalam hal ini, koefisien berupa angka 0.95. *Source code* 4.4 merupakan *source code pre emphasis.*

```
if (wavFmtStruct.fmtChannels == 1) {
tempData1[0] = tempData[0];
   for (int i = 1; i < wavDataStruct.dataSize; i++)
     { tempData1[i] = (float)(tempData[i] - 0.95 *
tempData[i - 1]);
     }}
else {
tempData2[0] = tempDataLeft[0];
tempData3[0] = tempDataRight[0];
   for (int i = 1; i < wavDataStruct.dataSize/2; i++)
     {
      tempData2[i] = (float)(tempDataLeft[i] - 0.95 *
tempDataLeft[i - 1]);
      tempData3[i] = (float)(tempDataRight[i] - 0.95
* tempDataRight[i - 1]);
     }
     }
```

***Source code* 4.4** *Pre emphasis*

4.3.3.2 *Frame Blocking*

Untuk mempercepat jalannya proses, proses *pre-emphasis* dilanjutkan dengan proses *frame blocking* yang akan membagi data menjadi beberapa bagian menggunakan *jagged array*. Setiap *frame* terdiri dari 8192 data. Penentuan panjang *frame* ini berdasarkan *sample rate* sebesar 44100Hz. *Sample rate* ini mencakup minimal dua kali frekuensi tertinggi yang bisa dijangkau untuk pengenalan *chord*. Pembagian *frame* tidak harus bilangan pangkat dua sesuai dengan jumlah data yang bisa diolah pada proses transformasi. *Source code* 4.5 merupakan *source code* untuk proses *frame blocking*.

```
for (int i = 0; i < x; i++)
     if (i < x - 1)
     {
      dataA[i] = new float[panjang];
     }
     else
     {
      int q = nilai % panjang;
     if (q == 0)
      dataA[i] = new float[panjang];
     else
      dataA[i] = new float[q];
      }
for (int j = 0; j < dataA[i].Length; j++)
     {
     dataA[i][j] = tempData1[lg];
     lg++;
     }
```

***Source code* 4.5** *Frame Blocking*

4.3.3.3 *Windowing*

Windowing dengan metode pembobotan *hamming* akan mengakibatkan data mendekati nol. Metode ini dilakukan dengan mengalikan data pada tiap-tiap *frame* dengan *window hamming*, setelah melalui proses seleksi *mono* ataupun *stereo* sebelumnya. Hal

ini berhubungan dengan panjang data *stereo* dan *mono* yang berbeda. *Windowing* mengakibatkan sinyal tidak terlalu tinggi ataupun terlalu rendah. Sinyal awal dan akhir pada tiap *frame* akan mendekati nilai 0. *Source code* 4.6 merupakan *source code* pada tahap *windowing*

```
   {for (int i = 0; i < x; i++){
  tempWindow[j] = (float)(0.54 - (0.46 * (Math.Cos((2
* 180 * j) / (panjang - 1)))));

    for (int j = 0; j < dataA[i].Length; j++)
    {
      dataA[i][j] = tempData1[lg];
      lg++;
   dataAa[j]=(float)Math.Round((dataA[i][j]
*tempWindow[j]),0);
    }
```

***Source code* 4.6** *Windowing*

4.3.3.4 Transformasi *Wavelet*

Data hasil proses *windowing* diolah untuk menjadi data dalam domain frekuensi dengan *Maximal Overlap Discrete Wavelet* (MODWT) jenis Haar. Matriks *wavelet* diciptakan sesuai dengan lebar *frame* sejumlah N, jadi tidak akan ada peristiwa *downsampling* jika menggunakan jenis transformasi ini. Matriks berdimensi NxN akan dikalikan dengan data pada tiap *frame* dan akan mengalami beberapa kali iterasi tertentu sesuai dengan jumlah data. Tidak adanya *downsampling* pada jenis transformasi ini memungkinkan semua data bisa ditransformasikan. Karena kebutuhan sistem berupa data dalam domain frekuensi, maka jenis koefisien yang digunakan adalah jenis koefisien *wavelet* berorde rendah yang akan mengenali frekuensi dengan rata-rata tinggi. Transformasi ini akan menghasilkan keluaran sinyal dalam domain frekuensi yang tidak terlalu jauh berbeda dengan sinyal dalam domain waktu setelah melalui pergeseran sesuai dengan level suatu data tersebut. Suatu matriks a[m,n] akan dibentuk pada proses ini sesuai dengan jenis filter untuk kemudian dikalikan dengan hasil dari proses *windowing* yang telah dibulatkan sesuai dengan syarat MODWT dengan

masukan data integer atau diskrit. *Source code* 4.7 merupakan *source code* untuk transformasi *wavelet* jenis MODWT.

```
int itr = (int)(Math.Log(dataA[i].Length) /
(Math.Log(Math.E)));

int max = dataA[i].Length;
int atas = max;
int bawah = max;
float[,] a = new float[max, max];
float[] hasil = new float[max];
int m = 1, n = max - 1;

if (itr == 0)
{
  for (int j = 0; j < dataA[i].Length; j++)
  {
    dataAa[j] = dataAa[j];
  }
}
else
{
  for (int d = 1; d <= itr; d++)
  {
    a = matrik(a, m, n, atas, bawah);
    for (int r = 0; r < max; r++)
    {
     float temp = 0;
       for (int s = 0; s < max; s++)
       {
        temp += a[r, s] * dataAa[s];
       }
       hasil[r] = temp;
     }
    for (int r = 0; r < dataA[i].Length; r++)
    {
      dataAa[r] = hasil[r];
    }
    m += (int)((Math.Pow(2, d - 1) - 1) + 1);
    n -= (int)((Math.Pow(2, d - 1) - 1) + 1);
    }
```

***Source code* 4.7** Transformasi *Wavelet*

```
    for (int r = 0; r < dataA[i].Length; r++)
    {
    dat.Add(dataAa[r]);
    }
   }
  }

public float[,] matrik(float[,] a, int m, int n, int
atas, int bawah) {

  for (int x = 0; x < atas; x++){
     for (int y = 0; y < bawah; y++)
      {  if (x == y)
            a[x, y] = (float)0.5;
         else if (x == y + m)
            a[x, y] = (float)-0.5;
         else if (x == y - n)
            a[x, y] = (float)-0.5;
         else
            a[x, y] = 0;
       }
     }
        return a;
    }
```

***Source code* 4.7** (lanjutan) Transformasi *Wavelet*

4.3.3.5 Pengelompokan Frekuensi

Setelah data sudah berupa frekuensi, maka pola *chord* sudah mulai terlihat. Proses ini akan mengelompokkan *chord* minimal 3 titik *sample* yang memenuhi frekuensi 12 nada dengan 9 oktaf. Indeks awal dan akhir dalam satu kelompok disimpan terlebih dahulu dalam *array list* untuk kemudian dipanggil sebagai batasan awal dan akhir kelompok frekuensi tertentu yang disimpan dalam *array*. Tiap-tiap kelompok ini yang akan digunakan sebagai masukan pada proses selanjutnya. *Source code* 4.8 merupakan *source code* pengelompokan frekuensi.

```
while (index_st < mot.Length - 1)
{
 if (mot[index_st] > max)
 { max = mot[index_st];
   index_max = index_st;
   index_st++; }
else index_st++;
}
  //cari kiri max
  index_st = index_max;
  index_kiri = index_st - kons;
  while (index_st > 0)
  {
  if (mot[index_st - kons] == mot[index_st])
  {
   index_kiri = index_st - kons;
   index_st--;
  }
   else
  {
    break;
   }
 }
 //cari kanan max
 index_st = index_max;
 index_kanan = index_st + kons;
 while (index_st < mot.Length)
 {
 if (mot[index_st + kons] == mot[index_st])
  {
    index_kiri = index_st+kons;
    index_st++;
  }
 else
  {
   break;
  }
  }
  for (int i = index_kiri; i <= index_kanan; i++)
  {
   temp.Add(i);}
   temp.Add(-1);
```

***Source code* 4.8** Pengelompokan Frekuensi

4.3.3.6 Pemetaan *Pitch Class Profile* (PCP)

Setiap kelompok gunung pada proses sebelumnya akan dilanjutkan untuk dipetakan sesuai dengan 12 nada kromatik. Pemetaan ini akan menghasilkan 12 koefisien usetiap kelompok frekuensi (gunung). Setelah itu koefisien-koefisien ini diskalakan pada angka 0 dan 1. Pemetaan dilakukan dengan melakukan pengurangan tiap-tiap frekuensi titik *sample* dengan frekuensi yang disimpan dalam file Oktaf.txt sesuai dengan 12 nada tersebut tanpa memperhatikan letak oktaf. Setelah itu, nilai minimal tiap nada akan ditemukan dengan suatu integer pembanding dibawah frekuensi minimum yang digunakan dalam penelitian ini. Setelah terbentuk pemetaan maka angka biner pada skala tersebut diganti dengan jenis angka bipolar, dan kemudian dikalikan dengan transposenya. Matriks inilah yang digunakan untuk proses pelatihan. *Source code* 4.9 merupakan *source code* pemetaan *pitch class profile.*

```
 for (int r = awal; r <akhir; r++)
 {if (((issi[r] >= 32.703) && (issi[r] <= 15804)))
   { float minim = 32;
     simpan[ind] = issi[r];
     for (int i = 0; i < 12; i++)
     {
      for (int j = 0; j < 9; j++)
      {
       hasil_pcp[i, j] = Math.Abs(nada[i, j] -
simpan[ind]);
       if (hasil_pcp[i, j] <= minim)
       {
         minim = hasil_pcp[i, j];
         index_min = i;
       }}}
      indexs[ind] = index_min;
      int k = 0;
      for (int i = 0; i < 12; i++)
      {
      if (i == indexs[ind])
      k = 1;
      else
      k = 0;
      Kordd[i] = k;
      Kords[i] += Kordd[i];
   }
  ind++;}
}
 for (int i = 0; i < 12; i++)
 {
  if (Kords[i] != 0)
    Kords[i] = 1;
 if (Kords[i] == 0)
    Kords[i] = -1;
    jumlah += Kords[i];
  }
```

***Source code* 4.9** Pemetaan *Pitch Class Profile*

4.3.4 Pelatihan dan Pengenalan

Dua proses terakhir yang harus dilalui untuk pengenalan *chord* adalah pelatihan dengan jaringan syaraf tiruan Hopfield, dan pengenalan dengan *Euclidean Distance.*

4.3.4.1 Pelatihan

Proses pelatihan pada sistem ini menggunakan jaringan syaraf tiruan Hopfield. Data akan di *training* sampai sepuluh kali iterasi sehingga memungkinkan data semakin dekat dengan kondisi konvergen. Data t*raining* akan melalui proses *training* dengan 24 *chord* mayor dan minor yang cirinya sudah disimpan dalam file-file *chord*.txt. File- file tersebut merupakan matriks yang akan digunakan sebagai penguji data masukan yang didapat dari perkalian angka bipolar tiap pola *chord* dengan transposnya dengan diagonal utama berisi angka 0. Hasil dari pelatihan ini berupa 24 buah matriks hasil pelatihan data uji dengan 24 data latih yang berasal dari pola data asal. *Source code* 4.10 merupakan *source code* pelatihan dengan JST Hopfield.

```
 for (int s = 0; s < 5; s++)
 {
  for (int k = 0; k < 24; k++)
  {
   for (int g = 0; g < 12; g++)
   {
    float[] Hasilkali = new float[24];
    for (int w = 0; w < 12; w++)
    {
     Hasilkali[w] = Uji[k][g, w];
    }
    for (int m = 0; m < 12; m++)
    { for (int n = 0; n < 12; n++)
     {
       Hasilkali[m] += chords[k][g, n] * Uji[k][n, m];
      }
     }
   for (int a = 0; a < 12; a++)
   {
    if (Hasilkali[a] >= 0)
        Uji[k][g, a] = 1;
    else
        Uji[k][g, a] = -1;
}
}
}
}
```

***Source code* 4.10** Pelatihan dengan JST Hopfield

4.3.4.2 Pengenalan

Pengenalan dilakukan dengan perhitungan *Euclidean Distance.* Perhitungan dilakukan pada tiap-tiap *chord* sebanyak 24 kali. Dari hasil perhitungan *chord input* dengan *chord* asal pada proses ini akan dapat diketahui *chord-chord* hasil pengenalan dengan jarak Euclidean terkecil. Matriks tiap-tiap hasil pelatihan akan dicari jarak terpendeknya dengan tiap-tiap matriks data asli. *Source code* 4.11 merupakan *source code* pengenalan *Euclidean Distance.*

```
for (int k = 0; k < 24; k++)
{
  for (int i = 0; i < 12; i++)
{
    for (int j = 0; j < 12; j++)
     {
     totalSel[k][i] +=
(float)Math.Pow((chordasal[k][i, j] - Uji[k][j, i]),
2);
     }
     hasilECLD[k] += totalSel[k][i];
     ECLD[k] = (float)Math.Sqrt(hasilECLD[k]);
   }
  }}

for (int a = 0; a < 24; a++)
{
  for (int j = intLower0; j <= intUpper0; j++)
  {

       Minimal.Add(ECLD[j]);
       yMin[j] = Minimal.Min();
       if (yMin[j] == ECLD[0])
       {
         yM = "Am";
       }
       else if (yMin[j] == ECLD[1])
       {
         yM = "A";
       }
       else if (yMin[j] == ECLD[2])
       {
         yM = "A#m";
       }
       else if (yMin[j] == ECLD[3])
       {
         yM = "A#";
       }
       else if (yMin[j] == ECLD[4])
       {
         yM = "Bm";
       }
```

***Source code* 4.11** Pengenalan dengan *Euclidean Distance*

```
      else if (yMin[j] == ECLD[5])
      {
         yM = "B";
      }
      else if (yMin[j] == ECLD[6])
      {
         yM = "Cm";
      }
      else if (yMin[j] == ECLD[7])
      {
         yM = "C";
      }
      else if (yMin[j] == ECLD[8])
      {
         yM = "C#m";
      }
      else if (yMin[j] == ECLD[9])
      {
         yM = "C#";
      }
      else if (yMin[j] == ECLD[10])
      {
         yM = "Dm";
      }
      else if (yMin[j] == ECLD[11])
      {
         yM = "D";
      }
      else if (yMin[j] == ECLD[12])
      {
         yM = "D#m";
      }
      else if (yMin[j] == ECLD[13])
      {
         yM = "D#";
      }
      else if (yMin[j] == ECLD[14])
      {
         yM = "Em";
      }
      else if (yMin[j] == ECLD[15])
      {
        yM = "E";
      }
```

***Source code* 4.11** (lanjutan) Pengenalan dengan *Euclidean Distance*

```
      else if (yMin[j] == ECLD[16])
      {
        yM = "Fm";
      }
      else if (yMin[j] == ECLD[17])
      {
        yM = "F";
      }
      else if (yMin[j] == ECLD[18])
      {
        yM = "F#m";
      }
      else if (yMin[j] == ECLD[19])
      {
        yM = "F#";
      }
      else if (yMin[j] == ECLD[20])
      {
        yM = "Gm";
      }
      else if (yMin[j] == ECLD[21])
      {
        yM = "G";
      }
      else if (yMin[j] == ECLD[22])
      {
        yM = "G#m";
      }
      else if (yMin[j] == ECLD[23])
      {
        yM = "G#";
      }
     }
  } hitung++;
```

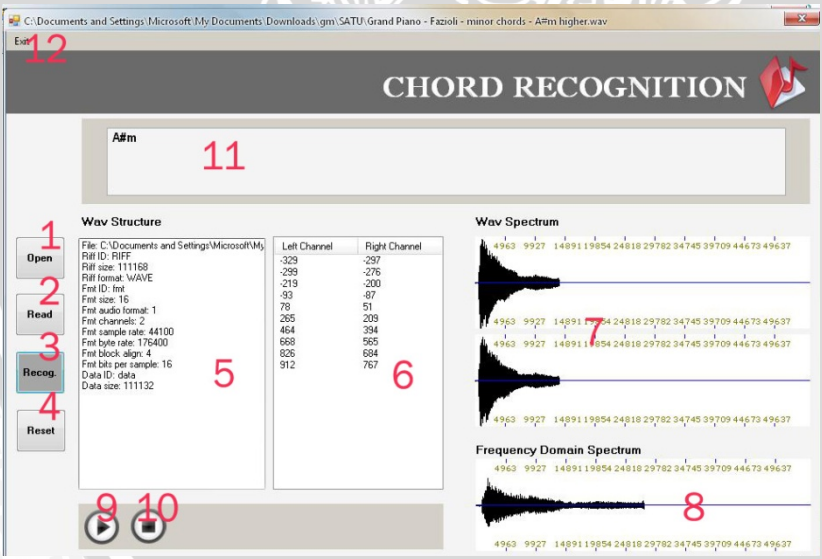
***Source code* 4.11** (lanjutan) Pengenalan dengan *Euclidean Distance*

4.4 Implementasi *Interface* (Antarmuka)

User Interface aplikasi *Chord Recogition* dengan Haar *Wavelet* dan Jaringan Syaraf Tiruan Hopfield menggunakan 1 *form* yang ditunjukkan pada gambar 4.1. *Form* dengan *button open* pada no.1 untuk membukan direktori wav, *button read* ditunjukkan dengan no.2 untuk pembacaan file wav, button *recognize* pada no.3 yang

akan menampilkan *chord-chord* hasil pengenalan pada *listview* yang ditampilkan pada *textbox* no.6, *button reset* pada no.4 untuk pengenalan data wav baru, WAV*DrawingPanel* pada no. 7 dan no.8 untuk menampilkan *spectrum* sinyal, dan *image* mini media *player* untuk *play* pada *no.9*, dan *stop* pada no 10. Hasil pengenalan ditunjukkan pada *textbox* pada no.11. Untuk keluar dari aplikasi ini disediakan menu *exit* yang ditampilkan pada no. 12. Peletakkan bagian-bagian pada *form* dapat dilihat pada gambar 4.1.

Saat pertpama kali menggunakan aplikasi ini proses membuka file wav dilakukan terlebih dahulu dengan. Setelah itu jika button "*read*" ditekan aplikasi akan memproses informasi *header* wav, dan data wav yang sudah dalam bentuk data diskrit. Secara otomatis titik *sample* data akan bisa dilihat pada tabel. Setelah itu untuk melakukan proses *training* dan pengenalan dapat dilakukan dengan menekan *button* "recog.". Proses *training* dengan Jaringan Syaraf Tiruan Hopfield dan pengenalan dengan *Euclidean Distance* akan diproses. Setelah proses selesai *chord* hasil pengenalan akan muncul pada *textbox*.



Gambar 4.1 Aplikasi Pengenalan *Chord*

4.5 Pengujian dan Analisis Hasil

Setelah aplikasi telah berhasil dibuat, diperlukan adanya pengujian dan analisis hasil dari aplikasi tersebut. Pengujian dan analisis tersebut melalui perhitungan matematika untuk mendapatkan tingkat keberhasilan dari aplikasi yang telah dibuat.

4.5.1 Pengujian

Tahap pengujian ini dilakukan pada tiga tahap:

Pengujian dilakukan pada 30 buah data yang terdiri dari beberapa alat music, seperti gitar, keyboard, dan piano. Data terdiri dari dua jenis file wav, yaitu stereo dan mono. Hasil pengenalan akan dibandingkan dengan *chord-chord* asli.

4.5.2 Hasil dan Analisa Pengujian

a. Percobaan pertama

Pengujian pertama dilakukan pada 10 data yang berisi file wav dengan satu *chord* penyusun. *Chord-chord* yang digunakan dalam percobaan pertama ini menggunakan *chord-chord* mayoritas frekuensi ada oktaf rendah atau menengah. Alat musik yang digunakan adalah keyboard, gitar, dan piano. Tingkat keberhasilan rata-rata tidak terlalu baik dalam proses pengenalan.

$$Akurasi = \frac{6}{10} x\ 100\% = 60\%$$

Akurasi ini dipengaruhi oleh posisi oktaf yang terlalu rendah atau menengah, serta perekaman yang tidak mendukung terbentuknya sinyal suara yang baik dan minim *noise*.

Tabel 4.2 Hasil uji coba tahap pertama

No	Nama *Chord*	Jumlah *Chord*	Jumlah *Chord* Benar	Akurasi %
1	**A#m**	**1**	**1**	**100**
2	**A#**	**1**	**0**	**0**
3	**A**	**1**	**0**	**0**
4	**Am**	**1**	**1**	**100**
5	**C#m**	**1**	**1**	**100**
6	**B**	**1**	**0**	**0**
7	**Cm**	**1**	**1**	**100**
8	**A(2)**	**1**	**1**	**100**
9	**Cm**	**1**	**0**	**0**
10	**D**	**1**	**0**	**0**

b. Percobaan Kedua

Perobaan kedua ini dilakukan pada 10 buah file, yang tiap file nya terdiri dari chord-chord untuk pembuktian keberhasilan penelitian. Tingkat keberhasilan pada percobaan kedua ini menurun dari percobaan pertama karena banyaknya sinyal yang terpotong, sehingga menyebabkan pengelompokan sinyal menjadi tidak sesuai dengan sinyal asli.

$$Akurasi = \frac{2}{10} x\ 100\% = 20\%$$

Tabel 4.3 Hasil uji coba tahap kedua

No	Nama *Chord*	Jumlah *Chord*	Jumlah *Chord* Benar	Akurasi %
1	Em	1	0	0
2	F	1	0	0
3	G#m	1	0	0
4	A	1	1	100
5	Am	1	1	100
6	Gm	1	0	0
7	G	1	0	0
8	G#m(2)	1	0	0
9	F(2)	1	0	0
10	Dm	1	0	0

c. Percobaan Ketiga

Percobaan ketiga ini dilakukan pada 10 buah file chord. Percobaan ini mengalami peningkatan pengenalan. Mayoritas chord yang dikenalai berada pada nada-nada beroktaf tinggi. Alat music yang berbeda ternyata juga berpengaruh pada pengenalan chord ini.

$$Akurasi = \frac{4}{10} x\ 100\% = 40\%$$

Tabel 4.4 Hasil uji coba tahap ketiga

No	Nama *Chord*	Jumlah *Chord*	Jumlah *Chord* Benar	Akurasi %
1	Fm	1	0	0
2	Bm	1	1	100
3	A	1	1	100
4	D#	1	1	100
5	B	1	0	0
6	Bm(2)	1	1	100
7	C#	1	0	0
8	D	1	0	0
9	E	1	0	0
10	Fm(2)	1	0	0

Dari hasil penelitian secara keseluruhan. Tingkat akurasi dalam penelitian ini adalah 40%. Hasil ini menunjukkan bahwa metode transformasi wavelet Haar menggunakan MODWT tidak tapat pada data sinyal karena pemrosesan pada seluruh data berselang dengan nol sesuai dengan suatu level tertentu, selain itu pembulatan angka setelah proses windowing mengakibatkan banyak informasi yang hilang. Itu membuktikan bahwa semakin rumit gabungan data, maka akan semakin sulit pula kemungkinan data untuk dikenali.

BAB V
PENUTUP

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini adalah:

1. Pengolahan sinyal digital dilakukan untuk dapat mengambil suatu informasi yang dibutuhkan pada suatu sinyal yang masih berupa analog.
2. Hasil pengenalan sangat dipengaruhi oleh proses transformasi *wavelet* Haar jenis *Maximal Overlap Discrete Wavelet* (MODWT) yang menggunakan algoritma piramida dengan penambahan selang nol setiap levelnya. Selang ini menyebabkan tidak semua data dihitung dengan merata.
3. Hasil pelatihan menggunakan Jaringan Syaraf Tiruan Hopfield yang selanjutnya dikenali dengan pencarian jarak terpendek menggunakan *Euclidean Distance* menghasilkan data bisa dikenali dengan mudah.
4. Hasil pengujian sistem berupa identifikasi tingkat akurasi menggunakan nilai rata-rata data benar dan total data. Tingkat total akurasi yang didapat adalah 40% dan tingkat kesalahan 60% dengan 30 data uji dari masukan *chord* mayor dan minor yang berbeda, dan beberapa jenis alat musik.

5.2 Saran

Dari hasil penelitian ini dapat dikemukakan beberapa saran untuk memperbaiki kinerja sistem aplikasi pengenalan *chord* adalah :

1. Pengenalan *chord* hendakya di transformasi pada tiap-tiap titik sample dengan teliti dari awal dan akhir sinyal secara menyeluruh yang bisa juga dilakukan dengan metode *Discrete Wavelet Transform* (DWT).
2. Pencarian data dari alat musik yang lebih beragam dapat dilakukan untuk mengetahui tingkat keakuratan yang lebih teliti.

DAFTAR PUSTAKA

Anonymous. *Note Frequencies.* http://www.seventhstring.com/resources/notefrequencies.html/ Tanggal akses 30 Oktober 2010.

Anurogo, D. *Manfaat Musik Klasik bagi Kesehatan.* http://www.kabarindonesia.com/berita.php?pil=3&jd=Manfaat+Musik+Klasik+bagi+Kesehatan&dn=20071220213230/. Tanggal akses 29 Oktober 2010.

Basuki A., Huda M., dan Amalia T. 2006. *Aplikasi Pengolahan Suara untuk Request Lagu.* Institut Teknologi Sepuluh November. Surabaya.

Brodjol, Suhartono. 2009. *Neural Networks Multiscale Autoregressive Model For Forecasting Seasonal Time Series Data.* Institute Teknologi Sepuluh November. Surabaya.

Chahyati D. 2003. *Wavelet.* http://www.cs.ui.ac.id/WebKuliah/citra/2003/Wavelet1.doc/. Tanggal akses 2 Oktober 2010.

Clifford Gower, John. 2006. *Euclidean Distance Matrix*. http://www.stanford.edu/~dattorro/. Tanggal akses 2 Oktober 2010.

Gunawan I., Gunadi K.. 2010. *Pembuatan Perangkat Lunak Wave Manipulator Untuk Memanipulasi File Wav.* Universitas Kristen Petra.

Khadkevich M., Omologo M. 2009. *Phase-Change Based Tuning For Automatic Chord Recognition.* University of Degli. Trento.

Kristanto A. 2004. *Jaringan Syaraf Tiruan (Konsep Dasar, Algoritma, dan Aplikasi).* Yogyakarta: Gava Media

Lee, Kyogu, dan Malcom S. 2006. *Automatic Chord Recognition form Audio using an HMM with Supervised Learning.* University of Victoria.

Melinda. *Pengaruh Pendidikan Seni terhadap Perkembangan Otak Anak.* http://www.melindahospital.com/modul/user/detail_artikel.php?id=893_Pengaruh-Pendidikan-Seni-terhadap-Perkembangan-Otak-Anak. Tanggal akses 29 Oktober 2010.

Pasaribu A..*Tangga Nada.* http://solfegio.indonesianforum.net/teori-musik-f6/tangga-nada-t134.htm/. Tanggal akses 29 Oktober 2010.

Percival, D.B., and A.T. Walden. 2000. *Wavelets Methods for Time Series Analysis.* Cambridge University Press. Cambridge.

Putra, Andika B., Iwut I., dan Haryanto J. 2007. *Speech Recognition Menggunakan Gabor Wavelet dan Jaringan Syaraf Tiruan Backpropagation Untuk Sistem Keamanan Berbasis Suara.* STT Telkom. Bali.

Sambu, Gari R. 2008. *Pintar Main Gitar dalam 7 Hari.* Yogyakarta : Media Pressindo.

Somantri Y., Haritman E.. *Pengenalan Pembicara dengan Ekstraksi Ciri MFCC Menggunakan Kuantisasi Vektor (VQ).* http://file.upi.edu/Direktori/E%20%20FPTK/JUR.%20PEND.%20TEKNIK%20ELEKTRO/195708051985031%20%20YOYO%20SOMANTRI/HASIL%20PENL.%20Yoyo/Presentasi_vq_rev.pdf/. Tanggal akses 6 Oktober 2010.

Surdulescu R. 2003. *A Gentle Introduction to Wavelets.* http://www.sonic.net/~surdules/articles/gentle_wavelets/index.html. Tanggal akses 25 November 2010.

Tanudjaja H. 2007. *Pengolahan Sinyal Digital dan Sistem Pemrosesan Sinyal.*Yogyakarta: Andi

Wibisono Y. 2010. *Klasifikasi Berita Berbahasa Indonesia menggunakan Naïve Bayes Classifier*. Jurusan Pendidikan Matematika FPMIPA UPI. Bandung.

Wilson S. 2003. *WAVE PCM soundfile format.* https://ccrma.stanford.edu/courses/422/projects/WaveFormat/. Tanggal akses 2 September 2010.

Huda Miftachul, Drs, MT, Basuki Kurnia Dwi, S.Si, M.Kom, Akbar Fandy, Permana Junaidy Febrianzah. 2010. *Konversi nada-nada Akustik Menjadi Chord Menggunakan Pitch Class Profile*. ITS. Surabaya.


UNIVERSITAS BRAWIJAYA