PENGGOLONGAN SUARA MANUSIA DALAM CITRA PARTITUR VOKAL MENGGUNAKAN MODIFIKASI ALGORITMA *KIM'S OPTICAL MUSIC RECOGNITION* (KOMR)

SKRIPSI

oleh :

YUNANTO DWI PURNOMO
0410960066-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2011

PENGGOLONGAN SUARA MANUSIA DALAM CITRA PARTITUR VOKAL MENGGUNAKAN MODIFIKASI ALGORITMA *KIM'S OPTICAL MUSIC RECOGNITION* (KOMR)

SKRIPSI

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

oleh :

YUNANTO DWI PURNOMO
0410960066-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2011

UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENGGOLONGAN SUARA MANUSIA DALAM CITRA PARTITUR VOKAL MENGGUNAKAN MODIFIKASI ALGORITMA *KIM'S OPTICAL MUSIC RECOGNITION* (KOMR)

Oleh:
YUNANTO DWI PURNOMO
0410960066-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 8 Agustus 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana dalam bidang Ilmu Komputer

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

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc.
NIP. 196709071992031001

Telah dipertahankan
di depan Majelis Penguji
Pada tanggal 5 Agustus 2008

Tugas Akhir ini telah diterima sebagai salah satu persyaratan untuk memperoleh gelar
Sarjana dalam Bidang Ilmu Komputer

Pembimbing I

Wayan Firdaus Mahmudy, S.Si., M.T.

Pembimbing II

Agus Wahyu Widodo, S.T., M.Cs.

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Agus Suryanto, M.Sc., Ph.D.

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama	**: Yunanto Dwi Purnomo**
NIM	**: 0410960066**
Jurusan	**: Matematika**
Penulis skripsi berjudul	**:Penggolongan Suara Manusia dalam Citra Partitur Vokal Menggunakan Modifikasi Algoritma *Kim's Optical Music Recognition* (KOMR)**

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 di kemudian 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, 8 Agustus 2011

Yang menyatakan,

(Yunanto Dwi Purnomo)
NIM. 0410960066

UNIVERSITAS BRAWIJAYA

PENGGOLONGAN SUARA MANUSIA DALAM CITRA PARTITUR VOKAL MENGGUNAKAN MODIFIKASI ALGORITMA *KIM'S OPTICAL MUSIC RECOGNITION* (KOMR)

ABSTRAK

Musik merupakan bahasa universal. Cara baku untuk menuliskan not-not musik ke dalam suatu media disebut partitur. Dalam partitur vokal, tertulis nada-nada yang harus dinyanyikan oleh manusia. Suara manusia dapat digolongkan berdasarkan range atau nada-nada yang terbatas. Seringkali seorang penyanyi pemula merasa kesulitan untuk membawakan suatu partitur, karena nada yang tertera terlalu tinggi atau terlalu rendah untuk dicapai. Maka pada penelitian ini akan dihasilkan model perangkat lunak penggolongan suara yang dapat mengatasi kelemahan tersebut.

Algoritma yang digunakan adalah modifikasi dari *Kim's Optical Music Recognition (KOMR)*. Uji coba diterapkan pada 5 citra partitur vokal. Masing–masing partitur diambil 5 potongan birama dari 4 golongan suara, yaitu sopran, alto, tenor, dan bass, sehingga diperoleh data percobaan sebanyak 100.

Hasil pengujian menunjukkan bahwa algoritma *Kim's Optical Music Recognition* yang telah dimodifikasi dapat diimplementasikan dalam model perangkat lunak untuk penggolongan suara manusia dalam citra partitur vokal. Persentase keakuratan penggolongan suara manusia sangat dipengaruhi oleh pemilihan potongan citra. Persentase keakuratan perangkat lunak penggolongan suara manusia dalam citra partitur pada percobaan sebanyak 25 data untuk masing-masing golongan suara yaitu sopran sebesar 84%, alto sebesar 96%, tenor sebesar 100%, dan bass sebesar 84%. Perbedaan persentase disebabkan pada data uji coba, terdapat not-not yang melebihi nilai threshold pada golongan suara sopran, alto, dan bass.

HUMAN VOICE CLASSIFICATION
IN VOCAL SCORE IMAGE
BY USING MODIFIED ALGORITHM
***KIM'S OPTICAL MUSIC RECOGNITION* (KOMR)**

ABSTRACT

Musik has been considered as universal language. Standard way to write musical notes into the media was called score. In vocal score, there are some notes that must be sung by human. Human voice can be classified based on ranges or limited tones. Occasionally an amateur singer feels too hard to perform some scores, because their notes are way too high or too low to be reached. Hence in this research, software model voice classification that can solve these problems will be created.

Algorithm which will be used is modified Kim's Optical Music Recognition (KOMR). The tests were performed on 5 vocal score image. Each score was taken 5 cropped bar from 4 voice classes, sopran, alto, tenor, and bass, so the number of data tests would be 100 datas.

Testing results show that modified Kim's Optical Music Recognition algorithm could be implemented in software model to classify human voice in vocal score image. Accuracy percentage from human voice classification was greatly influenced by cropped image choices. Accuracy percentage in the test from 25 datas from each vocal range, sopran for 84%, alto for 96%, tenor for 100%, and bass for 84%. The difference of accuracy result were caused by the anomaly of data tests. There were some notes that exceed threshold point in voices range, sopran, alto, and bass.

KATA PENGANTAR

Puji Syukur kehadirat Allah *Subhanahu wa ta'ala*, yang telah melimpahkan rahmat dan hidayah-Nya, sehingga skripsi yang berjudul "Penggolongan Suara dalam Citra Partitur Vokal Menggunakan Modifikasi *Kim's Optical Music Recognition* (KOMR)" ini dapat diselesaikan. Skripsi ini disusun sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, Fakultas MIPA, Universitas Brawijaya.

Shalawat serta Salam tidak lupa penulis curahkan kepada junjungan Nabi besar Rasulullah *Shallahu 'alaihi wasalam*, keluarga, shahabat, dan para pejuang risalah yang disampaikannya sampai akhir zaman.

Dalam menyelesaikan skripsi ini, penulis telah banyak mendapat bantuan dari berbagai pihak. Untuk itu Penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Candra Dewi, S.Kom., M.Sc. selaku pembimbing skripsi I
2. Drs. Marji, MT selaku pembimbing Skripsi II sekaligus Ketua Program Studi Ilmu Komputer
3. Dr. Abdul Rouf A., M.Sc. selaku Ketua Jurusan Matematika.
4. Edy Santoso,SSi.,M.Kom selaku penasihat akademik.
5. Segenap Bapak dan Ibu dosen yang telah mendidik penulis selama penulis menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
6. Bapak, Ibu, serta Kakak yang senantiasa memberi do'a dan motivasi.
7. Segenap staff dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya.
8. Sahabat terbaik penulis, Susmawan Dita Afianto dan Arfebriyan Wahyu Krisna yang memberikan motivasi untuk terus maju.
9. Sahabat-sahabat Paduan Suara Mahasiswa Universitas Brawijaya (*Brawijaya University Student Choir*) yang memberikan dukungan serta semangat.
10. Semua rekan Ilmu Komputer angkatan 2004.
11. Pihak lain yang tidak bisa penulis sebutkan satu persatu

Penulis menyadari bahwa skripsi ini jauh dari sempurna dan banyak kekurangan. Salinan skripsi, kritik, dan saran dapat diperoleh dan disampaikan melalui email di suratiaku2@yahoo.com. Akhir kata, penulis berharap semoga skripsi ini dapat memberikan sesuatu yang bermanfaat bagi semua pihak yang membacanya.

Malang, Agustus 2011

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Musik merupakan bahasa universal yang digunakan manusia di seluruh dunia. Untuk menyebarkan dan memindahkan informasi musik ke masyarakat luas, para penggubah lagu memerlukan suatu media baku. Pada jaman dahulu, para penggubah memindahkan informasi mengenai lagu kepada orang dengan cara menyanyikan langsung lagu yang digubah, dan orang tersebut harus mengingat dengan tepat tinggi rendahnya nada dan lama nada yang dinyanyikan. Cara ini mempunyai banyak kelemahan, karena kemampuan tiap orang berbeda. Pada akhirnya ketika sampai ke orang yang paling akhir, lagu tersebut seringkali tidak tepat nadanya, maupun durasi yang dinyanyikan menjadi lebih pendek atau panjang.

Kemudian pada awal abad ke-11, biarawan Benediktin, Guido d'Arezzo menemukan cara penulisan not-not musik dengan simbol-simbol yang dituliskan pada dokumen, yang menunjukkan tinggi rendahnya nada yang dicapai, serta durasi nada yang harus dinyanyikan. Dokumen standar ini kemudian disebut partitur musik. Partitur berisi lima garis horizontal yang saling berjarak sama. Garis ini disebut paranada. Selanjutnya garis ini berisi simbol-simbol yang menunjukkan tinggi rendahnya nada, durasi nada, serta waktu untuk berhenti. Sejak itu musik mengalami perkembangan yang signifikan. (Feldman, 2007)

Pada perkembangannya, pemusik saat ini memainkan nada-nada sesuai partitur yang diberikan oleh penggubah lagu. Bila seorang penyanyi harus menyanyikan sebuah lagu, maka ia harus menyanyikan seberapa tinggi dan durasi masing-masing nada sesuai dengan partitur. Penyanyi profesional tidak mengalami kesulitan untuk membaca partitur yang diberikan kepadanya. Namun seringkali penyanyi pemula yang belum mengenal not balok mengalami kesulitan untuk menyanyikan nada-nada yang diminta oleh partitur, karena nada-nada tersebut terlalu rendah atau terlalu tinggi.

Setiap alat musik, termasuk suara manusia memiliki batas nada terendah atau tertinggi yang dapat dikeluarkan. Bila nada yang diminta melebihi jangkauan ini, maka suara yang akan dihasilkan

menjadi tidak nyaman didengar, atau bahkan tidak terdengar sama sekali. Jangkauan nada ini disebut *range*.

Untuk mengatasi masalah tersebut, diperlukan suatu aplikasi yang dapat memperkirakan batas-batas nada dalam partitur yang dapat dinyanyikan oleh seorang penyanyi dengan baik. Nada-nada itu harus terletak dalam jangkauan suara penyanyi tersebut. Aplikasi ini nantinya akan membaca kisaran not-not yang tertulis dalam partitur, dan digunakan untuk menentukan apakah not-not tersebut digunakan oleh penyanyi wanita bersuara tinggi (disebut soprano), atau bersuara rendah (alto). Bila dinyanyikan oleh pria, maka yang bersuara tinggi disebut tenor, dan yang bersuara rendah disebut bass.

Citra partitur merupakan citra khusus yang memerlukan penanganan berbeda dibandingkan dengan citra lain. Garis paranada yang terdapat dalam partitur tidak dapat dihilangkan begitu saja, karena menyebabkan banyak informasi yang hilang (*loss of information*). Di dalam teknologi informasi, aplikasi yang menangangi citra partitur untuk diambil informasinya disebut *Optical Music Recognition*. Salah satu algoritma dalam bidang OMR ini dibuat oleh Kim. Algoritma Kim's Optical Music Recognition dibuat untuk membaca sebuah partitur musik dan memperoleh informasinya. Informasi ini digunakan untuk memainkan musik pada lengan robot.

Untuk membuat aplikasi ini digunakan pemrograman citra digital berdasarkan modifikasi algoritma *Kim's Optical Music Recognition*. Modifikasi ini perlu dilakukan untuk mempermudah langkah-langkah yang perlu dilakukan. Algoritma ini meliputi *image data acquisiton, preprocessing, coarse classification, fine classification, dan music syntax check* (Kim, 1987). Dari proses tersebut, not-not dalam partitur akan dapat dikenali. Selanjutnya dilakukan pengukuran jarak not-not yang akan dinyanyikan.

1.2 Rumusan Masalah

Mengacu permasalahan yang diuraikan dalam latar belakang, maka rumusan masalah bisa ditekankan pada:

1. Bagaimana mengimplementasikan modifikasi *Kim's Optical Music Recognition* dalam perangkat lunak yang dapat melakukan penggolongan suara pada citra partitur sesuai dengan langkah-langkah yang disebutkan.

2. Berapa akurasi sistem untuk dapat melakukan penggolongan suara pada citra partitur.

1.3 Batasan Masalah

Dalam perencanaan dan pembuatan skripsi ini perlu dilakukan pembatasan masalah. Pembatasan masalah yang diajukan dalam skripsi ini antara lain:

1. Citra partitur yang digunakan merupakan partitur vokal
2. *Input* yang digunakan berupa potongan satu birama pada citra partitur, dan *output* yang dihasilkan berupa teks keterangan golongan suara dari citra partitur tersebut
3. Citra partitur hanya terdiri dari satu garis paranada
4. Citra partitur berupa potongan birama skala keabuan
5. Not-not yang dimasukkan hanya bernilai ¼ ()
6. Simbol musik yang dikenali yaitu: 
7. Citra partitur yang dimasukkan harus sudah jelas memakai kunci F atau kunci G

1.4 Tujuan

Tujuan penyusunan skripsi ini adalah :

1. Mengimplementasikan modifikasi algoritma *Kim's Optical Music Recognition* dalam perangkat lunak yang dapat memperkirakan suara penyanyi yang menyanyikan suatu partitur lagu.
2. Menguji tingkat akurasi perangkat lunak terhadap penggolongan suara pada citra partitur

1.5 Manfaat Penelitian

Dari penelitian ini, diharapkan dapat diciptakan model perangkat lunak yang mampu melakukan perkiraan golongan suara penyanyi yang akan menyanyikan suatu partitur lagu, sehingga dapat mendukung perkembangan musik.

1.6 Metodologi Penelitian

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan skripsi ini adalah:

1. Studi literatur
   Mempelajari teori-teori yang berhubungan dengan partitur musik, penggolongan suara, dan pengolahan citra digital.
2. Pendefinisian dan analisis masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
   Membuat perancangan model perangkat lunak dan mengimplementasikan hasil rancangan tersebut dengan membuat model perangkat lunak penggolongan suara dalam partitur musik.
4. Uji coba dan analisis hasil implementasi
   Menguji perangkat lunak dan menganalisis hasil dari implementasi tersebut apakah telah sesuai dengan tujuan yang dirumuskan sebelumnya.

1.7 Sistematika Penulisan

1. BAB I Pendahuluan
   Memuat latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi pembahasan, dan sistematika pembahasan.
2. BAB II Landasan Teori
   Membahas teori-teori yang mendukung dalam perencanaan dan pembuatan aplikasi
3. BAB III Metodologi
   Berisi tentang metode penelitian dan perencanaan aplikasi
4. BAB IV Hasil dan Pembahasan
   Membahas tentang perencanaan dan pembuatan aplikasi, serta memuat hasil pengujian terhadap aplikasi yang telah dibuat
5. BAB V Kesimpulan dan Saran
   Memuat kesimpulan dan saran

BAB II
TINJAUAN PUSTAKA

2.1 Penggolongan Suara Manusia Dalam Musik

2.1.1 Pengertian Suara Manusia

Bunyi yang dikeluarkan oleh manusia disebut suara. Suara ditimbulkan dalam rongga suara. Rongga suara mempunyai pita-pita suara. Jika pita-pita suara bergetar, timbullah bunyi. Pita suara digetarkan oleh udara dari nafas. Tinggi rendah suara diatur oleh otot di sekeliling pita suara. Warna suara tiap orang berbeda-beda. Perbedaan ini ditimbulkan oleh perbedaan bentuk mulut, bentuk tenggorokan, bentuk hidung, bentuk geligi, dan lain-lain (Redixta, 2007).

2.1.2 Pengertian Musik dan Nada

Musik merupakan bagian dari suara, sebuah seni yang didasarkan pada pengaturan suara dalam suatu waktu. Musik dibedakan dari suara yang lain dengan mengenali empat atribut utama yaitu *pitch*, *dynamics*, *tone color*, and *duration* (Kamien, 1988).

Nada adalah suara yang frekuensinya telah ditetapkan. IMC (*International Music Council*) telah menetapkan nada a1=440 artinya nada a1 itu harus bergetar 440 getaran dalam 1 detik atau dengan kata lain nada a1=440 getaran per detik. Karena yang menemukan getaran suara itu adalah tuan Hertz, maka nada a1=440 Hertz (a1=440 Hz) sedang nada a0(a)=220 Hz dan nada a2=880 Hz. Kalau nada a1 di luar 440 Hz disebut nada sumbang. Jika melebihi 440 Hz disebut *sharp*/tajam, dan kurang dari 440 Hz disebut *flat*. Karena itu diusahakan nada a1 yang dihasilkan instrumental maupun vokal harus bernada a1 *pitch*, yaitu 440 Hz (Simanungkalit, 2008).

Tangga nada adalah tinggi atau rendahnya suara secara relatif dan terutama tergantung pada frekuensi getaran atmosfer yang terbawa oleh bunyi itu. Makin banyak getaran per detiknya, berarti makin besar frekuensinya, makin tinggi tangga nadanya makin sedikit getaran, makin rendah tangga nada. Dalam apa yang disebut tangga nada standar, atau filharmonik, frekuensi dari 440 getaran, atau putaran per detik dimasukkan dalam tangga nada A di atas C tengah.

Perubahan sedikit pada skala tepat menghasilkan suatu skala yang memungkinkan perubahan dari tuts ke tuts. Ini adalah skala tertinggikan, yaitu skala yang kini dipergunakan. Di sini oktaf dibagi ke dalam 12 seminada yang sederajat. Perbandingan antara 1 seminada dengan seminada berikut yang langsung di bawahnya dalam skala adalah sekitar 1,0595. Misalnya, adalah kunci A yang didasarkan pada A – 440 (440 putaran per detik) bila akan mempunyai tabel frekuensi berikut: A-440; G-392,0; F-349,2; E-329,6; D-293,7; C-261,6; B-246,9; A-220. Kalau sebarang frekuensi di sini dikalikan dengan 1,0595, akan diperoleh frekuensi not berikut di bawahnya (Knauss, 2003). Nama nada beserta frekuensinya ditampilkan pada tabel 2.1.

Tabel 2.1 Tabel nada beserta frekuensinya

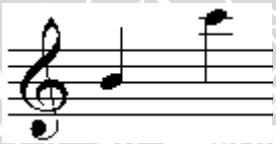
Nada	**Frekuensi (Hz)**
A	440
G	392
F	349,2
E	329,6
D	293,7
C	261,6
B	246,9
A	220

2.1.3 Golongan Suara Manusia

Suara manusia digolongkan berdasarkan pada *range*/ambitus. *Range*/ambitus adalah nada paling rendah sampai nada paling tinggi yang dapat diraih. Golongan suara manusia terdiri dari:

a. Soprano

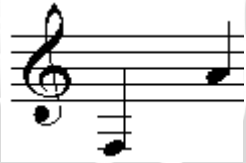
Adalah jenis suara perempuan yang berambitus tertinggi, secara kasar berambitus dari nada b1 sampai nada c3. Suara laki-laki yang mempunyai ambitus sama dengan suara perempuan, dimiliki anak laki-laki pada umur muda di mana suaranya belum berubah. Dalam hal ini bisa juga terdapat pada suara laki-laki dewasa dengan suara duplikasinya yang disebut suara laki-laki *falsetto*, suara yang dihasilkan oleh resonator rongga kepala. *Range* soprano ditampilkan pada gambar 2.1



Gambar 2.1 Kisaran *range* soprano

b. Alto

Dalam partitur paduan suara, alto juga disebut contralto, yaitu ambitus suara perempuan paling rendah (f sampai a1). Biasanya alto ini membawakan suara kedua tertinggi dalam paduan suara. Ada juga yang disebut alto laki-laki (*counter* tenor), yaitu suara laki-laki yang hampir sama ambitusnya dengan alto perempuan. Ambitus tertinggi dari suara laki-laki bisa berproduk dengan menggunakan suara kepala/*head register*. *Range* alto ditampilkan pada gambar 2.2.

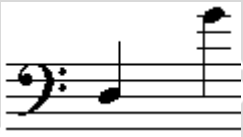


Gambar 2.2 Kisaran *range* alto

c. Tenor

Tenor adalah suara laki-laki yang berambitus paling tinggi. Dalam partitur musik paduan suara (*choral music*) suara tenor tertinggi adalah soprano anak laki-laki (*boy soprano*). Partitur

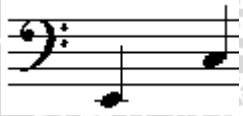
paduan suara untuk tenor biasanya ditulis dalam kunci G seperti yang dipakai sopran. Tetapi dalam pengertian dan kenyataan, suara tenor itu berada satu oktaf di bawah suara sopran. Ambitus tenor adalah d sampai g1, tetapi dengan latihan-latihan khusus dapat lebih ditingkatkan ke ambitus yang lebih besar. *Range* tenor ditampilkan pada gambar 2.3.



Gambar 2.3 Kisaran *range* tenor

d. Bass

Bass adalah suara laki-laki berambitus paling rendah (E besar sampai c). Sifat dan karakter dari bass yang bersuara sangat rendah, besar, dan dalam dapat mengimbangi kewibawaan suara alto. Dalam musik instrumental suara bass ini sering dijadikan dasar atau landasan dari garis-garis harmoni. Hal ini terdapat dalam ciptaan-ciptaan zaman Baroque (1600-1750). Nada-nada dari bass ini disebut juga sebagai kebalikan dari *treble* atau suara nada-nada tinggi (*high pitch voice*), baik dewasa (soprano dan *male* alto/alto laki-laki) maupun anak-anak. Range bass ditampilkan pada gambar 2.4.



Gambar 2.4 Kisaran *range* bass

Penggolongan suara dan *range* ditampilkan pada tabel 2.2 (Simanungkalit, 2008).

Tabel 2.2 Tabel acuan penggolongan suara

Golongan Suara	*Range*
Soprano	b1 – c3
Alto	f – a1
Tenor	d – g1
Bass	E – c

2.2 Partitur Musik

Partitur musik merupakan bentuk dari notasi musik yang ditulis dengan tangan atau dengan menggunakan program komputer. Media penulisan partitur umumnya menggunakan kertas, walaupun beberapa tahun belakangan, akses dalam notasi musik juga melibatkan presentasi dalam layar komputer (Kamien, 1988).

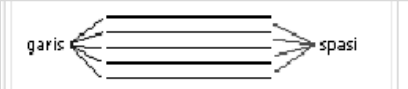
2.2.1 Partitur Vokal

Partitur vokal merupakan pengurangan dari partitur lengkap sebuah karya musik (sebagai contoh opera, *musical*, *oratorio*, *cantata*, dll.) untuk menunjukkan bagian vokal (solo dan paduan suara) dalam garis biramanya dan bagian piano dari orkestra (biasanya dua tangan) di bawah bagian vokal; bagian orkestra yang murni dari partitur juga dikurangi untuk piano. Bila sebagian dari karya musik merupakan *a cappela*, pengurangan piano dari bagian vokal sering ditambahkan untuk membantu dalam latihan (sering terjadi dalam kasus partitur *a cappela* rohani). Walaupun tidak digunakan dalam pertunjukan, partitur vokal merupakan cara yang baik bagi seorang solois dan penyanyi paduan suara untuk mempelajari musik dan berlatih terpisah dari bagian instrumen. Partitur vokal umumnya tidak melibatkan dialog yang diucapkan, kecuali tanda-tanda tertentu (Temperley, 2001).

2.2.2 Paranada

Dalam notasi musik balok, paranada adalah lima garis horisontal tempat not ditulis. Not dapat diletakkan di garis atau di antara garis (spasi) paranada. Garis paranada diberi nomor dari bawah ke atas; garis paling bawah disebut garis kelima dan garis paling atas disebut garis pertama. Not yang terletak di garis atau spasi lebih tinggi berarti memiliki tinggi nada lebih tinggi.

Not pada paranada dibaca dari kiri ke kanan. Not yang terletak di sebelah kiri dimainkan sebelum not di sebelah kanan (Kamien,1988). Contoh paranada ditampilkan pada gambar 2.5.

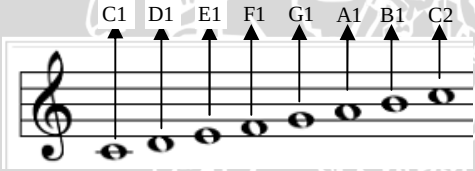


Gambar 2.5 Paranada

2.2.3 Notasi Musik

Notasi musik merupakan cara untuk melambangkan nada. Tujuh dari duabelas *pitch* yang mengisi oktaf dari musik barat dinamakan dari tujuh huruf pertama alfabet: A, B, C, D, E, F, G. Bagian huruf-huruf ini diulang terus menerus untuk mewakili nada yang sama di oktaf yang lebih rendah atau lebih tinggi.Tujuh nada ini berhubungan dengan tuts putih di piano. Nada C yang paling dekat dengan bagian tengah piano disebut C *middle*.

Hanya tujuh nada di dalam oktaf yang diberi nama, yaitu yang diproduksi oleh tuts putih pada piano. Lima nada lainnya, yang dimainkan di tuts hitam, ditandai dengan salah satu dari tujuh huruf yang sama dengan penambahan tanda *sharp* (#) atau tanda flat (b). Sebagai contoh, *pitch* antara C dan D bisa disebut C *sharp* – lebih tinggi dari C biasa atau D *flat*– lebih rendah daripada D. Kejadian yang sama digunakan pada tuts hitam yang lain. Tanda pugar (♮) digunakan untuk membatalkan tanda *sharp* atau *flat* sebelumnya (Kamien, 1988). Contoh not balok dalam partitur diitampilkan pada gambar 2.6.



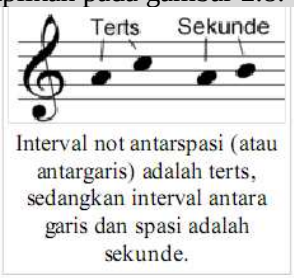
Gambar 2.6 Contoh not musik yang ditulis ke not balok.

2.2.4 Notasi Balok

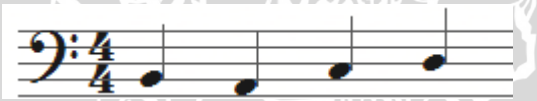
Notasi balok merupakan cara penulisan notasi musik. Dalam notasi balok, sistem paranada bergaris lima digunakan sebagai dasar. Bersama dengan keterangan mengenai tempo, ketukan, dinamika, dan instrumentasi yang digunakan, not ditempatkan pada paranada dan dibaca dari kiri ke kanan.

Durasi nada dilambangkan dengan nilai not yang berbeda-beda, sedangkan tinggi nada dilambangkan dalam posisi not secara vertikal pada paranada. Interval dua not yang dipisahkan satu garis paranada (yaitu berada pada dua spasi yang bersebelahan) seperti digambarkan pada ilustrasi di samping merupakan interval terts, sedangkan interval antara not pada spasi dengan not pada garis adalah interval

sekunde. Tanda kunci pada awal paranada menunjukkan tinggi nada yang diwakili oleh garis dan spasi pada paranada tersebut. Pada gambar di samping, kunci-G digunakan, menandakan bahwagaris kedua dari bawah melambangkan nada g^1. Dengan demikian, interval terts pada gambar di samping adalah pasangan nada a1–c2,sedangkan interval sekunde merupakan pasangan nada a1–b1. Not-not yang melambangkan tinggi nada di luar jangkauan kelima garis paranada dapat digambarkan dengan menggunakan garis bantu yang diletakkan di atas atau di bawah paranada (Kamien, 1988). Contoh citra notasi balok dengan kunci G ditampilkan pada gambar 2.7, dan kunci F ditampilkan pada gambar 2.8.



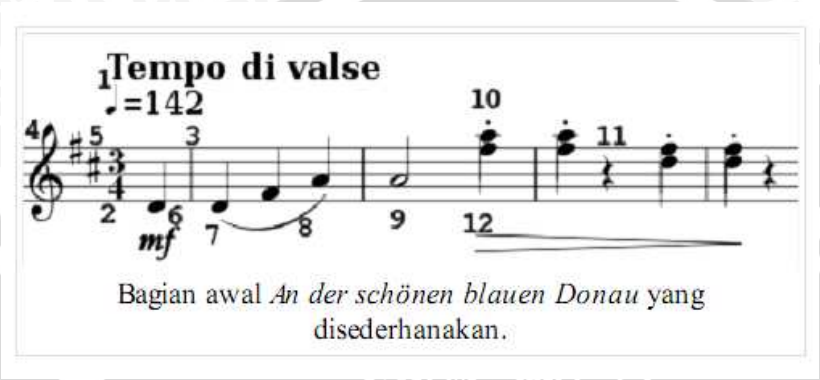
Gambar 2.7 Notasi balok dengan kunci G



Gambar 2.8 Notasi balok dengan kunci F

2.2.5 Penjelasan Partitur Musik

Contoh citra partitur musik secara lengkap ditampilkan pada gambar 2.9.



Gambar 2.9 Contoh Partitur

Penggunaan notasi balok dijelaskan dalam contoh yang diambil dari bagian awal karya Johann Strauss, *An der schönen blauen Donau* yang disederhanakan.

1. Di sebelah kiri atas pada awal lagu biasanya ditempatkan petunjuk tempo (yaitu kecepatan lagu), seringkali dalam bahasa Italia, yang di sini menunjukkan "tempo *waltz*". Selain itu juga terdapat penanda metronom dalam satuan BPM (*beats per minute*), di sini 142 ketukan per menit.
2. Tanda birama menunjukkan ritme lagu. Angka di bagian atas tanda birama menunjukkan jumlah ketukan per birama, sedangkan angka di bawah menunjukkan nilai not perketukan. Tanda birama 3/4 di sini menunjukkan bahwa terdapat tiga ketukan dalam birama, satu ketukan kuat diikuti dua ketukan lemah, dan masing-masing ketukan bernilai not seperempat.
3. Garis birama merupakan pemisah antar birama.
4. Pada bagian awal paranada terdapat kunci-G yang menandakan bahwa garis kedua dari bawah melambangkan nada g^1 (berfrekuensi sekitar 418 Hz).
5. Tanda mula utama yang di sini terdiri dari dua tanda mula kres pada garis nada c dan f menunjukkan bahwa kedua nada tersebut dinaikkan setengah nada dalam semua oktaf (dimainkan

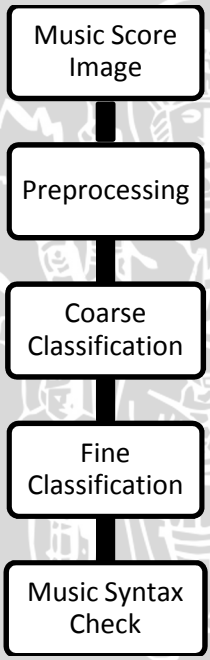
sebagai nada cis dan fis) serta menunjukkan bahwa karya musik bersangkutan bertangga nada D mayor atau B minor.

6. Not pertama adalah not seperempat dengan nada d1, dengan dinamika (nyaring lembutnya suara) mf (bahasa Italia, *mezzo forte*: agak nyaring). Dapat dilihat bahwa not tersebut langsung diikuti garis birama walaupun tiga ketuk dalam birama tersebut belum selesai. Dengan demikian, karya ini dimulai bukan dengan ketukan pertama bertekanan, melainkan dengan ketukan ketiga lemah dalam suatu birama pembuka (*anacrusis*).
7. Not kedua juga merupakan not seperempat dan bernada d1 yang jatuh pada ketukan pertama dalam birama berikutnya.
8. Tanda *legato* menghubungkan not d1 tersebut dengan not fis1 dan a1, menandakan bahwa ketiga not tersebut harus dimainkan secara legato (sambung-menyambung).
9. Pada birama berikutnya terdapat not setengah bernada a1 berdurasi dua ketukan.
10. Berikutnya terdapat not seperempat dengan dua kepala not pada posisi nada fis2 dan a2, menandakan bahwa kedua nada tersebut harus dimainkan bersamaan. Di atas not tersebut terdapat tanda *staccato*, menandakan bahwa not tersebut harus dimainkan secara *staccato* (terpisah nyata dari not sebelum dan sesudahnya).
11. Tanda diam seperempat menandakan bahwa tidak ada nada yang dimainkan selama (dalam hal ini) satu ketukan.
12. Di bawah tiga birama terakhir terdapat tanda *decrescendo*, menandakan bahwa pada ketiga birama tersebut terdapat perubahan dinamika, yaitu dimainkan makin melembut (dapat juga ditulis *decresc.* atau *dim.*, *diminuendo*) (Kamien, 1988).

2.3 Algoritma Kim's Optical Music Recognition

W. J. Kim menerapkan proses pengenalan partitur musik cetak (*recognition system for printed music score*) dengan jumlah data musik yang banyak per *frame*. Sistem yang diajukan didesain untuk mengenali jumlah data musik yang banyak per *frame* tanpa menggunakan pemroses khusus untuk mengenali simbol musik dan juga didesain untuk tidak terpengaruh pencahayaan yang buruk dan ukuran simbol musik.

Sistem yang ditawarkan terdiri dari lima fitur mayor, yang terdiri dari *preprocessing*, *coarse classification*, *fine classification*, dan*music syntax check*. Diagram blok proses KOMR ditampilkan pada gambar 2.10.



Gambar 2.10 Diagram Blok
Algoritma *Kim's Optical Music Recognition*

Modifikasi KOMR dilakukan dalam proses *coarse classification*, *fine classification*, dan *music syntax check*.

2.3.1 Pengertian Citra

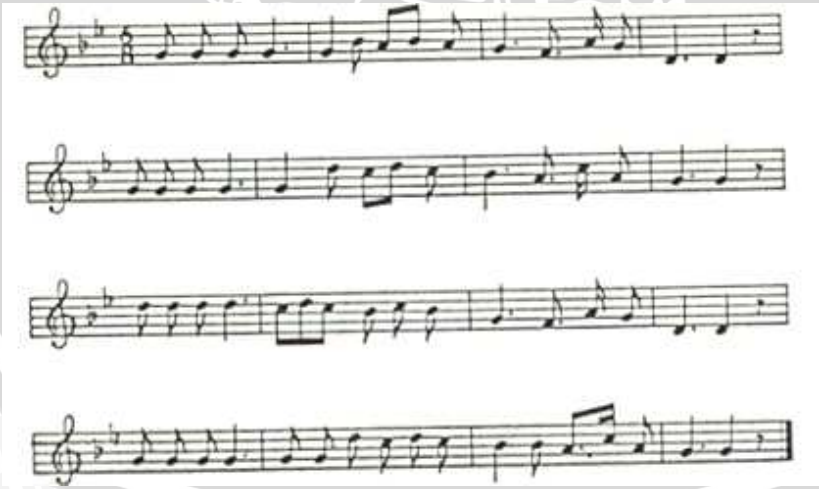
Definisi citra menurut kamus webster adalah "suatu representasi, kemiripan atau imitasi dari suatu obyek atau benda" (Achmad dan Firdausy, 2005).

Citra skala keabuan memberi kemungkinan warna yang lebih banyak daripada citra biner, karena ada nilai-nilai lain di antara nilai minimum (biasanya = 0) dan nilai maksimumnya. Banyaknya kemungkinan nilai dan nilai maksimumnya bergantung pada jumlah bit yang digunakan. Format citra ini disebut skala keabuan karena pada umumnya warna yang dipakai adalah antara hitam sebagai warna minimal dan warna putih sebagai warna maksimalnya, sehingga warna antaranya adalah abu-abu (Achmad dan Firdausy, 2005).

2.3.2 Preprocessing

Preprocessing meliputi penerapan algoritma pendeteksian *stave* dan algoritma *adaptive thresholding*. Algoritma tersebut diterapkan pada citra partitur skala keabuan yang telah ditajamkan. Semua simbol musik diasumsikan berada di dalam partitur.

Algoritma yang digunakan dalam prosedur ini meliputi *image enhancement*, *stave detection*, *stave window thresholding*, dan *stave removal*. Contoh citra partitur ditampilkan pada gambar 2.11



Gambar 2.11 Contoh Partitur

a. Image Enhancement

Karena semua simbol musik berada saling berdekatan dalam citra partitur, terdapat beberapa *blur* di antara beberapa simbol. Dalam proses ini digunakan teknik *simple Laplacian convolution* untuk menyingkirkan *blurring* dari citra asli. 3x3 *window convolution mask* ditambahkan pada citra partitur asli untuk menyingkirkan *blurring*. $H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

a.1 Operator Laplacian

Turunan kedua dari tepi berjenis landai adalah sebuah fungsi yang memotong sumbu x pada lokasi tepi. Laplacian adalah kesamaan dua dimensi dari turunan kedua untuk fungsi tersebut. Rumus Laplacian untuk fungsi f(x,y) seperti pada rumus 2.1.

$$\nabla^2 \mathrm{f} = \frac{\partial^2 \mathrm{f}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{f}}{\partial \mathrm{y}^2} \quad (2.1)$$

Turunan kedua sepanjang arah x dan y yang diperkirakan dengan persamaan diferensial akan menghasilkan persamaan 2.2.

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial G_x}{\partial x}$$

$$= \frac{\partial \big(f(x+1,y) - f(x,y)\big)}{\partial x}$$

$$= \frac{\partial f(x+1,y)}{\partial x} - \frac{\partial f(x,y)}{\partial x}$$

$$= \big(f(x+2,y) - f(x+1,y)\big) - \big(f(x+1,y) - f(x,y)\big)$$

$$= f(x+2,y) - 2f(x+1,y) + f(x,y) \quad (2.2)$$

Namun perkiraan ini menyebabkan titik pusat bergeser pada (x+1,y). Oleh karena itu, dengan mengganti x dengan x-1 akan didapatkan persamaan 2.3.

$$\frac{\partial^2 f}{\partial x} = f(x+1,y) - 2f(x,y) + f(x-1,y) \quad (2.3)$$

Ini berarti pendekatan terhadap turunan kedua parsial sekarang telah berpusat pada (x,y) seperti yang diharapkan. Dengan cara yang sama, untuk turunan terhadap sumbu y, didapatkan persamaan 2.4.

$$\frac{\partial^2 f}{\partial y} = f(x, y+1) - 2f(x, y) + f(x, y-1) \quad (2.4)$$

Operator Laplacian memberitahukan keberadaan dari suatu tepi ketika keluaran dari operator membuat perpotongan dengan sumbu *x*. Namun bila suatu daerah dalam citra mempunyai nilai nol yang seragam, diabaikan dan tidak dianggap sebagai tepi. Secara prinsip, lokasi titik perpotongan dapat diduga sampai resolusi sub-piksel menggunakan interpolasi linear, tetapi hasilnya mungkin tidak akurat akibat pengaruh *noise* (Ahmad, 2005).

b. Stave Detection

Stave (garis paranada) merupakan lima garis horisontal berturutan. Untuk memusatkan perhatian kepada simbol musik di dalam *stave*, maka harus dibentuk *stave window*. Untuk melakukan ini, algoritma *stave detection* berdasarkan teknik *image segmentation*, diajukan untuk citra partitur skala keabuan. Prosedur algoritma *stave detection* sebagai berikut:

1. Dapatkan skala keabuan rata-rata untuk tiap garis horisontal.
2. Dapatkan histogram skala keabuan rata-rata untuk tiap garis horisontal.
3. Gunakan *threshold* level untuk memaksimalkan nilai yang diharapkan di antara varian kelas, tentukan garis horisontal yang disebut kandidat *stave*.
4. Dapatkan informasi *stave* seperti lebar *stave*, *stave window* dengan memilih lima garis horisontal berturutan.

b.1 Histogram Tingkat Keabuan

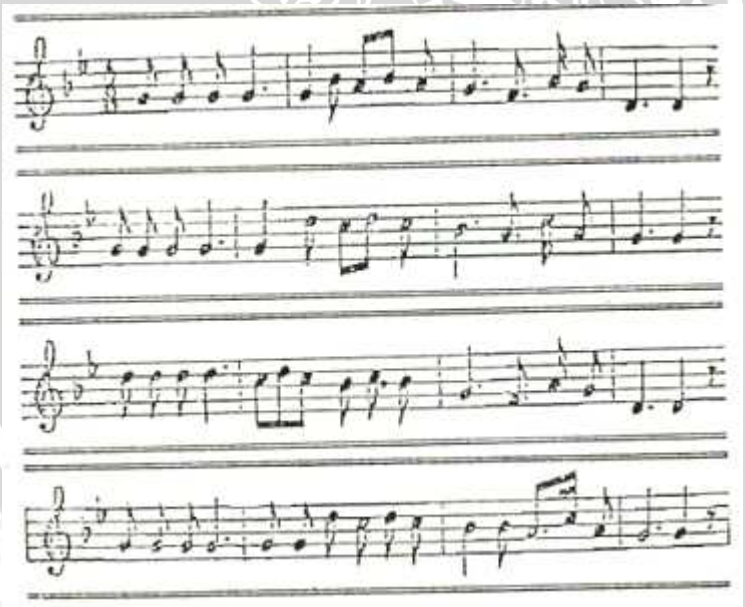
Salah satu alat bantu yang paling sederhana dan sangat berguna dalam pengolahan citra digital adalah histogram tingkat keabuan (*gray-level histogram*). Informasi suatu citra seringkali dapat diwakili oleh histogram ini. Komputasi histogram sangat sederhana dan cepat, serta dapat dilakukan pada saat suatu citra dipindahkan ke tempat lain (misalnya saat dibaca dari file).

Histogram tingkat keabuan adalah suatu fungsi yang menunjukkan jumlah titik yang ada di dalam suatu citra untuk setiap tingkat keabuan. Absisnya adalah tingkat keabuan, dan ordinatnya adalah frekuensi kemunculan atas banyaknya titik dengan nilai keabuan tertentu.

c. Stave Window Thresholding

Ada banyak cara untuk memperoleh citra *binary* dari citra skala keabuan untuk meningkatkan kecepatan komputasi. Bagaimanapun, banyak simbol musik kecil tersebar sangat dekat satu sama lain dan intensitas pencahayaan pada partitur tidak seragam, banyak informasi citra yang penting dapat hilang bila menggunakan metode *auto-thresholding* yang hanya memenuhi satu kriteria. Prosedur dari metode ini, disebut algoritma *adaptive thresholding* sebagai berikut:

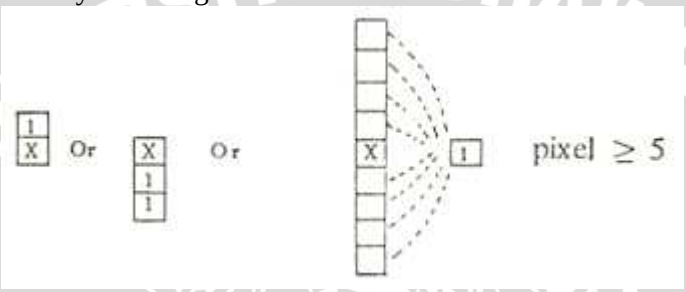
1. Tentukan *thresholding level* yang memaksimalkan nilai yang diharapkan di antara varian kelas. Pada langkah ini, nilai total rata-rata dan lembah integral dari histogram juga digunakan untuk meminimalkan informasi yang hilang.
2. Dengan menggunakan *thresholding level* yang dimodifikasi dari langkah 1, dapatkan citra biner dengan membagi *stave window* ke dalam beberapa bagian secara horisontal.
   Contoh citra hasil binerisasi dan pembentukan stave window ditampilkan pada gambar 2.12



Gambar 2.12 Citra biner dan *stave window*

d. Stave Removal

Bila semua informasi *stave* telah diperoleh dari prosedur sebelumnya, maka perlu dilakukan penghilangan *stave* dari citra biner untuk mengenali simbol musik lainnya secara lebih akurat. Penghilangan *stave* dapat dilakukan dengan memeriksa kesinambungan panjang garis vertikal (*vertical run length connectivity*). Sebagai contoh, sebuah titik pada garis horisontal dari *stave* dapat tidak dihapus bila panjang garis vertikal dari *stave* memenuhi syarat sebagai berikut:

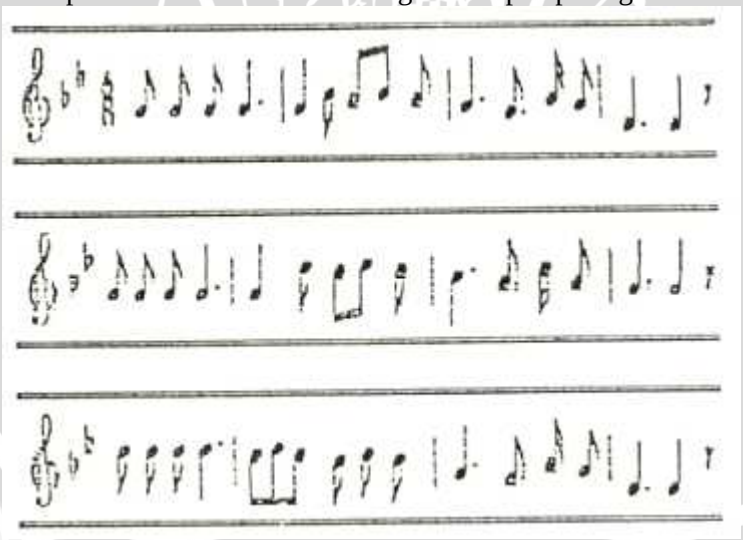


Gambar 2.13 Syarat *vertical run length connectivity*

Di mana 1 : piksel hitam

X : piksel yang sedang diperiksa pada kandidat *stave*

Citra partitur setelah *stave* dihilangkan tampak pada gambar 2.14.



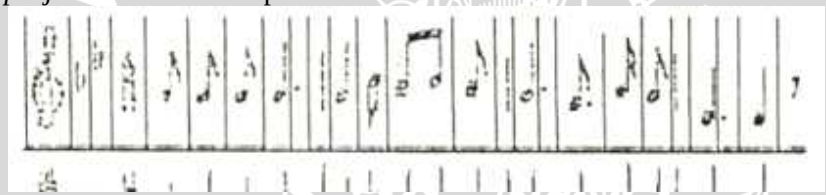
Gambar 2.14 Citra partitur dengan *stave* yang hilang

2.3.3 Coarse Classification

Sebagai hasil dari *preprocessing*, semua informasi dari *stave* telah diketahui dan diperoleh citra biner dengan *stave* yang telah dihilangkan. Langkah berikutnya adalah memisahkan setiap simbol musik ke dalam citra biner untuk mengenalinya satu per satu. Untuk memperpendek waktu pengenalan, simbol musik secara kasar digolongkan menjadi sembilan kelompok.

a. Horizontal Projection

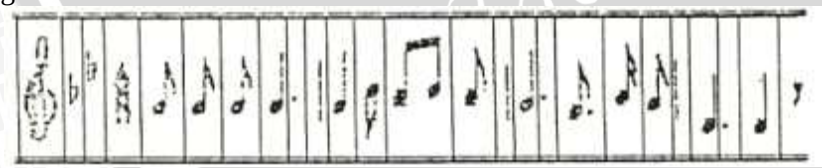
Metode *integral projection* biasanya digunakan untuk menganalisa foto atau mengenali partitur cetak. Untuk melakukan *coarse classification* pada simbol musik, sistem yang diajukan mengadaptasi teknik *horizontal projection* dari metode *integral projection*. Gambar 2.15 menunjukkan hasil dari *horizontal projection* dari beberapa simbol musik.



Gambar 2.15 *Horizontal Projection*

b. Symbol Windowing

Untuk mencapai ketepatan pengenalan yang tinggi dan mengurangi waktu pengenalan, Kim memperkenalkan sebuah *symbol window*, yang hanya mengandung satu simbol musik. Untuk mencapainya, nilai dari *horizontal projection* dibandingkan dengan fungsi *unit pulse*. Bila nilai suatu titik kurang dari nilai *threshold* tertentu setelah dibandingkan, titik-titik ini diasumsikan menjadi tepi vertikal dari *symbol window*. Contoh hasil citra ditampilkan pada gambar 2.16.



Gambar 2.16 Contoh *symbol window*

b.1 Connected Component Labelling

Connected components labelling (CCL) memindai sebuah citra dan mengelompokkan piksel-pikselnya ke dalam sebuah komponen berdasarkan *pixel connectivity*, sebagai contoh semua piksel di dalam komponen yang terhubung mempunyai nilai intensitas piksel yang mirip dan dalam suatu cara terhubung antara satu dengan lainnya. Ketika semua kelompok telah ditentukan, setiap piksel diberi label dengan sebuah keabuan (*graylevel*) atau warna (*color labeling*) berdasarkan komponen yang menjadi bagiannya (Gonzales, 1992).

Pseudocode dari CCL ditampilkan pada gambar 2.17.

```
algorithm TwoPass(data)
   linked = []
   labels = structure with dimensions of data, initialized with
            the value of Background

First pass

for row in data:
for column in row:
if data[row][col] is not Background
               neighbors = connected elements with the current
               element's label

if neighbors is empty
                  linked[NextLabel] = set containing NextLabel
                  labels[row][column] = NextLabel
                  NextLabel += 1

else

Find the smallest label

                  L = neighbors labels
                  labels[row][column] = min(L)
for label in L
                     linked[label] = union(linked[label], L)

Second pass

for row in data
for column in row
if labels[row][column] is not Background
               labels[row][column] =
min(linked[labels[row][column]])

return labels
```

Gambar 2.17 *Pseudocode* CCL

2.3.4 Fine Classification

Pada tahap ini, dilakukan pengelompokan terhadap simbol kunci G (𝄞), kunci F (𝄢), dan not (𝅗𝅥 atau 𝅘𝅥). Setelah didapatkan informasi kunci yang digunakandalam partitur, maka dilakukan tahap selanjutnya yaitu:

a. Penggolongan kunci G atau F

Citra partitur dapat dikelompokkan menjadi dua golongan, yaitu suara pria atau wanita. Bila citra partitur menggunakan kunci G (𝄞), maka digolongkan ke dalam suara wanita. Bila citra partitur menggunakan kunci F (𝄢), maka digolongkan ke dalam suara pria.

b. Penentuan *center notehead*

Center dari *notehead* didapatkan dengan menghitung rata-rata dari koordinat piksel. Koordinat piksel diukur dari *symbol window* pada citra partitur. Gambar 2.18 menunjukkan acuan koordinat citra. Rumus yang digunakan yaitu:

$$(\overline{x_c}, \overline{y_c}) = \frac{\left(\sum_{n=1}^{j} n\right),\left(\sum_{m=1}^{i} m\right)}{i * j},\ n = \left(\sum_{x=1}^{i} x\right),\ m = \left(\sum_{y=1}^{j} y_j\right) \quad (2.9)$$

di mana i * j merupakan ukuran dari *symbol window*.

Gambar 2.18 Pengukuran *center notehead*

c. Pengukuran jarak not

Setelah didapatkan posisi *center* masing-masing *notehead*, maka dilakukan pengukuran jarak koordinat piksel masing-masing not (x_c,y_c) terhadap koordinat piksel garis paranada terbawah (x_p,y_p). Rumus yang digunakan yaitu:

$$r = (y_c) - (y_p) \quad (2.10)$$

d. Penghitungan rata-rata not

Setelah didapatkan informasi jarak masing-masing not, dilakukan penghitungan rata-rata jarak. Rumus yang digunakan yaitu:

$$\bar{r} = \frac{\sum_{i=0}^{n}(r_i)}{n} \quad (2.11)$$

di mana n merupakan jumlah not yang diukur.

2.3.5 Music Syntax Check

Setelah didapatkan rata-rata not, hasil yang diperoleh dibandingkan dengan nilai *threshold* untuk masing-masing golongan suara. (Kim, 1987).

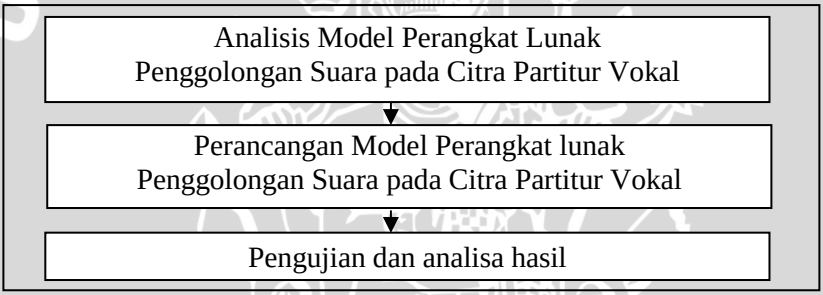
BAB III
METODE PENELITIAN

Pada bab ini akan dibahas langkah-langkah pembuatan perangkat lunak penggolongan suara manusia pada citra partitur vokal serta metode-metode yang digunakan di dalamnya.

Tahap perancangannya adalah sebagai berikut:

1. Mempelajari metode yang digunakan dari literatur dan jurnal yang membahas masalah yang sama.
2. Merancang model perangkat lunak dengan menggunakan modifikasi pada penelitian sebelumnya.
3. Uji coba dan analisa model perangkat lunak pada citra partitur.

Langkah-langkah perancangan perangkat lunak ditunjukkan pada Gambar 3.1



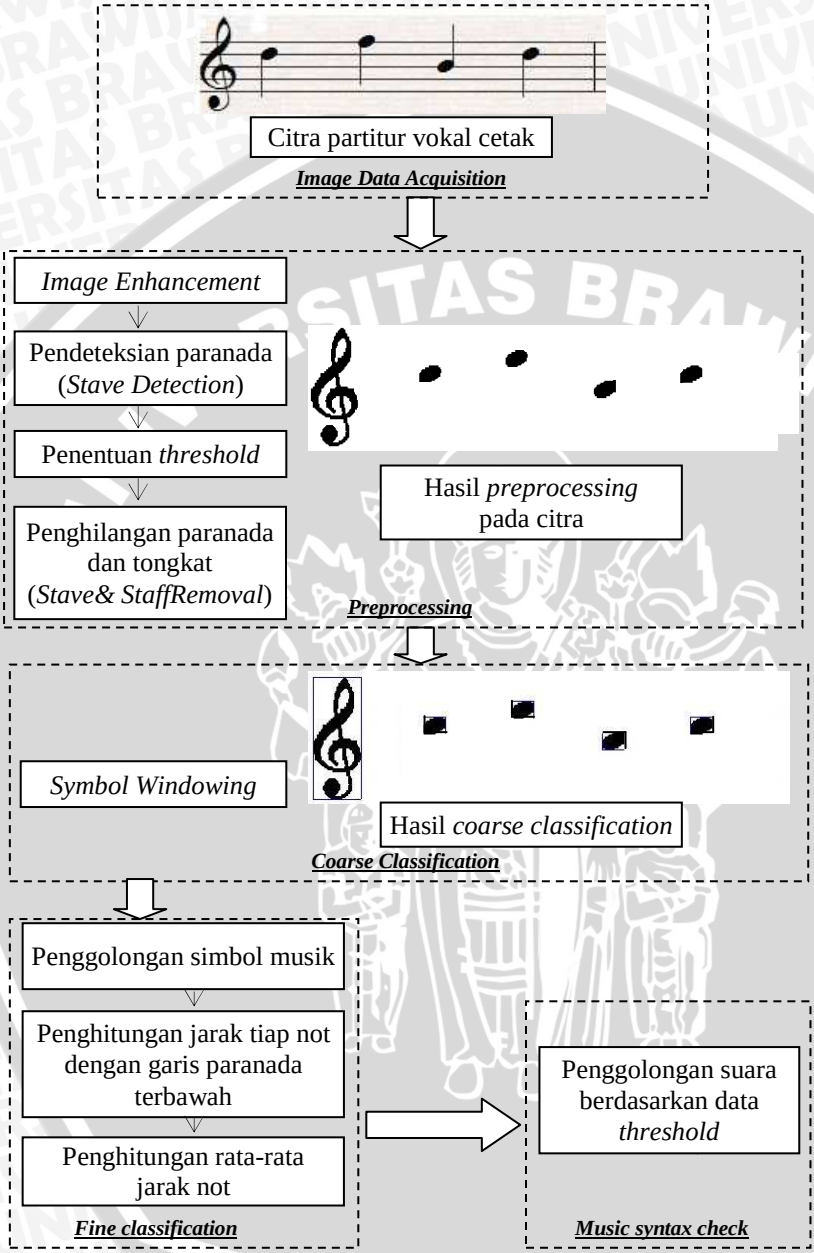
Gambar 3.1 Tahapan penelitian

3.1 Analisis Model Perangkat Lunak

Pada subbab ini akan dibahas berbagai hal yang diperlukan dalam proses *optical music recognition.*

3.1.1 Deskripsi Umum Perangkat Lunak

Perangkat lunak dirancang untuk dapat melakukan proses pengenalan secara otomatis terhadap masukan citra partitur musik cetak dengan jumlah data musik yang banyak per *frame*. Berdasarkan 6 tahap proses pengenalan partitur musik pada algoritma KOMR seperti yang dijelaskan pada bab 1.1, maka sistem yang akan dibangun dimodifikasi untuk memperkirakan penggolongan suara pada citra partitur. Arsitektur sistem dapat dilihat pada gambar 3.2.



Gambar 3.2 Proses umum sistem

3.2 Perancangan Model Perangkat Lunak

3.2.1 Preprocessing

Pada tahap *preprocessing*, di dalamnya meliputi 5 tahap, yaitu: *Image Enhancement*, Pendeteksian Paranada (*Stave Detection*), Penghilangan Paranada (*Stave Removal*), Penghilangan Tongkat (*Staff Removal*), dan Penentuan *Threshold*.

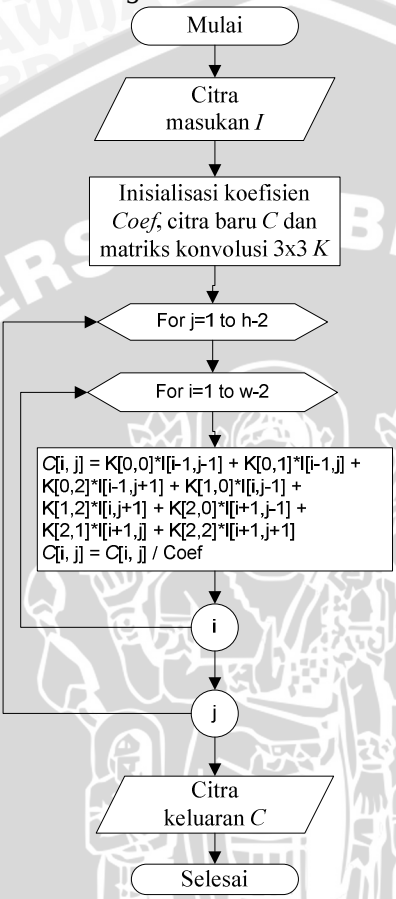
a. Image Enhancement

Image enhancement dilakukan dengan menerapkan *Laplacian Convolution* pada citra partitur vokal. Piksel awal yang ditemukan pada tepian citra tidak diproses. Pemrosesan piksel dimulai dari piksel sebelum tepi dari citra. Setiap piksel diambil, kemudian nilai dari 8 piksel di sekitarnya dikalikan dengan matriks 3x3 yang telah ditentukan. Nilai piksel ini kemudian ditukar dengan nilai yang baru didapat dari hasil perkalian pada citra baru hasil *enhancement*.

Window convolution yang digunakan yaitu $H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Proses dilakukan dengan menggunakan metode konvolusi *Simple Laplacian*. Langkah yang pertama adalah melakukan inisialisasi dengan membentuk matriks konvolusi *K* yang berdimensi 3x3. Dari matriks konvolusi ini kemudian dicari nilai koefisien *Coef* dengan cara menjumlahkan nilai dari semua elemen matriks konvolusi. Koefisien ini nanti akan digunakan sebagai nilai pembagi pada proses selanjutnya. Kemudian buat citra baru *C* dengan dimensi yang sama dengan citra masukan *I*. Citra baru C akan menjadi keluaran dari proses ini. Setelah inisialisasi selesai dilakukan, langkah selanjutnya adalah melakukan konvolusi terhadap setiap piksel citra masukan *I* dengan perkecualian bagian tepi / pinggir dari citra masukan diabaikan. Untuk setiap piksel *I*[i,j] pada citra masukan, buat sebuah *sub window* dengan dimensi yang sama dengan matriks konvolusi dan jadikan piksel tersebut sebagai pusat dari *sub window*. Kalikan nilai setiap elemen pada matriks konvolusi dengan nilai setiap piksel pada *sub window* yang bersesuaian. Jumlahkan hasil dari seluruh perkalian tersebut kemudian bagi dengan koefisien *Coef* yang telah didapat sebelumnya. Lakukan normalisasi nilai akhir yaitu bila nilai > 255 maka nilai = 255 dan bila nilai < 0 maka nilai = 0.

Substitusikan nilai akhir ini ke piksel *C*[i,j] yang baru. Gambar 3.3 menunjukkan diagram alir *image enhancement*.



Gambar 3.3 Diagram alir *Image Enhancement*

b. Stave Detection

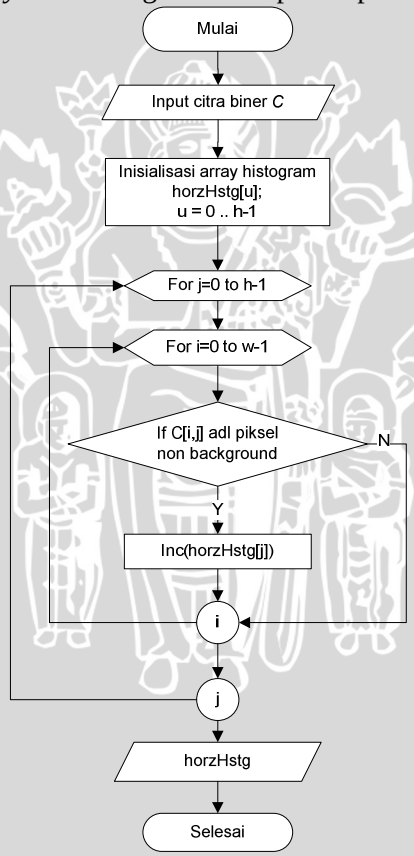
Stave detection yang dilakukan berdasarkan KOMR. Langkah-langkah yang diperlukan dalam *stave detection* sebagai berikut:

- Dapatkan skala keabuan rata-rata untuk tiap garis horisontal dengan melakukan *scanning* secara horisontal dari piksel citra paling tepi.
- Dapatkan histogram skala keabuan rata-rata untuk tiap garis horisontal.

Langkah-langkahnya adalah sbb:

1. Buat array histogram *horzHstg* dengan panjang *h*-1, yang menunjukkan histogram proyeksi skala keabuan secara horizontal dari citra masukan. Setiap elemen array diberi nilai 0.
2. Lakukan iterasi piksel terhadap citra masukan secara vertikal atau baris per baris. Untuk setiap barisnya lakukan iterasi piksel secara horizontal.
3. Setiap kali menemukan piksel yang bukan *background* tambahkan nilai *horzHstg* sehingga *horzHstg[j] = horzHstg[j] +* 1. Bila iterasi secara horizontal sudah selesai maka iterasi berpindah ke baris selanjutnya.

   Diagram alir *grayscale* histogram ditampilkan pada gambar 3.4



Gambar 3.4 Diagram alir *Grayscale Histogram*

- Dengan menggunakan *threshold level* untuk memaksimalkan nilai yang diharapkan antara varian kelas, ditentukan garis horizontal yang disebut kandidat *stave*.

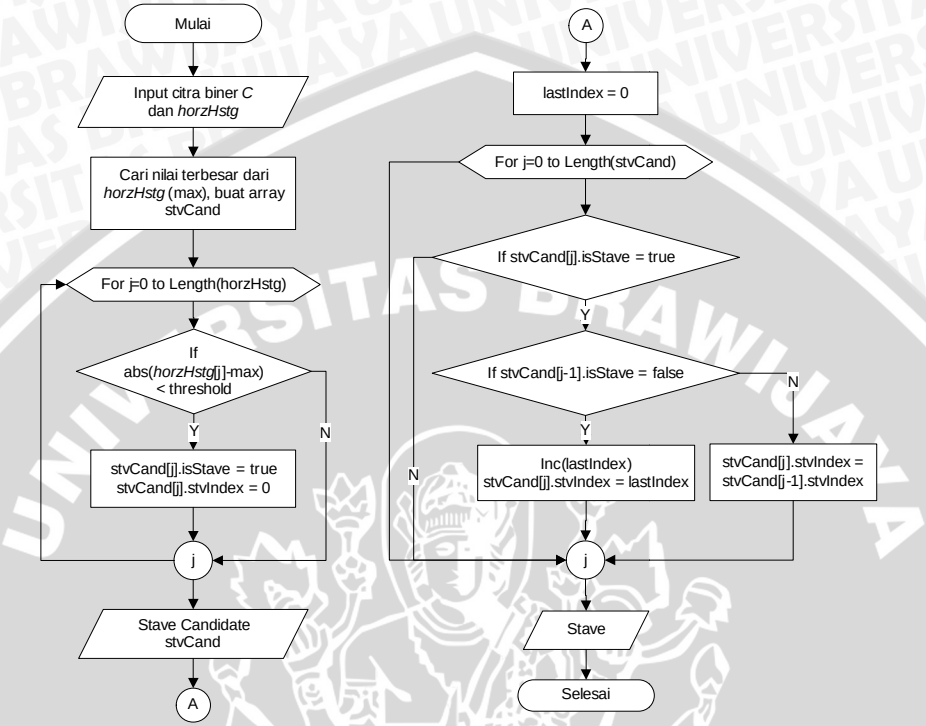
Kandidat *stave* memiliki karakteristik sebagai berikut:

1). Kandidat-kandidat *stave* memiliki skala keabuan yang besar pada Histogram Tingkat Keabuan (*grayscale histogram*). Selain itu besar skala keabuannya tidak terlalu berbeda antara yang satu dengan lain (*threshold*).
2). Kandidat-kandidat *stave* yang saling berdekatan / berhimpitan (jarak kandidat *stave* yang satu dengan yang lainnya adalah 1) dianggap sebagai satu *stave*.

Berdasar pada asumsi diatas maka langkah – langkah *stave detection* dapat dirinci sebagai berikut:

1). Terlebih dahulu dicari skala keabuan yang paling besar *max* dari array histogram *horzHstg*.
2). Dibuat array *stave candidate stvCand* dengan panjang yang sama dengan array *horzHstg*. Array ini berguna untuk menandai apakah suatu baris pada indeks tertentu merupakan kandidat *stave* atau bukan.
3). Cari satu persatu selisih antara setiap skala keabuan *horzHstg[j]* dengan nilai *max* yang didapat. Bila selisihnya kurang dari nilai *threshold* yang telah ditentukan sebelumnya maka baris ke-j dianggap sebagai kandidat *stave* (*stvCand[j].isStave* diubah menjadi *true*).
4). Dari informasi stvCand ini, kelompokkan kandidat *stave* menjadi satu *stave*, dengan syarat kandidat *stave* tersebut memiliki *stvCand[j].isStave* yang nilainya true dan indeks antar kandidat *stave* berdekatan.

Diagram alir dari stave detection ditampilkan pada gambar 3.5



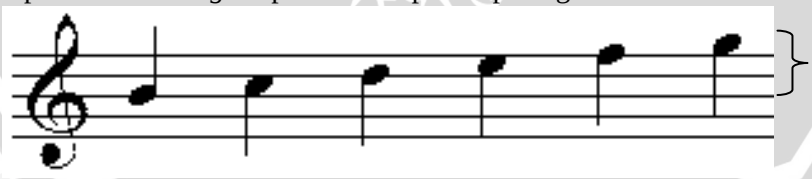
Gambar 3.5 Diagram alir *stave detection*

c. Penentuan threshold suara

Berdasarkan tinjauan pustaka, maka dapat ditentukan *threshold* masing-masing golongan suara, yaitu:

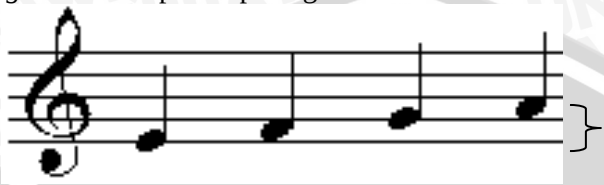
c.1. Menggunakan kunci G (𝄞)

c.1.1. Soprano, yaitu piksel antara garis paranada ketiga dan keempat sampai ke atas. *Range* soprano ditampilkan pada gambar 3.6.



Gambar 3.6 *Range* soprano

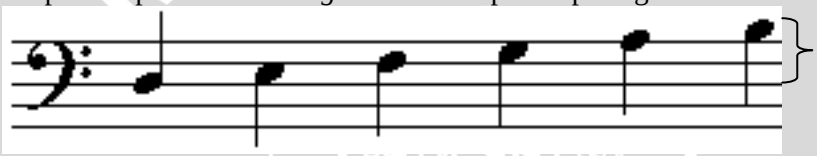
c.1.2. Alto, yaitu piksel antara garis paranada terbawah sampai garis ketiga. *Range* alto ditampilkan pada gambar 3.7.



Gambar 3.7 *Range* alto

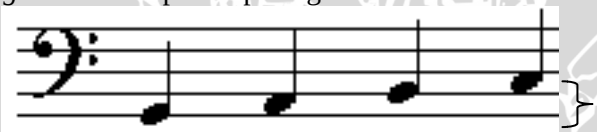
c.2. Menggunakan kunci F (𝄢)

c.2.1. Tenor, yaitu piksel antara antara garis paranada ketiga dan keempat sampai ke atas. *Range* tenor ditampilkan pada gambar 3.8.



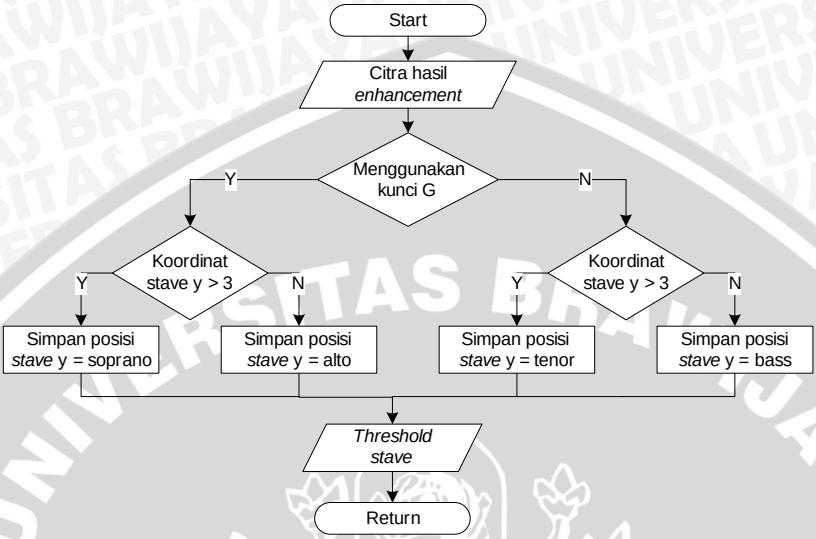
Gambar 3.8 *Range* tenor

c.2.2. Bass, yaitu piksel antara garis paranada terbawah sampai garis ketiga. *Range* bass ditampilkan pada gambar 3.9.



Gambar 3.9 *Range* bass

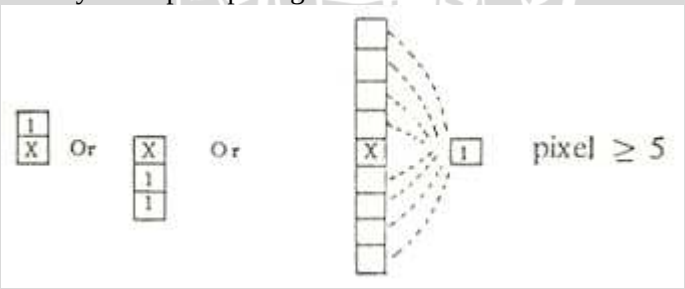
Dari data tersebut, maka dapat ditentukan koordinat *y* yang menjadi *threshold* dalam citra partitur. Diagram alir penentuan *threshold* suara ditampilkan pada gambar 3.10.



Gambar 3.10 Diagram alir penentuan *threshold* suara

d. Stave Removal

Setelah posisi *stave* diketahui, maka *stave* perlu dihilangkan supaya pengenalan simbol-simbol musik menjadi lebih mudah. Bila tidak dihilangkan, stave ini akan menjadi *noise* dalam sebuah citra partitur. Penghilangan *stave* dapat dilakukan dengan memeriksa *vertical length connectivity*. Sebuah titik pada garis horisontal dari *stave* dapat tidak dihapus bila panjang garis vertikal dari *stave* memenuhi syarat seperti pada gambar 3.11.

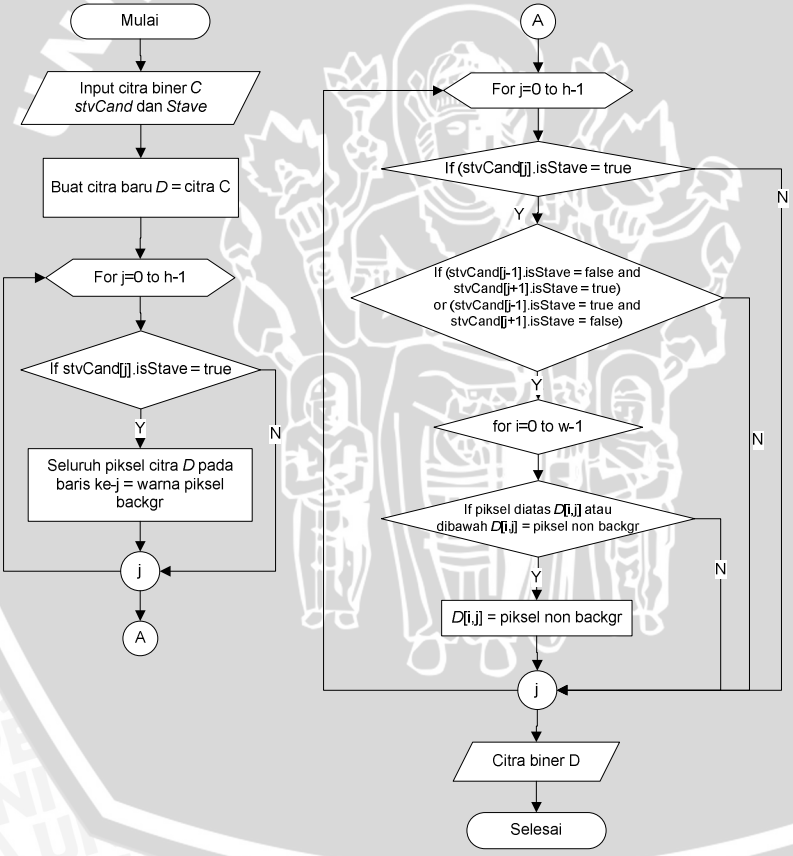


Gambar 3.11 Syarat *vertical run length connectivity*

Di mana 1 :piksel hitam

X :piksel yang sedang diperiksa pada kandidat *stave*

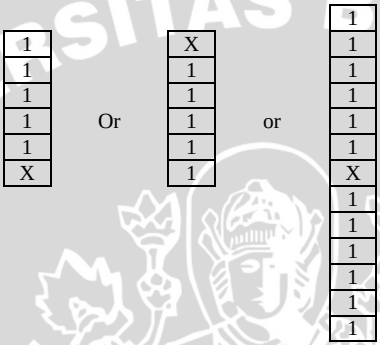
Dari informasi *stave* dan stvCand, dilakukan *stave removal* dengan menggantikan seluruh piksel pada baris ke–j yang nilai stvCand.isStave-nya *true*. Proses ini kemudian dilanjutkan dengan melakukan perbaikan pada citra karena proses *stave removal* menghapus *stave* tanpa memperhatikan bagian dimana *stave* saling berpotongan dengan not musik. Hal ini akan membuat sebuah not musik yang berpotongan dengan *stave* seperti terpisah-pisah. Untuk memperbaikinya, cek sekali lagi baris yang sebelumnya diproses. Cek piksel per piksel, diperiksa apakah piksel di atas atau di bawahnya merupakan piksel hitam atau bukan. Bila piksel hitam maka ubah piksel yang sedang diproses tersebut menjadi piksel hitam. Diagram alir *stave removal* ditampilkan pada gambar 3.12.



Gambar 3.12 Diagram alir *stave removal*

e. Staff Removal

Pada tahap ini dilakukan penghilangan terhadap garis vertikal selain simbol-simbol musik yang dikenali, atau disebut tongkat (*staff*). Dengan menggunakan *vertical run length connectivity*, ditentukan *threshold* supaya piksel-piksel yang tersambung membentuk *staff* dapat dihilangkan. Syarat vertical run length connectivity ditampilkan pada gambar 3.13

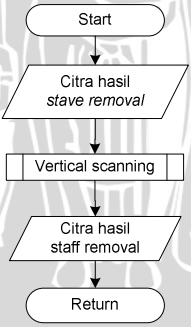


Gambar 3.13 Syarat *vertical length connectivity*

Di mana 1 : piksel hitam

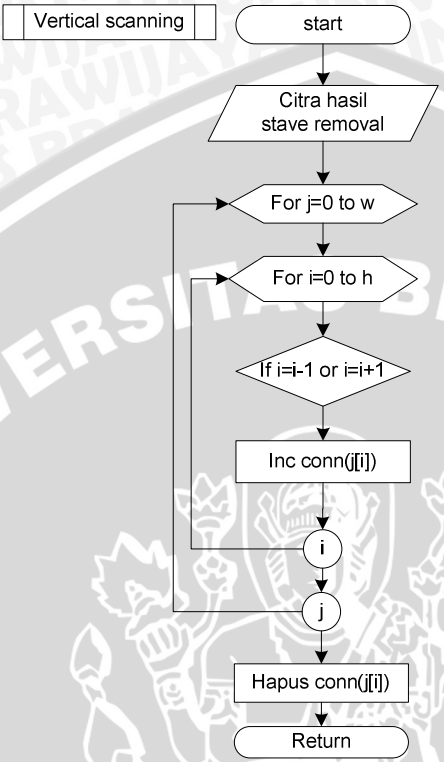
X : piksel yang sedang diperiksa pada kandidat *staff*

Garis vertikal yang memenuhi syarat tersebut di atas dihapus. Diagram alir staff removal ditampilkan pada gambar 3.14.



Gambar 3.14 Diagram alir *staff removal*

Penjelasan diagram alir dari *predefined proccess vertical scanning* ditampilkan pada gambar 3.15.



Gambar 3.15 Diagram alir proses *vertical scanning*

3.2.2 Coarse Classification

Pada tahap ini bertujuan untuk mengelompokkan simbol-simbol musik dalam citra partitur.

a. Symbol Windowing

Dengan menggunakan algoritma *Connected Component Labelling*, maka tiap simbol dalam citra partitur dimasukkan ke dalam *window*.

Connectivity check dilakukan dengan memeriksa label piksel yang berada di sebelah utara dan barat dari piksel yang sedang diperiksa. Kondisi berikut ini untuk menentukan nilai dari label untuk diterapkan pada piksel yang sedang diperiksa.

Kondisi yang diperiksa:

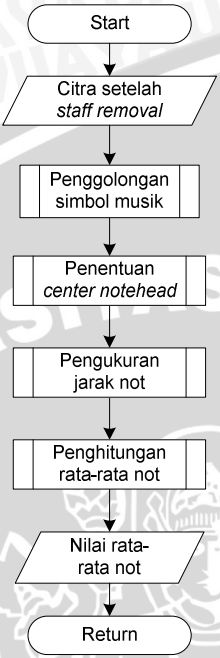
1. Apakah piksel di sebelah kiri (barat) memiliki nilai yang sama?
    1. **Ya**–Kami berada di dalam area yang sama. Tambahkan label yang sama pada piksel
    2. **Tidak**–Cek kondisi berikutnya
2. Apakah piksel di sebelah utara dan barat dari piksel yang diperiksa mempunya nilai yang sama namun label yang berbeda?
    1. **Ya**–Kami tahu bahwa piksel utara dan barat berada pada area yang sama dan harus disatukan. Tambahkan label piksel barat dan utara pada piksel yang diperiksa, dan simpan nilai kesamaannya.
    2. **Tidak**–Cek kondisi berikutnya
3. Apakah piksel sebelah kiri memiliki nilai berbeda dan yang di utara mempunyai nilai sama?
    1. **Ya**–Tambahkan label piksel utara pada piksel yang diperiksa
    2. **Tidak**–Cek kondisi berikutnya
4. Apakah piksel di sebelah barat dan utara memiliki nilai piksel yang berbeda?
    1. **Ya**–Ciptakan label baru dan tambahkan pada piksel yang diperiksa

Algoritma ini bekerja terus, dan menciptakan label baru bila perlu.

3.2.3. Fine Classification

Pada tahap ini, dilakukan pengelompokan terhadap simbol musik berupa kunci G (𝄞), kunci F (𝄢), dan not (𝅗𝅥 atau 𝅘𝅥), penentuan center notehead, pengukuran jarak not untuk menentukan *range*/ambitus, serta penghitungan rata-rata not.

Diagram alir dari fine classification ditampilkan pada gambar 3.16.



Gambar 3.16 Diagram alir *fine classification*

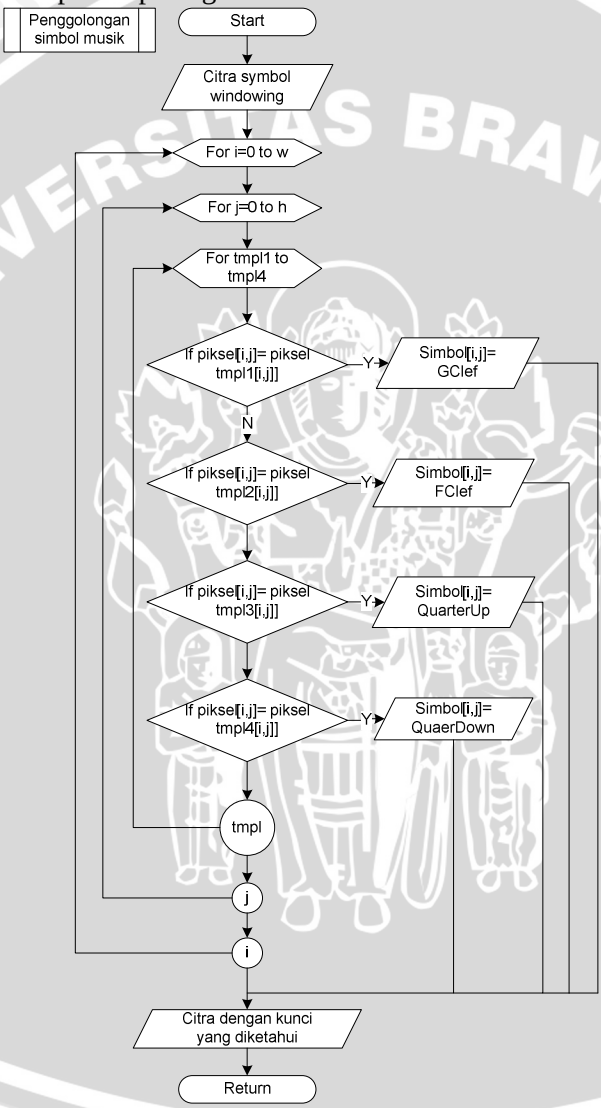
a. Penggolongan simbol musik

Setelah didapatkan informasi kunci yang digunakan dalam partitur, maka citra partitur dapat dikelompokkan menjadi dua golongan, yaitu suara pria atau wanita. Bila citra partitur menggunakan kunci G, maka digolongkan ke dalam suara wanita. Bila citra partitur menggunakan kunci F, maka digolongkan ke dalam suara pria. Selanjutnya dilakukan penggolongan terhadap not musik, yaitu yang berbentuk 𝅗𝅥 atau 𝅘𝅥. Proses pengenalan dilakukan dengan pencocokan tiap simbol musik dengan *template* yang telah disediakan. Langkah-langkahnya:

1). Ambil *window* citra simbol musik dari citra asli.
2). Resize *window* dari simbol musik dengan ukuran yang sama dengan *window template* yang tersedia.
3). Bandingkan *template* dengan simbol secara perpiksel.
4). Cari *matching score* simbol dengan template.

5). Bagi dengan piksel hitam masing-masing *template*, kemudian kalikan dengan 100.
6). Cari nilai paling maksimum.

Diagram alir *predefined proccess* dari penggolongan simbol musik ditampilkan pada gambar 3.17.

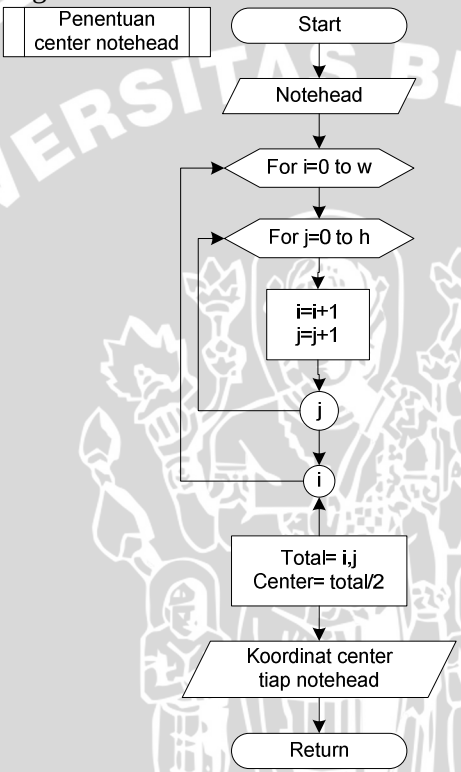


Gambar 3.17 Diagram alir proses penggolongan simbol musik

b. Penentuan center notehead

Center dari *notehead* didapatkan dengan menghitung rata-rata dari koordinat piksel. Koordinat piksel diukur dari *symbol window* pada citra partitur. Rumus yang digunakan yaitu persamaan (2.9).

Diagram alir *predefined process* dari penentuan *center notehead* ditampilkan pada gambar 3.18.

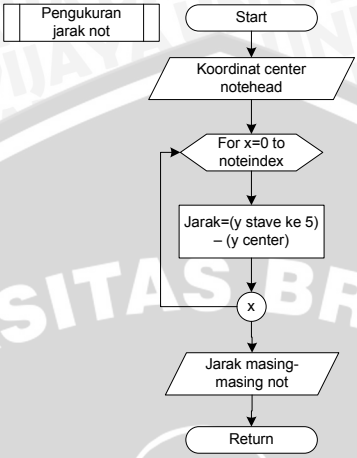


Gambar 3.18 Diagram alir penentuan *center notehead*

c. Pengukuran jarak not

Setelah didapatkan posisi *center* masing-masing *notehead*, maka dilakukan pengukuran jarak koordinat piksel masing-masing not (x_c,y_c) terhadap koordinat piksel garis paranada terbawah (x_p,y_p). Rumus yang digunakan yaitu persamaan (2.10).

Diagram alir *predefined process* dari pengukuran jarak not ditampilkan pada gambar 3.19

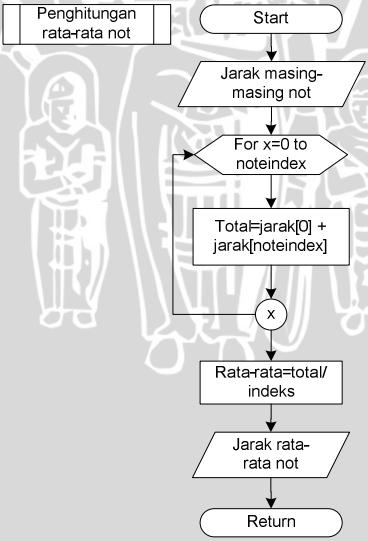


Gambar 3.19 Diagram alir pengukuran jarak not

d. Penghitungan rata-rata not

Setelah didapatkan informasi jarak masing-masing not, dilakukan penghitungan rata-rata jarak. Rumus yang digunakan yaitu persamaan 2.11.

Diagram alir predefined process dari penghitungan rata-rata not ditampilkan pada gambar 3.20



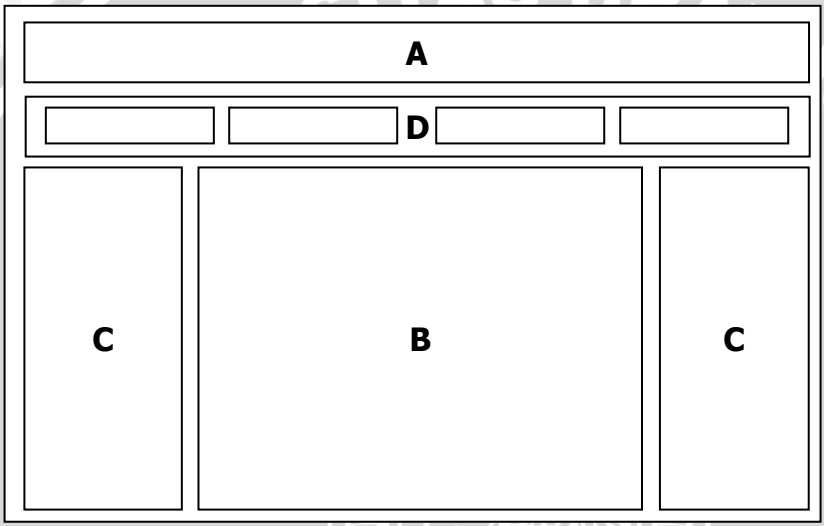
Gambar 3.20 Diagram alir penghitungan rata-rata not

3.2.4 Music Syntax Check

Setelah jarak rata-rata not diketahui, maka nilai yang didapatkan akan dibandingkan dengan nilai *threshold* untuk masing-masing batas suara yang telah ditentukan sebelumnya.

3.3. Perancangan Interface

Interface yang dirancang untuk perangkat lunak ini ditampilkan pada gambar 3.16.



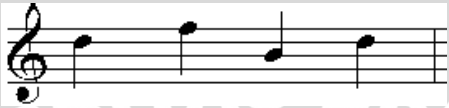
Gambar 3.21 Tampilan *user interfaces*

Adapun penjelasan masing-masing kotak dialog pada gambar 3.16 adalah:
A : Judul atau Nama Aplikasi
B : Menampilkan citra proses
C : Menampilkan hasil analisa proses *Fine Classification*
D : Menampilkan tombol proses

3.4. Perancangan Uji Coba

Penggolongan suara manusia pada citra partitur vokal dengan modifikasi algoritma KOMR dilakukan dengan mengukur jarak rata-rata semua not yang tercetak di dalam citra partitur, dengan garis paranada (*stave*) terbawah dalam citra partitur. Bila pada citra

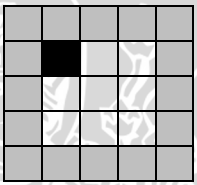
didapatkan hasil bahwa citra menggunakan kunci G, maka penggolongan suara didasarkan pada suara wanita. Bila pada citra didapatkan hasil bahwa citra menggunakan kunci F, maka penggolongan suara didasarkan pada suara pria. Uji coba diterapkan pada 1 citra partitur. Contoh citra partitur vokal yang akan diproses pada gambar 3.22



Gambar 3.22 Citra uji coba

- Pada proses *image enhancement* citra yang diubah ke dalam matriks tampak pada gambar 3.23. Contoh perhitungannya sebagai berikut:

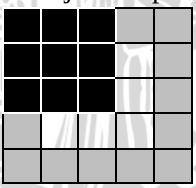
$$\begin{bmatrix} 30 & 40 & 30 & 20 & 45 \\ 40 & \mathbf{50} & 40 & 45 & 50 \\ 45 & 45 & 45 & 50 & 35 \\ 35 & 35 & 40 & 45 & 45 \\ 45 & 50 & 45 & 50 & 50 \end{bmatrix}$$



Gambar 3.23 Citra yang diubah ke dalam bentuk matriks

Warna abu-abu merupakan tepian citra yang tidak diproses. Yang pertama kali diproses adalah piksel [1,1] yang bernilai 50. Kemudian nilai dari 8 piksel sekitarnya diambil, dan dikalikan dengan matriks *window convolution*. Hasilnya dibagi dengan jumlah piksel matriks *window convolution*. Gambar 3.24 menunjukkan proses pertama.

$$\begin{bmatrix} \mathbf{30} & \mathbf{40} & \mathbf{30} & 20 & 45 \\ \mathbf{40} & \mathbf{50} & \mathbf{40} & 45 & 50 \\ \mathbf{45} & \mathbf{45} & \mathbf{45} & 50 & 35 \\ 35 & 35 & 40 & 45 & 45 \\ 45 & 50 & 45 & 50 & 50 \end{bmatrix}$$

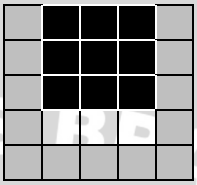


Gambar 3.24 Citra proses pertama

Piksel [1,1] =-1*30 + -1*40 + -1*30
+ -1*40 + 12*50 + -1*40
+ -1*45 + -1*45 + -1*45
=-30 + -40 + -30 + -40 + 600 + -40 + -45 + -45 + -45
=285 / 4 = 71

Nilai 71 ini kemudian ditukarkan dengan 50 pada piksel [1,1] pada citra baru hasil *enhancement*. Berikutnya 8 piksel di antara piksel [1,2] diambil dan dikalikan dengan matriks *windows convolution*. Hasilnya ditampilkan pada gambar 3.25.

$$\begin{bmatrix} 30 & \mathbf{40} & \mathbf{30} & \mathbf{20} & 45 \\ 40 & \mathbf{50} & \mathbf{40} & \mathbf{45} & 50 \\ 45 & \mathbf{45} & \mathbf{45} & \mathbf{50} & 35 \\ 35 & 35 & 40 & 45 & 45 \\ 45 & 50 & 45 & 50 & 50 \end{bmatrix}$$



Gambar 3.25 Citra proses kedua

Piksel [1,2] = -1*40 + -1*30 + -1*20
+ -1*50 + 12*40 + -1*45
+ -1*45 + -1*45 + -1*50
= -40 + -30 + -20 + -50 + 480 + -45 + -45 + -45 + -50
= 155 / 4 = 39

Nilai 39 ini kemudian ditukarkan dengan 40 pada piksel [1,2] pada citra baru hasil *enhancement*. Berikutnya 8 piksel di antara piksel berikutnya, yaitu piksel [1,3] diambil dan dikalikan dengan matriks *windows convolution*, dan seterusnya. Hingga hasil yang didapatkan pada citra baru hasil *enhancement* pada gambar 3.26.

$$\begin{bmatrix} 30 & 40 & 30 & 20 & 45 \\ 40 & 71 & 39 & 56 & 50 \\ 45 & 52 & 47 & 64 & 35 \\ 35 & 18 & 29 & 45 & 45 \\ 45 & 50 & 45 & 50 & 50 \end{bmatrix}$$

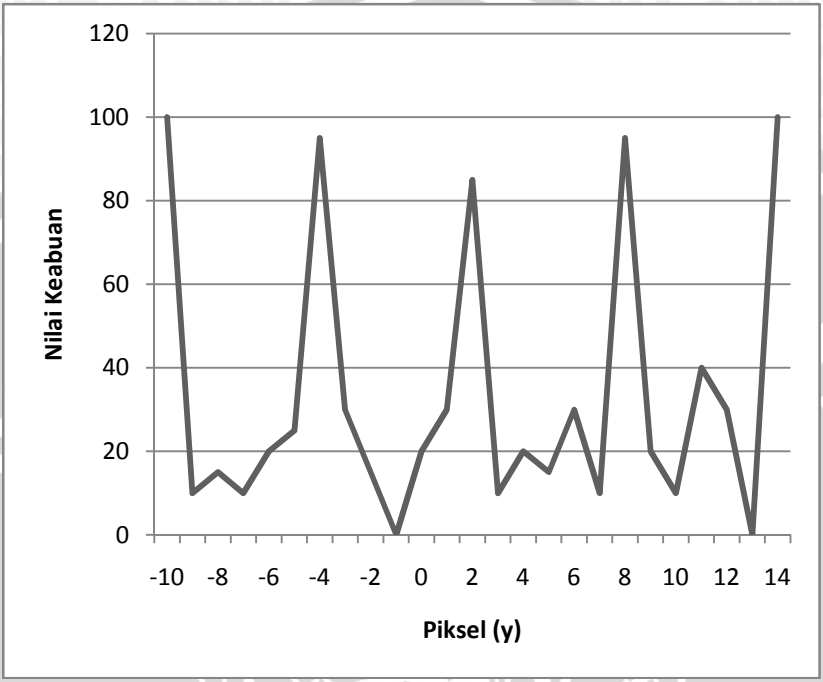
Gambar 3.26 Matriks citra hasil *Image Enhancement*

- Dari hasil *scanning* secara horisontal dari piksel citra paling tepi, didapatkan skala keabuan rata-rata untuk tiap garis horisontal. Misalkan data yang didapatkan ditampilkan pada tabel 3.1.

Tabel 3.1 Nilai keabuan tiap koordinat y pada citra

Piksel (y)	**-10**	**-9**	**-8**	**-7**	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**
Nilai Keabuan	100	10	15	10	20	25	95	30	15	0	20	30	85
Piksel (y)	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	
Nilai Keabuan	10	20	15	30	10	95	20	10	40	30	0	100	

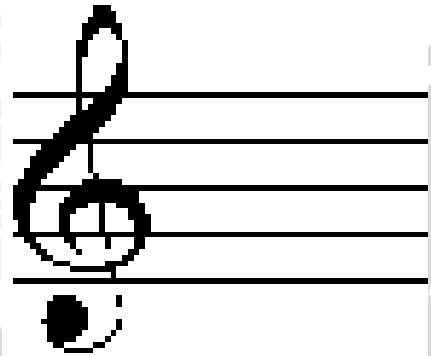
- Dari data tersebut dapat dibentuk histogram skala keabuan rata-rata untuk tiap garis horisontal, seperti pada gambar 3.27.



Gambar 3.27 Histogram skala keabuan

Dalam kasus ini, dengan menggunakan *threshold* nilai keabuan ≥ 80, maka koordinat (sumbu *y*) kandidat *stave* yang diperoleh yaitu garis horizontal pada piksel dengan koordinat y = -10,y =-4, y = 2, y = 8, dan y = 14.
Kemudian simpan posisi masing-masing garis *stave* (koordinat *y*).

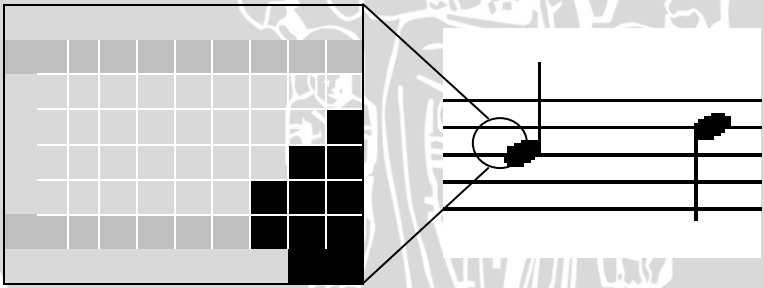
- Untuk mengetahui *threshold* dari masing-masing golongan suara, ditentukan koordinat *y* sesuai *range* golongan suara.
Contoh perhitungan:
Posisi garis paranada dari *stave detection* didapatkan seperti tampak pada gambar 3.28.



Gambar 3.28 Koordinat y pada *stave*

Maka *threshold* untuk masing-masing suara yaitu:
a. Sopran = $2 \leq y \leq 14$
b. Alto = $-10 \leq y < 2$
c. Tenor = $2 \leq y \leq 14$
d. Bass = $-10 \leq y < 2$

- Setelah koordinat *y stave* dan *threshold* masing-masing suara disimpan, maka dilakukan penghapusan *stave* dengan menerapkan *vertical run length connectivity*, seperti pada gambar 3.29.



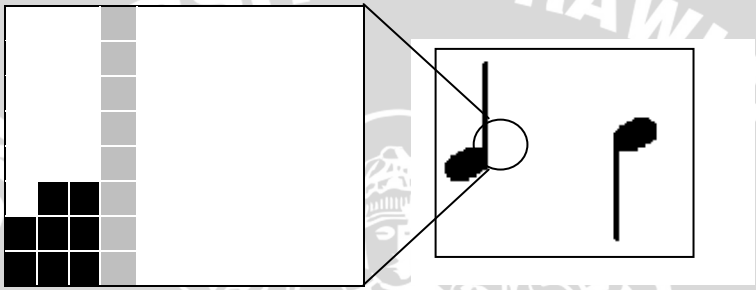
Gambar 3.29 Citra partitur dalam piksel

Piksel berwarna abu-abu dihapus karena tidak memenuhi syarat *vertical run length connectivity*. Sementara piksel berwarna hitam tidak dihapus karena memenuhi syarat.

Citra partitur setelah *stave* dihilangkan tampak pada gambar 3.30

Gambar 3.30 Citra setelah mengalami *stave removal*

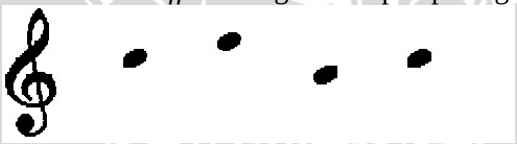
- Setelah *stave* dihapus, maka citra kembali diperiksa *vertical length connectivity*, untuk menghapus *staff*, seperti pada gambar 3.31.



Gambar 3.31 Citra partitur dalam piksel

Piksel berwarna abu-abu dihapus karena memenuhi syarat *vertical run length connectivity*. Sementara piksel berwarna hitam tidak dihapus karena tidak memenuhi syarat.

Citra partitur setelah *staff* dihilangkan tampak pada gambar 3.32.



Gambar 3.32 Citra setelah mengalami *staff removal*

- Kemudian dilakukan *symbol windowing* pada citra.

Sebelum proses *symbol windowing*, terlebih dahulu dilakukan proses inisialisasi dengan membuat *array* MusicNotes. MusicNotes berfungsi untuk menyimpan informasi koordinat window masing-masing simbol. Informasi ini disimpan dalam MusicNotes[].TopLeft dan MusicNotes[].BottomRight. Inisialisasi selanjutnya adalah membuat array PixelPrcssed dengan dimensi yang sama dengan citra biner. Semua elemen PixelPrcssed bertipe boolen dan diinisialisasi dengan nilai *false*. PixelPrcssed berfungsi untuk menandai piksel

yang seudah diproses sehingga piksel tidak akan diproses lagi secara berulang-ulang. Lakukan iterasi pada piksel citra biner D. Bila piksel citra D pada indeks [i,j] adalah piksel hitam dan piksel tersebut belum diproses (ditunjukkan dengan PixelPrcssed pada indeks [i,j] yang bernilai false) maka lakukan pengecekan terhadap piksel-piksel di sekitar D[i,j]. *Pseudocode* pengecekan piksel seperti tampak pada gambar 3.33.

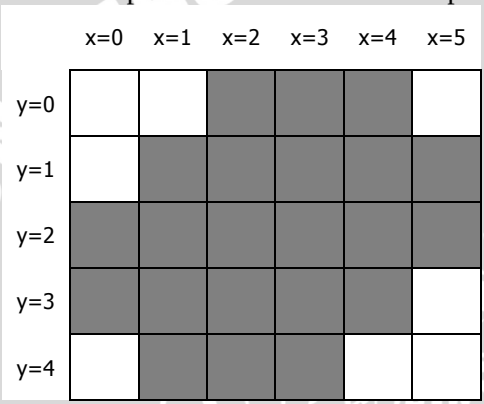
```
procedure CheckNeighbourPixels(binaryImg, x, y)
begin
  if (piksel[x,y] dr binaryImg adl piksel hitam)
  begin
    if (piksel[x,y] belum diproses)
      begin
        if (x <= MusicNotes[a].TopLeft.x)
          then MusicNotes[a].TopLeft.x = x;
        if (y <= MusicNotes[a].TopLeft.y)
          then MusicNotes[a].TopLeft.y = y;
        if (x > MusicNotes[a].BottomRight.x)
          then MusicNotes[a].BottomRight.x = x;
        if (y > MusicNotes[a].BottomRight.y)
          then MusicNotes[a].BottomRight.y = y;
        Tandai piksel[x,y] untuk menunjukkan bahwa piksel telah
        diproses;

        // Periksa ke-8 piksel di sekitar piksel ini (secara
        // rekursif)
        CheckNeighbourPixels(binaryImg, x-1, y-1);
        CheckNeighbourPixels(binaryImg, x, y-1);
        CheckNeighbourPixels(binaryImg, x+1, y-1);
        CheckNeighbourPixels(binaryImg, x-1, y);
        CheckNeighbourPixels(binaryImg, x+1, y);
        CheckNeighbourPixels(binaryImg, x-1, y+1);
        CheckNeighbourPixels(binaryImg, x, y+1);
        CheckNeighbourPixels(binaryImg, x+1, y+1);

    end;
  end;
end;
```

Gambar 3.33 *Pseudocode symbol windowing*

Pengecekan piksel ini dilakukan secara rekursif melalui fungsi CheckNeighbourPixels. Piksel yang akan terus diproses secara rekursif adalah piksel-piksel yang memenuhi syarat: piksel tersebut merupakan piksel hitam dan juga masih belum diproses. Pada saat yang sama, simpan koordinat dari *symbol window*. Simpan koordinat ini dengan syarat, untuk koordinat MusicNotes[].TopLeft, koordinat

(x,y)-nya adalah koordinat (x,y) yang terkecil, dan untuk koordinat MusicNotes[].BottomRight, koordinat (x,y)-nya adalah koordinat (x,y) yang terbesar. Tandai piksel ini untuk menunjukkan bahwa piksel telah diproses sehingga pada *loop* rekursif selanjutnya piksel ini tidak akan diproses kembali. Periksa kedelapan piksel lain di sekitar piksel ini dengan memanggil fungsi CheckNeighbourPixels secara rekursif. Proses rekursif ini akan berhenti saat tidak ada lagi piksel hitam yang belum diproses. Setiap kali proses rekursif selesai akan dihasilkan satu *symbol window*.

Misal didapatkan simbol not musik seperti pada gambar 3.34.



Gambar 3.34 Citra not musik dalam piksel

1). Inisialisasi TopLeft.x = 10000, TopLeft.y = 10000 dan BottomRight.x = -10000, BottomRight.y = -10000
2). Pada proses pertama didapatkan piksel hitam pada koordinat (2,0). Karena x=2 ≤ 10000 dan y=0 ≤ 10000, maka koordinat TopLeft dari citra adalah (2,0). Karena x=2 > -10000 dan y=0 > -10000, maka koordinat BottomRight dari citra adalah (2,0)
3). Tandai piksel (2,0) sudah diproses.
4). Dilakukan pemeriksaan ke-8 piksel di sekitar piksel (2,0)
5). Dilakukan pemeriksaan piksel hitam tetangganya, yaitu piksel (3,0). Karena x=3 bukan ≤ 2 dan y=0 ≤ 0, maka koordinat TopLeft dari citra adalah (2,0). Karena x=3 > 2 dan y=0 bukan > 0, maka koordinat BottomRight dari citra adalah (3,0)
6). Tandai piksel (3,0) sudah diproses.
7). Demikian seterusnya dilakukan hingga semua piksel pada citra diproses.

8). Didapatkan koordinat TopLeft dari citra yaitu (0,0), dan koordinat BottomRight dari citra yaitu (5,4)

Citra partitur setelah dilakukan *symbol windowing* tampak pada gambar 3.35.

Gambar 3.35 Citra setelah mengalami *symbol windowing*

- Dari *symbol windowing*, setiap simbol musik kemudian dibandingkan dengan template simbol musik yang telah tersedia. Template yang disediakan yaitu 𝄞 𝄢 𝅗𝅥 𝅘𝅥 dengan ukuran *window* tertentu. Misal tersedia template seperti pada gambar 3.36

Window 30x80 Σ piksel hitam 900 | window 30x80 Σ piksel hitam 900 | window 10x40 Σ piksel hitam 200 | window 10x40 Σ piksel hitam 200

Gambar 3.36 *Template* simbol musik

Setiap simbol musik yang terdapat dalam citra dibandingkan dengan keempat *template* tersebut. Misal diambil salah satu simbol 𝅗𝅥 pada citra:

1). Diperoleh informasi simbol 𝅗𝅥 memiliki ukuran window 9x39. Window *simbol* ini kemudian di*resize* sesuai ukuran masing-masing *template*.

2). Setelah dibandingkan dengan masing-masing template, diperoleh matching score dengan template 1 sebesar 600, dengan template 2 sebesar 400, dengan template 3 sebesar 200, dengan template 4 sebesar 100.

3). Maka nilai 1 sebesar : 600/900 * 100 = 67

Nilai 2 sebesar : 400/900 * 100 = 44

Nilai 3 sebesar : 200/200 * 100 = 100

Nilai 4 sebesar : 100/200 * 100 = 50

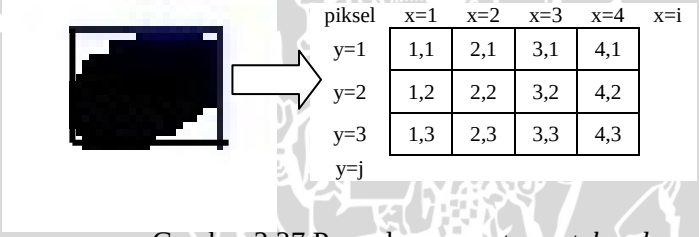
4). Nilai terbesar yang diperoleh dari template 3. Maka simbol tersebut dikenali sebagai simbol ♩

- Demikian seterusnya hingga semua simbol dikenali. Selanjutnya diketahui bahwa citra tersebut menggunakan kunci G.

- Setelah diketahui pengunaan kunci, selanjutnya diperiksa koordinat semua piksel dari citra hasil *coarse classification*. Kemudian dilakukan perhitungan piksel tengah dari not (*center notehead*). Hasil yang didapatkan merupakan koordinat piksel yang digunakan untuk mengukur jarak not terhadap garis *stave* terbawah.

Contoh perhitungan:

Citra yang didapat dari *symbol windowing* diambil satu persatu untuk dihitung koordinat titik tengahnya, seperti pada gambar 3.37



Gambar 3.37 Pengukuran *center notehead*

$$(\overline{x_c}, \overline{y_c}) = \frac{\left(\sum_{n=1}^{j} n\right),\left(\sum_{m=1}^{i} m\right)}{i*j} \text{ di mana } n = \left(\sum_{x=1}^{i} x\right), m = \left(\sum_{y=1}^{j} y\right)$$

$$= \frac{(1+2+3+4+1+2+3+4+1+2+3+4),(1+2+3+1+2+3+1+2+3+1+2+3)}{4*3}$$

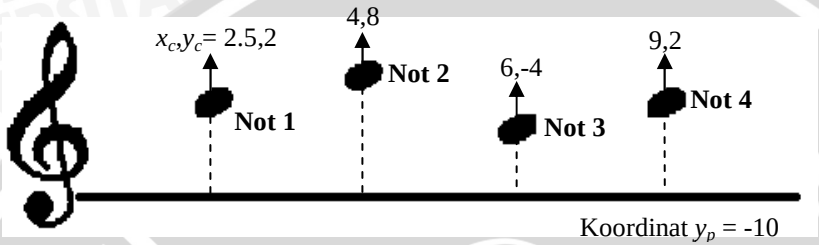
$$= \frac{(30),(24)}{12}$$

$$= 2.5,2$$

Maka koordinat dari *center notehead* tersebut adalah (2.5,2)

- Setelah didapatkan semua koordinat *center notehead*, dilakukan pengukuran masing-masing not terhadap koordinat *y stave* terbawah.
Contoh perhitungan:

Dari perhitungan *center notehead* terhadap masing-masing not, diperoleh hasil seperti pada gambar 3.38



Gambar 3.38 Koordinat *center notehead*

Semua koordinat tersebut dimasukkan ke dalam perhitungan:

$$r = (y_c) - (y_p)$$

r_1= 2 - (-10) = 12
r_2 = 8 - (-10) = 18
r_3 = -4 - (-10) = 6
r_4 = 2 - (-10) = 12

Maka jarak 4 not tersebut terhadap *stave* terbawah yaitu: not 1 = 12, not 2 = 18, not 3 = 6, not 4 = 12.

- Setelah didapatkan informasi jarak masing-masing not, dilakukan penghitungan rata-rata jarak.
Contoh perhitungan:

Dari jarak 4 not yang didapatkan di langkah sebelumnya, maka rata-rata jaraknya yaitu:

$$\bar{r} = \frac{\sum_{i=0}^{n}(r_i)}{n}$$
$$= \frac{12+18+6+12}{4}$$
$$= \frac{48}{4} = 12$$

Maka citra partitur tersebut memiliki jarak rata-rata terhadap *stave* terbawah sebesar 12.

- Setelah jarak rata-rata not diketahui, maka nilai yang didapatkan akan dibandingkan dengan nilai *threshold* untuk masing-masing batas suara yang telah ditentukan sebelumnya.

Contoh perhitungan :

Karena dari citra partitur menggunakan kunci G, dan r (jarak rata-rata not) didapatkan sebesar 12, maka golongan suara pada citra partitur tersebut adalah sopran.

- Setelah citra diproses, maka hasil yang diperoleh akan dimasukkan ke dalam tabel, untuk mencatat indeks data percobaan citra partitur, posisi *stave* ke 5, penggolongan kunci yang diperoleh, penentuan *threshold* suara, penentuan jarak rata-rata not, serta penggolongan suara akhir. Data pada tabel ini akan digunakan untuk mengevaluasi hasil yang diperoleh dari perangkat lunak penggolongan suara. Rancangan tabel yang digunakan ditampilkan pada tabel 3.2

Tabel 3.2 Tabel Evaluasi Hasil

Citra Partitur	Stave Terbawah	Penggolongan Kunci	Threshold Suara	Jarak rata-rata	Golongan Suara

BAB IV
HASIL DAN PEMBAHASAN

Implementasi merupakan proses transformasi representasi rancangan ke bahasa pemrograman yang dapat dimengerti oleh komputer. Bab ini akan membahas hasil implementasi yang dihasilkan oleh perangkat lunak, untuk selanjutnya dilakukan evaluasi terhadap hasil penggolongan suara oleh sistem.

4.1 Lingkungan Implementasi

Lingkungan implementasi meliputi lingkungan perangkat keras serta lingkungan perangkat lunak.

4.1.1 Lingkungan perangkat keras

Perangkat keras yang digunakan dalam pengembangan sistem pendeteksi poin-poin *minutiae* ini adalah:

1. Intel(R) Core(TM)2 Duo – Processor 2,1 GHz.
2. RAM 1,87 GB
3. *Harddisk* dengan kapasitas 250 GB
4. Perangkat keluaran berupa monitor 14 inchi
5. Perangkat masukan berupa keyboard dan mouse

4.1.2 Lingkungan perangkat lunak

Perangkat lunak yang digunakan dalam pengembangan sistem ini adalah :

1. Sistem Operasi *Microsoft Windows 7 Ultimate*
2. *Borland Delphi 7*

4.2 Implementasi Program

Berdasarkan perancangan perangkat lunak maka pada subbab ini akan dibahas mengenai implementasi dari perancangan tersebut.

4.2.1 *Input* Citra Partitur

Proses *input* yang dilakukan oleh *user* dengan membuka *file* citra partitur yang telah tersimpan sebelumnya. Format *file* citra yang dapat dikenali ada dua, yaitu ekstensi bmp dan jpg. Gambar 4.1 menunjukkan cara membuka *file* citra.

```
procedure TFormMain.RibbonMainMenuItems0Click(Sender: TObject);
var
  SUCCEED: Boolean;
begin
  RibbonMainMenu.ActivePage := nil;
  InitVariable;
  SUCCEED := True;
  if (OpenImageDialog.Execute) then
  begin
    try
      try
        ImageMusicScore.Picture.LoadFromFile(OpenImageDialog.FileName);
        OriginalBitmap := TBitmap.Create;
        OriginalBitmap.Width := ImageMusicScore.Picture.Graphic.Width;
        OriginalBitmap.Height := ImageMusicScore.Picture.Graphic.Height;
        if (ImageMusicScore.Picture.Graphic is TBitmap) then
        begin
          with TBitmap(ImageMusicScore.Picture.Graphic) do
          begin
            if (PixelFormat = pf24bit) or (PixelFormat = pf32bit) then
            begin
              OriginalBitmap.PixelFormat := PixelFormat;
            end
            else
            begin
              PixelFormat := pf24bit;
              OriginalBitmap.PixelFormat := pf24bit;
            end
          end;
        end
        else if (ImageMusicScore.Picture.Graphic is TJPEGImage) then
        begin
          with TJPEGImage(ImageMusicScore.Picture.Graphic) do
          begin
            PixelFormat := jf24Bit;
            Scale := jsFullSize;
            Grayscale := false;
            Performance := jpBestQuality;
            ProgressiveDisplay := false;
            OriginalBitmap.PixelFormat := pf24bit;
          end;
        end;
```

Gambar 4.1 *Sourcecode* cara membuka file citra

Selanjutnya di dalam proses membuka file citra, terdapat proses *enhance* (dengan menerapkan konvolusi) dan pengubahan citra menjadi *binary*.

4.2.2 *Image Enhancement*

Proses *image enhancement* dilakukan dengan melakukan proses konvolusi, seperti pada gambar 4.2.

```
OriginalBitmap.Canvas.Draw(0, 0, ImageMusicScore.Picture.Graphic);
OriginalBitmap.Canvas.Draw(0, 0, Greyscale(OriginalBitmap));
// Konvolusikan dengan Simple Laplacian
SimpleLaplacian := TConvolution.Create;
ModifiedBitmap := TBitmap.Create;
ModifiedBitmap.Width := OriginalBitmap.Width;
ModifiedBitmap.Height := OriginalBitmap.Height;
ModifiedBitmap.PixelFormat := OriginalBitmap.PixelFormat;
ModifiedBitmap.Canvas.Draw(0, 0, SimpleLaplacian.Convolve(OriginalBitmap));
```

Gambar 4.2. *Sourcecode* proses konvolusi

Proses konvolusi dibedakan menjadi dua jenis, yaitu untuk format citra 24 bit (bmp) dan format citra 32 bit (jpg). Fungsi dari proses konvolusi ditampilkan pada gambar 4.3

```
function TConvolution.Convolve(bitmap: TBitmap): TBitmap;
var
  a, b, x, y: Integer;
  coef: Real;
  newRed, newGreen, newBlue: Real;
  row24_1, row24_2, row24_3, rowOutput24: PRGBTripleArray;
  row32_1, row32_2, row32_3, rowOutput32: PRGBQuadArray;
begin
  coef := 0;
  for a := 0 to 2 do   // cari nilai koefisien pembagi
    for b := 0 to 2 do
      coef := coef + kernel[a, b];
  if (coef = 0) then
    coef := 1;
  try
  begin
    Result := TBitmap.Create;
    Result.Width := bitmap.Width;
    Result.Height := bitmap.Height;
    Result.Canvas.Draw(0, 0, bitmap);
    Result.PixelFormat := bitmap.PixelFormat;
    if (bitmap.PixelFormat = pf24bit) then // 24 bit
    begin
      row24_2 := bitmap.Scanline[0];
      row24_3 := bitmap.ScanLine[1];
      for y := 1 to bitmap.Height-2 do
```

```
    begin
      row24_1 := row24_2;
      row24_2 := row24_3;
      row24_3 := bitmap.ScanLine[y+1];
      rowOutput24 := Result.ScanLine[y];
      for x := 1 to bitmap.Width-2 do
      begin
        // Konvolusi per chanel warna
        // Red
        newRed := (kernel[0, 0]*row24_1[x - 1].rgbtRed) + (kernel[1,
0]*row24_1[x].rgbtRed) +  (kernel[2, 0]*row24_1[x + 1].rgbtRed) + (kernel[0,
1]*row24_2[x - 1].rgbtRed) + (kernel[1, 1]*row24_2[x].rgbtRed) + (kernel[2,
1]*row24_2[x + 1].rgbtRed) +  (kernel[0, 2]*row24_3[x - 1].rgbtRed) + (kernel[1,
2]*row24_3[x].rgbtRed) +  (kernel[2, 2]*row24_3[x + 1].rgbtRed);
        newRed := newRed / coef;
        if (newRed > 255) then newRed := 255
        else if (newRed < 0) then newRed := 0;
        // Green
        newGreen := (kernel[0, 0]*row24_1[x - 1].rgbtGreen) + (kernel[1,
0]*row24_1[x].rgbtGreen) + (kernel[2, 0]*row24_1[x + 1].rgbtGreen) + (kernel[0,
1]*row24_2[x - 1].rgbtGreen) + (kernel[1, 1]*row24_2[x].rgbtGreen) + (kernel[2,
1]*row24_2[x + 1].rgbtGreen) + (kernel[0, 2]*row24_3[x - 1].rgbtGreen) +
(kernel[1, 2]*row24_3[x].rgbtGreen) + (kernel[2, 2]*row24_3[x + 1].rgbtGreen);
        newGreen := newGreen / coef;
        if (newGreen > 255) then newGreen := 255
        else if (newGreen < 0) then newGreen := 0;
        // Blue
        newBlue := (kernel[0, 0]*row24_1[x - 1].rgbtBlue) + (kernel[1,
0]*row24_1[x].rgbtBlue) + (kernel[2, 0]*row24_1[x + 1].rgbtBlue) + (kernel[0,
1]*row24_2[x - 1].rgbtBlue) + (kernel[1, 1]*row24_2[x].rgbtBlue) + (kernel[2,
1]*row24_2[x + 1].rgbtBlue) + (kernel[0, 2]*row24_3[x - 1].rgbtBlue) + (kernel[1,
2]*row24_3[x].rgbtBlue) + (kernel[2, 2]*row24_3[x + 1].rgbtBlue);
        newBlue := newBlue / coef;
        if (newBlue > 255) then newBlue := 255
        else if (newBlue < 0) then newBlue := 0;
        rowOutput24[x].rgbtRed := Trunc(newRed);
        rowOutput24[x].rgbtGreen := Trunc(newGreen);
        rowOutput24[x].rgbtBlue := Trunc(newBlue);
      end;
    end;
  end
```

```
    else // 32 bit
    begin
      row32_2 := bitmap.Scanline[0];
      row32_3 := bitmap.ScanLine[1];
      for y := 1 to bitmap.Height-2 do
      begin
        row32_1 := row32_2;
        row32_2 := row32_3;
        row32_3 := bitmap.ScanLine[y+1];
        rowOutput32 := Result.ScanLine[y];
        for x := 1 to bitmap.Width-2 do
        begin
          // Konvolusi per chanel warna
          // Red
          newRed := (kernel[0, 0]*row32_1[x - 1].rgbRed) + (kernel[1,
0]*row32_1[x].rgbRed) + (kernel[2, 0]*row32_1[x + 1].rgbRed) + (kernel[0,
1]*row32_2[x - 1].rgbRed) + (kernel[1, 1]*row32_2[x].rgbRed) + (kernel[2,
1]*row32_2[x + 1].rgbRed) + (kernel[0, 2]*row32_3[x - 1].rgbRed) + (kernel[1,
2]*row32_3[x].rgbRed) + (kernel[2, 2]*row32_3[x + 1].rgbRed);
          newRed := newRed / coef;
          if (newRed > 255) then newRed := 255
          else if (newRed < 0) then newRed := 0;
          // Green
          newGreen := (kernel[0, 0]*row32_1[x - 1].rgbGreen) + (kernel[1,
0]*row32_1[x].rgbGreen) + (kernel[2, 0]*row32_1[x + 1].rgbGreen) + (kernel[0,
1]*row32_2[x - 1].rgbGreen) + (kernel[1, 1]*row32_2[x].rgbGreen) + (kernel[2,
1]*row32_2[x + 1].rgbGreen) + (kernel[0, 2]*row32_3[x - 1].rgbGreen) +
(kernel[1, 2]*row32_3[x].rgbGreen) + (kernel[2, 2]*row32_3[x + 1].rgbGreen);
          newGreen := newGreen / coef;
          if (newGreen > 255) then newGreen := 255
          else if (newGreen < 0) then newGreen := 0;
          // Blue
          newBlue := (kernel[0, 0]*row32_1[x - 1].rgbBlue) + (kernel[1,
0]*row32_1[x].rgbBlue) + (kernel[2, 0]*row32_1[x + 1].rgbBlue) + (kernel[0,
1]*row32_2[x - 1].rgbBlue) + (kernel[1, 1]*row32_2[x].rgbBlue) + (kernel[2,
1]*row32_2[x + 1].rgbBlue) + (kernel[0, 2]*row32_3[x - 1].rgbBlue) + (kernel[1,
2]*row32_3[x].rgbBlue) + (kernel[2, 2]*row32_3[x + 1].rgbBlue);
          newBlue := newBlue / coef;
          if (newBlue > 255) then newBlue := 255
          else if (newBlue < 0) then newBlue := 0;
          rowOutput32[x].rgbRed := Trunc(newRed);
          rowOutput32[x].rgbGreen := Trunc(newGreen);
          rowOutput32[x].rgbBlue := Trunc(newBlue);
        end;
      end;
```

Gambar 4.3 *Sourcecode* fungsi konvolusi

4.2.3 Binerisasi

Proses binerisasi citra bertujuan untuk mengubah format citra bmp dan jpg menjadi format biner. Hal ini dilakukan untuk memudahkan pemrosesan selanjutnya. *Source code* dari proses binerisasi ditampilkan pada gambar 4.4

```
function TBinaryImage.ConvertToBitmap: TBitmap;
var
  x, y: Integer;
  red1, green1, blue1, red2, green2, blue2: Byte;
  pRGB24: PRGBTripleArray;
  pRGB32: PRGBQuadArray;
begin
  try
  begin
    Result := TBitmap.Create;
    Result.Width := Width;
    Result.Height := Height;
    Result.PixelFormat := BitmapPixelFormat;
    // nilai untuk setiap chanel RGB dari warna background
    red1 := GetRValue(ColorBg); green1 := GetGValue(ColorBg); blue1 :=
GetBValue(ColorBg);
    // nilai untuk setiap chanel RGB dari warna non-background
    red2 := GetRValue(ColorNonBg); green2 := GetGValue(ColorNonBg); blue2 :=
GetBValue(ColorNonBg);
    if (BitmapPixelFormat = pf24bit) then
    begin
      for y := 0 to Height-1 do
      begin
        pRGB24 := Result.ScanLine[y];
        for x := 0 to Width-1 do
        begin
          if (Pixels[x, y].isPixelNote = True) then
          begin
            pRGB24[x].rgbtRed := red2;
            pRGB24[x].rgbtGreen := green2;
            pRGB24[x].rgbtBlue := blue2;
          end
          else
          begin
            pRGB24[x].rgbtRed := red1;
            pRGB24[x].rgbtGreen := green1;
            pRGB24[x].rgbtBlue := blue1;
          end;
        end;
      end;
    end
```

```
else if (BitmapPixelFormat = pf32bit) then
begin
  for y := 0 to Height-1 do
  begin
    pRGB32 := Result.ScanLine[y];
    for x := 0 to Width-1 do
    begin
      if (Pixels[x, y].isPixelNote = True) then
      begin
        pRGB32[x].rgbRed := red2;
        pRGB32[x].rgbGreen := green2;
        pRGB32[x].rgbBlue := blue2;
        pRGB32[x].rgbReserved := 255;
      end
      else
      begin
        pRGB32[x].rgbRed := red1;
        pRGB32[x].rgbGreen := green1;
        pRGB32[x].rgbBlue := blue1;
        pRGB32[x].rgbReserved := 255;
      end;
    end;
  end;
end;
```

Gambar 4.4 *Sourcecode* binerisasi

Gambar 4.5 menunjukkan citra asli, dan citra setelah mengalami proses konvolusi dengan matriks $H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix}$, didapatkan berdasarkan jurnal yang ditulis oleh (Kim, 1987), serta binerisasi.

(a) (b)

Gambar 4.5 (a) Citra partitur asli (b) *Enhanced image*

4.2.4 *Stave Detection*

Stave detection dilakukan terhadap citra hasil *enhancement*. Dalam *source code stave detection*, terdapat fungsi untuk melakukan *horizontal projection*. Proses ini bertujuan untuk membentuk *grayscale* histogram, yang digunakan untuk mengetahui *stave candidate* dari citra. *Source code horizontal projection* ditampilkan pada gambar 4.6

```
function TMusicScore.CreateHorizontalProjection(binaryImg: TBinaryImage):
Boolean;
var
  x, y: Integer;
  SUCCEED: Boolean;
begin
  SUCCEED := True;
  if (binaryImg = nil) then
    Result := not SUCCEED;
  /// Cari horizontal projection
  // Inisialisasi array horizontal projection
  SetLength(HorzProjection, binaryImg.Height);
  for y := 0 to binaryImg.Height-1 do
      HorzProjection[y] := 0;
  for y := 0 to binaryImg.Height-1 do
  begin
    for x := 0 to binaryImg.Width-1 do
    begin
      if (binaryImg.Pixels[x, y].isPixelNote = True) then
      begin
        Inc(HorzProjection[y]);
      end
    end;
  end;
  Result := SUCCEED;
end;
```

Gambar 4.6 *Sourcecode horizontal projection*

Horizontal projection ini digunakan dalam source code proses *stave detection*. Langkah-langkah dalam stave detection meliputi:

1). *Horizontal projection*

2). Pemilihan garis horizontal terpanjang maxLengthStave

3). Penentuan kandidat *stave*

Seluruh proses tersebut ditampilkan dalam *source code stave detection* pada gambar 4.7

```
function TMusicScore.DetectStaves(binaryImg: TBinaryImage): Boolean;
type
  TTmpStaveCandidate = record
    isStave: Boolean;
    index: Integer;
    staveCand: TStaveCandidate;
  end;
var
  x, y, a, b, c, d, e: Integer;
  maxLengthStave: TTmpStaveCandidate;
  tmpStave: array of TTmpStaveCandidate;
  staveMemberTotal: array [1..5] of Integer;
  thresholdLength: Real;
  difference: Integer;
  isStartStaveIndexFlag: Boolean;
  staveCandTotal, startIndex, staveIndex: Integer;
  SUCCEED: Boolean;
begin
  SUCCEED := True;
  try
    // Buat horizontal projection
    if (CreateHorizontalProjection(binaryImg) = False) then
      Result := not SUCCEED;
    /// Cari temporary kandidat stave.
    // Terlebih dahulu cari garis horizontal terpanjang maxLengthStave dengan
    // memilih elemen HorzProjection yang bernilai paling besar.
    maxLengthStave.isStave := True;
    maxLengthStave.index := -1;
    maxLengthStave.staveCand.Length := -1;
    maxLengthStave.staveCand.Pos.X := -1;
    maxLengthStave.staveCand.Pos.Y := -1;
    for y := 0 to binaryImg.Height-1 do
    begin
      if (HorzProjection[y] > maxLengthStave.staveCand.Length) then
      begin
        maxLengthStave.staveCand.Length := HorzProjection[y];
        maxLengthStave.staveCand.Pos.Y := y;
      end;
    end;
    if (maxLengthStave.staveCand.Length <= 1) then
    begin
      //MessageDlg('Citra input tidak memiliki stave!', mtError, [mbOK], 0);
      Result := not SUCCEED;
    end;
    // Dengan menggunakan maxLengthStave sebagai tolak ukur, tentukan kandidat stave
    // lainnya.
    staveCandTotal := 0;
    thresholdLength := (FormMain.Config.StaveLengthPercentage / 100) *
maxLengthStave.staveCand.Length;
    for y := 0 to binaryImg.Height-1 do
    begin
      if (HorzProjection[y] >= thresholdLength) then
```

```
begin
      Inc(staveCandTotal);
      SetLength(tmpStave, staveCandTotal);
      a := staveCandTotal - 1;
      tmpStave[a].isStave := True;
      tmpStave[a].index := -1;
      tmpStave[a].staveCand.Length := HorzProjection[y];
      tmpStave[a].staveCand.Pos.Y := y;
      for x := 0 to binaryImg.Width-1 do
      begin
        if (binaryImg.Pixels[x, y].isPixelNote = True) then
        begin
          tmpStave[a].staveCand.Pos.X := x;
          break;
        end;
      end;
    end;
  end;
  if (High(tmpStave) <= 4) then
  begin
    Result := not SUCCEED;
  end;
  // Kelompokkan masing-masing kandidat stave untuk membentuk satu stave. Pengelompokan
  // kandidat stave dilakukan dengan syarat apabila 2 kandidat stave saling berhimpitan.
  // 2 kandidat stave dikatakan berhimpitan bila kandidat stave yang satu berada pada
  // indeks y dan kandidat stave yang lain berada pada indeks y-1 atau y+1
  staveIndex := 1;
  startIndex := -1;
  RowCount := 0;
  isStartStaveIndexFlag := True;
  for a := 0 to staveCandTotal-1 do
  begin
    if (isStartStaveIndexFlag = True) then
    begin
      startIndex := a;
      tmpStave[a].index := staveIndex;
      isStartStaveIndexFlag := False;
    end
    else
    begin
      difference := Round(Abs(tmpStave[a].staveCand.Pos.Y - tmpStave[a-1].staveCand.Pos.Y));
      // Bila berhimpitan
      if (difference = 1) then
      begin
        tmpStave[a].index := tmpStave[a-1].index;
      end
      else
      begin
        // Bila stave yang terkumpul masih <= 5
        if (staveIndex < 5) then
        begin
          Inc(staveIndex);
          tmpStave[a].index := staveIndex;
          if ( a = (staveCandTotal-1) ) then
            Result := not SUCCEED;
        end;
      end;
```

```
      // Bila kelima stave sudah ditemukan, buat satu MusicNotesRow
      if (staveIndex = 5) then
      begin
        // Pastikan tidak ada lagi kandidat stave lain yang berindeks 5.
        d := a + 1;
        difference := 0;
        if (d <= staveCandTotal-1) then
        begin
          difference := Abs(tmpStave[d].staveCand.Pos.Y - tmpStave[a].staveCand.Pos.Y);
        end;
        // Bila sudah tidak ada lagi kandidat stave lain yang berindeks 5
        if (difference <> 1) then
        begin
          // Inisialisasi MusicNotesRows - MusicNotesRows.Staves
          SetLength(MusicNotesRows, RowCount+1);
          MusicNotesRows[RowCount] := TMusicNotesRow.Create;
          for d := 1 to 5 do
          begin
            MusicNotesRows[RowCount].Staves[d] := TStave.Create;
          end;
          // Hitung banyaknya kandidat stave yang menyusun stave
          for d := 1 to 5 do
          begin
            staveMemberTotal[d] := 0;
          end;
          for b := startIndex to a do
          begin
            Inc(staveMemberTotal[tmpStave[b].index]);
          end;
          for d := 1 to 5 do
          begin
            SetLength(MusicNotesRows[RowCount].Staves[d].Stave, staveMemberTotal[d]);
            MusicNotesRows[RowCount].Staves[d].NumberOfStaveCandidate :=
staveMemberTotal[d];
            MusicNotesRows[RowCount].Staves[d].StaveIndex := d;
            c := 0;
            for b := startIndex to a do
            begin
              if (tmpStave[b].index = d) then
              begin
                MusicNotesRows[RowCount].Staves[d].Stave[c] := tmpStave[b].staveCand;
                Inc(c);
                if (c = staveMemberTotal[d]) then
                  break;
              end;
            end;
          end; // end for d := 1 to 5 do
          Inc(RowCount);
          staveIndex := 1;
          isStartStaveIndexFlag := True;
        end; // end if (difference)
      end; // end if (staveIndex = 5)
    end;
  end;
```

Gambar 4.7 *Source code stave detection*

Gambar 4.8 menunjukkan citra hasil proses *stave detection*

Gambar 4.8 Citra setelah proses *stave detection*

4.2.5 *Stave Removal*

Pada langkah *stave removal* ini, dilakukan penghapusan semua kandidat *stave* pada citra yang telah terdeteksi, seperti pada gambar 4.9.

```
function TMusicScore.RemoveStave(binaryImg: TBinaryImage): TBinaryImage;
var
  z, u, v, w, a, b, lengths, pixelCount: Integer;
  tmpBinaryImg: TBinaryImage;
  isPixel: Boolean;
begin
  try
    tmpBinaryImg := TBinaryImage.Create(binaryImg);
    for z := 0 to RowCount-1 do
    begin
      for u := 1 to 5 do
      begin
        with MusicNotesRows[z].Staves[u] do
        begin
          if (Length(Stave) > 0) then
          begin
            /// Hapus stave
            // Hapus setiap kandidat stave
            lengths := Length(Stave) - 1;
            for v := 0 to lengths do
            begin
              w := Stave[v].Pos.X + Stave[v].Length - 1;
              for a := Stave[v].Pos.X to w do
              begin
                tmpBinaryImg.Pixels[a, Stave[v].Pos.Y].isPixelNote := False;
              end;
            end;
```

Gambar 4.9 *Source code* proses *stave removal*

Setelah dilakukan penghapusan *stave*, maka perlu dilakukan proses untuk memperbaiki piksel yang hilang pada simbol-simbol musik yang berpotongan dengan *stave*. *Source code* dari proses repair ditampilkan pada gambar 4.10

```
        // Repair bagian stave yang berpotongan dengan music note
        w := Stave[0].Pos.X + Stave[0].Length - 1;
        for a := Stave[0].Pos.X to w do
        begin
          isPixel := False;
          pixelCount := 0;
          for b := 1 to FormMain.Config.RLConnectivity do
          begin
            v := Stave[0].Pos.Y - b;
            if (tmpBinaryImg.Pixels[a, v].isPixelNote = True) then
            begin
              if (b = 1) then
                isPixel := True;
              Inc(pixelCount);
            end;
          end;
          if (isPixel = True) and (pixelCount = FormMain.Config.RLConnectivity) then
            for v := 0 to lengths do
            begin
              tmpBinaryImg.Pixels[a, Stave[v].Pos.Y].isPixelNote := True;
            end;
        end; // end for a :=
        lengths := Length(Stave) - 1;
        w := Stave[lengths].Pos.X + Stave[lengths].Length - 1;
        for a := Stave[lengths].Pos.X to w do
        begin
          isPixel := False;
          pixelCount := 0;
          for b := 1 to FormMain.Config.RLConnectivity do
          begin
            v := Stave[lengths].Pos.Y + b;
            if (tmpBinaryImg.Pixels[a, v].isPixelNote = True) then
            begin
              if (b = 1) then
                isPixel := True;
              Inc(pixelCount);
            end;
          end;
          if (isPixel = True) and (pixelCount = FormMain.Config.RLConnectivity) then
            for v := 0 to lengths do
            begin
              tmpBinaryImg.Pixels[a, Stave[v].Pos.Y].IsPixelNote := True;
            end;
        end; // end for a :=
      end; // end if (Length(Stave
    end; // end with MusicNotesRows[z]
  end; // end for u := 1
end; // end for z
```

Gambar 4.10 *Source code repair* citra

Citra setelah *stave removal* ditampilkan pada gambar 4.11

Gambar 4.11 Citra setelah *stave removal*

4.2.6 Coarse Classification

Pada proses coarse classification ini terdapat beberapa proses, yang dapat dibagi menjadi 2 yaitu DetectMusicNotes dan ClassifyMusicNotes. Pada DetectMusicNotes terdapat langkah-langkah yaitu:

1). Proses pemeriksaan konektivitas piksel-piksel yang membentuk satu simbol pada citra, menggunakan CCL, seperti gambar 4.12.

```
procedure TMusicScore.CheckNeighboursPixel(var musicNote: TMusicNote;
  labelIndex, x, y: Integer; var binaryImg: TBinaryImage);
begin
  if (binaryImg.Pixels[x, y].isPixelNote = True) then
  begin
    if (binaryImg.Pixels[x, y].isLabeled = False) then
    begin
      if (musicNote.FullWindowRect.TopLeft.X >= x) then
        musicNote.FullWindowRect.TopLeft.X := x;
      if (musicNote.FullWindowRect.TopLeft.Y >= y) then
        musicNote.FullWindowRect.TopLeft.Y := y;
      if (musicNote.FullWindowRect.BottomRight.X <= x) then
        musicNote.FullWindowRect.BottomRight.X := x;
      if (musicNote.FullWindowRect.BottomRight.Y <= y) then
        musicNote.FullWindowRect.BottomRight.Y := y;
      Inc(musicNote.PixelCount);
      // Berikan label pada BinaryImage agar piksel yang sudah ditandai tidak diproses lagi.
      binaryImg.Pixels[x, y].isLabeled := True;
      binaryImg.Pixels[x, y].LabelIndex := labelIndex;
      // Cek 8 piksel di sekelilingnya
      // pojok kiri atas
      if ((x > 0) and (y > 0)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x-1, y-1, binaryImg);
      // tengah atas
      if ((y > 0)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x, y-1, binaryImg);
      // pojok kanan atas
      if ((x < binaryImg.Width-1) and (y > 0)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x+1, y-1, binaryImg);
      // kiri tengah
      if ((x > 0)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x-1, y, binaryImg);
      // kanan tengah
      if ((x < binaryImg.Width-1)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x+1, y, binaryImg);
      // pojok kiri bawah
      if ((x > 0) and (y < binaryImg.Height-1)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x-1, y+1, binaryImg);
      // tengah bawah
      if (y < binaryImg.Height-1) then
      CheckNeighboursPixel(musicNote, LabelIndex, x, y+1, binaryImg);
      // pojok kanan bawah
      if ((x < binaryImg.Width-1) and (y < binaryImg.Height-1)) then
      CheckNeighboursPixel(musicNote, LabelIndex, x+1, y+1, binaryImg);
    end;
```

Gambar 4.12 *Source code connected component labelling*

2). Penentuan kunci F, karena memerlukan penanganan khusus. Simbol kunci F mempunyai 2 titik yang terpisah dari simbol, sehingga bila dilakukan pengecekan konektivitas, 2 titik ini dianggap terpisah.

3). Pendeteksian garis birama

Sourcecode dari DetectMusicNotes secara lengkap ditampilkan pada gambar 4.13

```
function TMusicScore.DetectMusicNotes(var binaryImg: TBinaryImage): Boolean;
  function ValueInRange(val, minVal, maxVal: Integer): Boolean;
  begin
    Result := (val >= minVal) and (val <= maxVal);
  end;
  function Overlap(Rect1, Rect2: TRect): Boolean;
  var
    overlapX, overlapY: Boolean;
  begin
    overlapX := ValueInRange(Rect1.Left, Rect2.Left, Rect2.Right) or
      ValueInRange(Rect2.Left, Rect1.Left, Rect1.Right);
    overlapY := ValueInRange(Rect1.Top, Rect2.Top, Rect2.Bottom) or
      ValueInRange(Rect2.Top, Rect1.Top, Rect1.Bottom);
    Result := overlapX and overlapY;
  end;
var
  a, b, c, x, y, z, u, v, w, h, index: Integer;
  noteSizeRatio: Real;
  notePixelRatio, lengths: Real;
  barValue, val: Real;
  SUCCEED: Boolean;
  tempHorzProjection: array of Integer;
  maxHorz, x1, y1, x2, y2: Integer;
  upperY, lowerY, posY: Integer;
  maxStaveSpace: Integer;
  isRepaired, isIntersected: Boolean;
begin
  SUCCEED := False;
  try
    if (RowCount > 0) then
    begin
      // Untuk setiap MusicNotesRow
      barValue := 0.1;
      for z := 0 to RowCount-1 do
      begin
        index := 0;
        for x := 1 to binaryImg.Width-2 do
        begin
          for y := MusicNotesRows[z].UpperBound to MusicNotesRows[z].LowerBound do
          begin
            if (binaryImg.Pixels[x, y].isPixelNote = true) then
            begin
              if (binaryImg.Pixels[x, y].isLabeled = false) then
              begin
                Inc(index);
                SetLength(MusicNotesRows[z].MusicNotes, index);
```

```
        MusicNotesRows[z].MusicNoteCount := index;
        MusicNotesRows[z].MusicNotes[index-1] := TMusicNote.Create;
        Inc(MusicNotesRows[z].MusicNotes[index-1].PixelCount);
        CheckNeighboursPixel(MusicNotesRows[z].MusicNotes[index-1], index-1, x, y, binaryImg);
      end;
    end;
  end;
end; // end for x = 1
b := 0;
for a := 0 to MusicNotesRows[z].MusicNoteCount-1 do
begin
  // Hilangkan Music Note yang jumlah piksel non-backgroundnya sangat sedikit.
  // Music Note semacam ini dianggap sebagai noise
  c := a - b;
  if (MusicNotesRows[z].MusicNotes[c].PixelCount <= FormMain.Config.NoisePixelCount) then
  begin
    // Ubah binaryimage-nya
    for x := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Left to
      MusicNotesRows[z].MusicNotes[c].FullWindowRect.Right do
    begin
      for y := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Top to
        MusicNotesRows[z].MusicNotes[c].FullWindowRect.Bottom do
      begin
        binaryImg.Pixels[x, y].isPixelNote := False;
      end;
    end;
    // Singkirkan Music Note 'noise' ini
    if (c = MusicNotesRows[z].MusicNoteCount-1) then
    begin
      MusicNotesRows[z].MusicNoteCount := MusicNotesRows[z].MusicNoteCount - 1;
      SetLength(MusicNotesRows[z].MusicNotes, MusicNotesRows[z].MusicNoteCount);
      Break;
    end
    else
    begin
      MusicNotesRows[z].MusicNotes[c].Free;
      MusicNotesRows[z].MusicNotes[c] := nil;
      Move(MusicNotesRows[z].MusicNotes[c + 1], MusicNotesRows[z].MusicNotes[c],
        (Length(MusicNotesRows[z].MusicNotes) - c - 1) * SizeOf(TMusicNote) + 1);
      MusicNotesRows[z].MusicNoteCount := MusicNotesRows[z].MusicNoteCount - 1;
      SetLength(MusicNotesRows[z].MusicNotes, MusicNotesRows[z].MusicNoteCount);
    end;
    Inc(b);
  end;
end;
// Ini 'tweak' spesifik untuk kunci F yang notabene punya dua jendol pengganggu
// di sebelah kanan. Diasumsikan jendol2 itu adalah sebuah lingkaran
// atau mendekati bentuk lingkaran, sehingga width dan height dari window jendol2 tsb
// memiliki ukuran yang hampir sama (window dari sebuah lingkaran selalu berbentuk persegi).
// Dalam kondisi yang ideal, rasio width dan height bernilai 1, tetapi dalam kondisi yang
// sebenarnya rasio ditentukan antara 0.8 - 1.0. Selain itu rasio jumlah piksel non-background
// yang mengisi window dengan ukuran window, minimal 0.5;
b := 0;
for a := 0 to MusicNotesRows[z].MusicNoteCount-1 do
begin
  c := a - b;
  w := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Right -
```

```
  MusicNotesRows[z].MusicNotes[c].FullWindowRect.Left + 1;
h := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Bottom -
  MusicNotesRows[z].MusicNotes[c].FullWindowRect.Top + 1;
if (w > h) then
  noteSizeRatio := h / w
else
  noteSizeRatio := w / h;
// Rasio banyaknya piksel non background terhadap jumlah piksel pada Music Note
// secara keseluruhan / ukuran window Music Note
notePixelRatio := MusicNotesRows[z].MusicNotes[c].PixelCount / (w * h);
if ((noteSizeRatio >= FormMain.Config.FClefSizeRatio) and (noteSizeRatio <= 1.0)) and
  (notePixelRatio >= FormMain.Config.FClefPixelRatio) then
begin
  // Music Note ini dianggap sebagai si jendol. Pilih salah satu piksel
  x := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Left;
  y := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Bottom;
  // Cek piksel2 di sebelah kanan dari piksel [x,y] milik si jendol
  // Sekali bertemu dengan piksel non-background A, langsung cari Music Note dimana
  // piksel non-background A itu berada
  for u := x-1 downto 0 do
  begin
    if (binaryImg.Pixels[u, y].isPixelNote =  True) then
    begin
      Break;
    end;
  end;
  if (u > 0) then
  begin
    for v := 0 to MusicNotesRows[z].MusicNoteCount-1 do
    begin
      with MusicNotesRows[z].MusicNotes[v] do
      begin
        if (((u >= FullWindowRect.TopLeft.X) and (u < FullWindowRect.BottomRight.X)) and
          ((y >= FullWindowRect.TopLeft.Y) and (y < FullWindowRect.BottomRight.Y))) then
        begin
          // Gabungkan lebih dahulu kedua window (window si jendol n window Music Note kunci F)
          // yaitu dengan menggabungkan Rectangle keduanya
          if (FullWindowRect.Right < MusicNotesRows[z].MusicNotes[c].FullWindowRect.Right) then
            FullWindowRect.Right := MusicNotesRows[z].MusicNotes[c].FullWindowRect.Right;
          // Langsung tentukan tipe dari Music Note ini
          MusicNoteType := mnFClef;
          // Hapus window si jendol
          if (c = MusicNotesRows[z].MusicNoteCount-1) then
          begin
            MusicNotesRows[z].MusicNoteCount := MusicNotesRows[z].MusicNoteCount - 1;
            SetLength(MusicNotesRows[z].MusicNotes, MusicNotesRows[z].MusicNoteCount);
          end
          else
          begin
            MusicNotesRows[z].MusicNotes[c].Free;
            MusicNotesRows[z].MusicNotes[c] := nil;
            Move(MusicNotesRows[z].MusicNotes[c + 1], MusicNotesRows[z].MusicNotes[c],
              (Length(MusicNotesRows[z].MusicNotes) - c - 1) * SizeOf(TMusicNote) + 1);
            MusicNotesRows[z].MusicNoteCount := MusicNotesRows[z].MusicNoteCount - 1;
            SetLength(MusicNotesRows[z].MusicNotes, MusicNotesRows[z].MusicNoteCount);
          end;
          Inc(b);
          Break;
        end; // end if (u > FullWindowRect ...
```

```
// Tweak bila Music Note 'terpenggal', khusus untuk F Clef
isIntersected := False;
b := 0;
for a := 0 to MusicNotesRows[z].MusicNoteCount-2 do
begin
  c := a - b;
  // Cari Music Note  yang window-nya saling beririsan
  with (MusicNotesRows[z]) do
  begin
    for u := a + 1 to MusicNotesRows[z].MusicNoteCount-1 do
    begin
      v := u - b;
      isIntersected := Overlap(MusicNotes[c].FullWindowRect, MusicNotes[v].FullWindowRect);
      if (isIntersected) then
      begin
        // Update window yang baru
        if (MusicNotes[v].FullWindowRect.Left < MusicNotes[c].FullWindowRect.Left) then
          MusicNotes[c].FullWindowRect.Left := MusicNotes[v].FullWindowRect.Left;
        if (MusicNotes[v].FullWindowRect.Right > MusicNotes[c].FullWindowRect.Right) then
          MusicNotes[c].FullWindowRect.Right := MusicNotes[v].FullWindowRect.Right;
        if (MusicNotes[v].FullWindowRect.Top < MusicNotes[c].FullWindowRect.Top) then
          MusicNotes[c].FullWindowRect.Top := MusicNotes[v].FullWindowRect.Top;
        if (MusicNotes[v].FullWindowRect.Bottom > MusicNotes[c].FullWindowRect.Bottom) then
          MusicNotes[c].FullWindowRect.Bottom := MusicNotes[v].FullWindowRect.Bottom;
        if (MusicNotes[c].MusicNoteType = mnQuarterNoteDown) or
          (MusicNotes[c].MusicNoteType = mnQuarterNoteUp) then
          MusicNotes[c].CenterNoteHead := CenterPoint(MusicNotes[c].HeadWindowRect);
        MusicNotes[c].PixelCount := MusicNotes[c].PixelCount + MusicNotes[v].PixelCount;
        // Hapus
        if (v = MusicNotesRows[z].MusicNoteCount-1) then
        begin
          MusicNoteCount := MusicNoteCount - 1;
          SetLength(MusicNotes, MusicNoteCount);
        end
        else
        begin
          MusicNotes[v].Free;
          MusicNotes[v] := nil;
          Move(MusicNotes[v + 1], MusicNotes[v],
            (Length(MusicNotes) - v - 1) * SizeOf(TMusicNote) + 1);
          MusicNoteCount := MusicNoteCount - 1;
          SetLength(MusicNotes, MusicNoteCount);
        end;
        Inc(b);
      end;
    end;
  end;
end;
end; // end if (.. mnNone)
end; // end for a
for a := 0 to MusicNotesRows[z].MusicNoteCount-1 do
```

```
    begin
      if (MusicNotesRows[z].MusicNotes[a].MusicNoteType = mnNone) then
      begin
        // Cari Music Note tipe bar
        // Ciri-cirinya adalah perbandingan antara tinggi dan lebar window-nya sangat
        // berbeda jauh
        w := MusicNotesRows[z].MusicNotes[a].FullWindowRect.Right -
          MusicNotesRows[z].MusicNotes[a].FullWindowRect.Left + 1;
        h := MusicNotesRows[z].MusicNotes[a].FullWindowRect.Bottom -
          MusicNotesRows[z].MusicNotes[a].FullWindowRect.Top + 1;
        val := w / h;
        if (val <= FormMain.Config.BarSizeRatio) then
        begin
          MusicNotesRows[z].MusicNotes[a].MusicNoteType := mnBar;
        end;
      end;
    end;
  end;
end;
```

Gambar 4.13 *Sourcecode* DetectMusicNotes

Citra partitur setelah mengalami DetectMusicNotes ditampilkan pada gambar 4.14

Gambar 4.14 Citra setelah *DetectMusicNotes*

Pada proses ClassifyMusicNotes, proses-proses yang terdapat di dalamnya yaitu:

1). Perbandingan tiap simbol musik dalam citra dengan masing-masing *template*. Selanjutnya dilakukan pengenalan terhadap simbol.

2). Penentuan *head note* untuk mencari koordinat *center head note*.

3). Penentuan *center note*.

Source code dari ClassifyMusicNotes ditampilkan pada gambar 4.15

```
function TMusicScore.ClassifyMusicNotes(binaryImg: TBinaryImage): Boolean;
var
  a, b, w, z, x, y, n, m, index, stemWidthTolerate, stemMin, stemMax: Integer;
  SUCCEED: Boolean;
  wid, hei: Integer;
  tmpBmp1, tmpBmp2: TBitmap;
  tmpBinaryImg: TBinaryImage;
  clrBackground, clrNonBackground: TColor;
  templatePixel, notePixel: Boolean;
  matchedPixel, matchedPixelMax, matchedIndex: Integer;
  tempVertProjection, tempHorzProjection: array of Integer;
  stemCandidate: array of Boolean;
begin
  SUCCEED := True;
  clrBackground := binaryImg.ColorBg;
```

```
clrNonBackground := binaryImg.ColorNonBg;
  try
  if (RowCount > 0) then
  begin
    // Untuk setiap MusicNotesRow
    for z := 0 to RowCount-1 do
    begin
      for w := 0 to MusicNotesRows[z].MusicNoteCount-1 do
      begin
        if (MusicNotesRows[z].MusicNotes[w].MusicNoteType = mnNone) then
        begin
          // Cari Music Note tipe yang lain
          if (FormTemplate.NoteTemplates <> nil) then
          begin
            if (FormTemplate.NoteTemplates.NoteCount <= 0) then
            begin
           MessageDlg('Error mengklasifikasi Music Note. Template masih kosong', mtError, [mbOK],0);
              Result := not SUCCEED;
            end;
            // Ubah binary image Music Note menjadi bitmap
            tmpBmp1 := TBitmap.Create;
            tmpBmp1.PixelFormat := pf24bit;
            tmpBmp2 := TBitmap.Create;
            tmpBmp2.PixelFormat := pf24bit;
            with (MusicNotesRows[z].MusicNotes[w]) do
            begin
              tmpBmp1.Width := FullWindowRect.Right - FullWindowRect.Left + 1;
              tmpBmp1.Height := FullWindowRect.Bottom - FullWindowRect.Top + 1;
              m := 0;
              for x := FullWindowRect.Left to FullWindowRect.Right do
              begin
                n := 0;
                for y := FullWindowRect.Top to FullWindowRect.Bottom do
                begin
                  if (binaryImg.Pixels[x, y].isPixelNote = True) then
                    tmpBmp1.Canvas.Pixels[m, n] := clrNonBackground
                  else
                    tmpBmp1.Canvas.Pixels[m, n] := clrBackground;
                  Inc(n);
                end;
                Inc(m);
              end;
            end;
            tmpBinaryImg := TBinaryImage.Create;
            matchedPixelMax := -1;
            matchedIndex := -1;
            for a := 0 to FormTemplate.NoteTemplates.NoteCount-1 do
            begin
              wid := FormTemplate.NoteTemplates.NoteTemplateCollection[a].Template.Width;
              hei := FormTemplate.NoteTemplates.NoteTemplateCollection[a].Template.Height;
              tmpBmp2 := TBitmap.Create;
              tmpBmp2.Width := wid;
              tmpBmp2.Height := hei;
              tmpBmp2.PixelFormat := pf24bit;
              tmpBmp2.Canvas.StretchDraw(Rect(0, 0, wid, hei), tmpBmp1);
tmpBmp2 := ReColor(tmpBmp2, FormMain.Config.ColorThreshold, clrBackground, clrNonBackground);
              tmpBinaryImg := TBinaryImage.Create(tmpBmp2, clrBackground, clrNonBackground);
              // Lakukan perbandingan per piksel antara Music Note dengan tiap template.
              // Jadikan template sebagai acuan perbandingan
```

```
              matchedPixel := 0;
              for y := 0 to hei-1 do
              begin
                for x := 0 to wid-1 do
                begin
              templatePixel :=
              FormTemplate.NoteTemplates.NoteTemplateCollection[a].Template.Pixels[x, y].isPixelNote;
                  notePixel := tmpBinaryImg.Pixels[x, y].isPixelNote;
                  if (templatePixel = True) then
                  begin
                    if (templatePixel = notePixel) then
                      Inc(matchedPixel);
                  end;
                end;
              end;
              matchedPixel := Round(
(matchedPixel / FormTemplate.NoteTemplates.NoteTemplateCollection[a].Template.PixelCount) *100);
              if (matchedPixel > matchedPixelMax) then
              begin
                matchedPixelMax := matchedPixel;
                matchedIndex := a;
              end;
            end; // end for a
            if (matchedIndex = -1) then
              MusicNotesRows[z].MusicNotes[w].MusicNoteType := mnNone
            else
              MusicNotesRows[z].MusicNotes[w].MusicNoteType :=
                FormTemplate.NoteTemplates.NoteTemplateCollection[matchedIndex].NoteType;
            // Untuk quarter note up dan down, cari head dari masing-masing Music Note
            if (MusicNotesRows[z].MusicNotes[w].MusicNoteType = mnQuarterNoteDown) or
              (MusicNotesRows[z].MusicNotes[w].MusicNoteType = mnQuarterNoteUp) then
            begin
              // Buat Vertical dan Horizontal Projection untuk tiap Music Note
              wid := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Right -
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Left + 1;
              hei := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom -
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top + 1;
              SetLength(tempVertProjection, wid);
              SetLength(tempHorzProjection, hei);
              SetLength(stemCandidate, wid);
              for a := 0 to wid-1 do
              begin
                tempVertProjection[a] := 0;
                stemCandidate[a] := False;
              end;
              for b := 0 to hei-1 do
              begin
                tempHorzProjection[b] := 0;
              end;
              a := 0;
              for x := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Left to
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Right do
              begin
                for y := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top to
                  MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom do
                begin
                  if (binaryImg.Pixels[x, y].isPixelNote) then
```

```
    begin
      Inc(tempVertProjection[a]);
    end;
  end;
  Inc(a);
end;
b := 0;
for y := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top to
  MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom do
begin
  for x := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Left to
    MusicNotesRows[z].MusicNotes[w].FullWindowRect.Right do
  begin
    if (binaryImg.Pixels[x, y].isPixelNote) then
    begin
      Inc(tempHorzProjection[b]);
    end;
  end;
  Inc(b);
end;
m := Trunc(hei / 2);
// n = lebar dari stem
n := 0;
for a := 0 to wid-1 do
begin
  if (tempVertProjection[a] > m) then
  begin
    stemCandidate[a] := True;
    Inc(n);
  end;
end;
stemMin := 1;
if (n <= 1) then
begin
  stemWidthTolerate := 1;
  stemMax := n + 2 * stemWidthTolerate;
end
else if (n = 2) then
begin
  stemWidthTolerate := 1;
  stemMax := n + 2 * stemWidthTolerate;
end
else if (n >= 3) then
begin
  stemWidthTolerate := Floor(n / 2);
  stemMax := n + 2 * stemWidthTolerate;
end;
if (MusicNotesRows[z].MusicNotes[w].MusicNoteType = mnQuarterNoteDown) then
begin
  a := Length(tempHorzProjection)-1;
  for y := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom downto
    MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top do
  begin
    if not((tempHorzProjection[a] >= stemMin) and (tempHorzProjection[a] <= stemMax))
      and (tempHorzProjection[a] > 0) then
    begin
      MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Top :=
        MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top;
```

```
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Left :=
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Left;
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Bottom := y;
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Right :=
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Right;
              Break;
            end;
            a := a - 1;
          end;
        end // end if (.. = mnQuarterNoteDown)
        else if (MusicNotesRows[z].MusicNotes[w].MusicNoteType = mnQuarterNoteUp) then
        begin
          a := 0;
          for y := MusicNotesRows[z].MusicNotes[w].FullWindowRect.Top to
            MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom do
          begin
            if not((tempHorzProjection[a] >= stemMin) and (tempHorzProjection[a] <= stemMax))
              and (tempHorzProjection[a] > 0) then
            begin
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Top := y;
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Left :=
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Left;
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Bottom :=
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Bottom;
              MusicNotesRows[z].MusicNotes[w].HeadWindowRect.Right :=
                MusicNotesRows[z].MusicNotes[w].FullWindowRect.Right;
              Break;
            end;
            Inc(a);
          end;
        end;
        MusicNotesRows[z].MusicNotes[w].CenterNoteHead := CenterPoint(
          MusicNotesRows[z].MusicNotes[w].HeadWindowRect);
      end;
      tmpBmp1.Free;
      tmpBmp2.Free;
      tmpBinaryImg.Free;
    end; // end if (FormTemplate.NoteTemplates <> nil) then
  end; // end if (<> mnNone)
end;
```

Gambar 4.15 *Sourcecode* ClassifyMusicNotes

Selanjutnya diperlukan *source code* untuk menampilkan *symbol window* pada *form* dan menghilangkan *staff* dari not. *Source code* yang diperlukan ditampilkan pada gambar 4.16

```
// Tampilkan window Notehead
        if (MusicNoteType = mnQuarterNoteUp) or (MusicNoteType = mnQuarterNoteDown) then
        begin
          // Hapus stem / tongkat note
          if (not Config.ShowQuarterNoteStem) then
            if (MusicNoteType = mnQuarterNoteDown) then
            begin
        rectNote := Rect(FullWindowRect.Left-1, HeadWindowRect.Bottom, FullWindowRect.Right+1,
              FullWindowRect.Bottom+1);
            ClassifyBitmap.Canvas.FillRect(rectNote);
            end
            else if (MusicNoteType = mnQuarterNoteUp) then
            begin
         rectNote := Rect(FullWindowRect.Left-1, FullWindowRect.Top-1, FullWindowRect.Right+1,
              HeadWindowRect.Top);
            ClassifyBitmap.Canvas.FillRect(rectNote);
            end;
          // garis atas
       ClassifyBitmap.Canvas.MoveTo(HeadWindowRect.TopLeft.X, HeadWindowRect.TopLeft.Y);
       ClassifyBitmap.Canvas.LineTo(HeadWindowRect.BottomRight.X, HeadWindowRect.TopLeft.Y);
          // garis kiri
       ClassifyBitmap.Canvas.MoveTo(HeadWindowRect.TopLeft.X, HeadWindowRect.TopLeft.Y);
       ClassifyBitmap.Canvas.LineTo(HeadWindowRect.TopLeft.X, HeadWindowRect.BottomRight.Y);
          // garis kanan
    ClassifyBitmap.Canvas.MoveTo(HeadWindowRect.BottomRight.X, HeadWindowRect.TopLeft.Y);
    ClassifyBitmap.Canvas.LineTo(HeadWindowRect.BottomRight.X, HeadWindowRect.BottomRight.Y);
          // garis bawah
    ClassifyBitmap.Canvas.MoveTo(HeadWindowRect.TopLeft.X, HeadWindowRect.BottomRight.Y);
    ClassifyBitmap.Canvas.LineTo(HeadWindowRect.BottomRight.X, HeadWindowRect.BottomRight.Y);
        end;
```

Gambar 4.16 *Sourcecode* untuk menampilkan *symbol window* dan menghilangkan *staff* pada not

Citra partitur setelah proses ClassifyMusicNotes tampak pada gambar 4.17

Gambar 4.17 Citra setelah ClassifyMusicNotes

4.2.7 Fine Classification

Pada fine classification terdapat 2 proses, yaitu:

1). Penghitungan rata-rata jarak semua not. Proses ini dilakukan untuk mengetahui golongan suara yang sesuai bagi simbol musik yang tertera pada partitur.

2). *Music syntax check*, yaitu menggolongkan simbol-simbol musik ke golongan suara yang sesuai dengan threshold yang didapat.

Sourcecode dari *fine classification* ditampilkan pada gambar 4.18

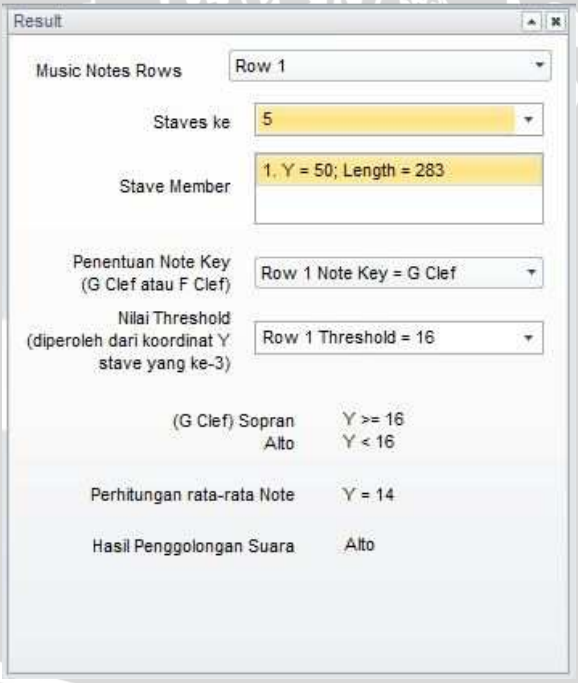
```
procedure TFormMain.ButtonRunProcessClick(Sender: TObject);
var
  a, b, lastStavePos, distance, totalDistanceAll, threshold, tolerateVal, musicNoteTotal, middleY: Integer;
  totalDistancePerRow: array of Integer; averageDistance, averageThreshold: Integer;
  NoteKeyG, NoteKeyF: Integer;
begin
  if (Config.ProcessAllAtOnce) then
  begin
    ButtonDetectStavesClick(nil); ButtonRemoveStavesClick(nil); ButtonDetectNotesClick(nil);
    ButtonClassifyNotesClick(nil);
  end;
  if (not ClassifyIndicator) then
    Exit;
  // Hitung
  totalDistanceAll := 0; musicNoteTotal := 0; tolerateVal := 1;
  SetLength(totalDistancePerRow, MusicScore.RowCount);
  for a := 0 to MusicScore.RowCount-1 do
  begin
    lastStavePos := MusicScore.MusicNotesRows[a].Staves[5].Stave[0].Pos.Y;
    totalDistancePerRow[a] := 0;
    for b := 0 to MusicScore.MusicNotesRows[a].MusicNoteCount-1 do
    begin
      with (MusicScore.MusicNotesRows[a]) do
      begin
        if (MusicNotes[b].MusicNoteType = mnGClef) then
        begin
          KeyNotes := mnGClef;
          Break;
        end
        else if (MusicNotes[b].MusicNoteType = mnFClef) then
        begin
          KeyNotes := mnFClef;
          Break;
        end;
      end;
    end;
    for b := 0 to MusicScore.MusicNotesRows[a].MusicNoteCount-1 do
    begin
      with (MusicScore.MusicNotesRows[a]) do
      begin
        if (MusicNotes[b].MusicNoteType = mnQuarterNoteDown) or
          (MusicNotes[b].MusicNoteType = mnQuarterNoteUp) then
        begin
          distance := lastStavePos - MusicNotes[b].CenterNoteHead.Y;
          totalDistanceAll := totalDistanceAll + distance;
          totalDistancePerRow[a] := totalDistancePerRow[a] + distance;
          Inc(musicNoteTotal);
        end;
      end;
    end;
  CBMusicNotesRows.Items.Clear;
  CBThreshold.Items.Clear;
  CBNoteKey.Items.Clear;
  NoteKeyG := 0;
  NoteKeyF := 0;
  LabelVoiceType1.Lines.Clear;
  LabelVoiceType2.Lines.Clear;
```

```
  averageThreshold := 0;
  for a := 0 to MusicScore.RowCount-1 do
  begin
    CBMusicNotesRows.Items.Add('Row ' + IntToStr(a+1));
    middleY := Round(MusicScore.MusicNotesRows[a].Staves[5].NumberOfStaveCandidate / 2);
    lastStavePos := MusicScore.MusicNotesRows[a].Staves[5].Stave[middleY].Pos.Y;
    middleY := Round(MusicScore.MusicNotesRows[a].Staves[3].NumberOfStaveCandidate / 2);
    threshold := lastStavePos - MusicScore.MusicNotesRows[a].Staves[3].Stave[middleY].Pos.Y;
    averageThreshold := averageThreshold + threshold;
    CBThreshold.Items.Add('Row ' + IntToStr(a+1) + ' Threshold = ' + IntToStr(threshold));
    if (MusicScore.MusicNotesRows[a].KeyNotes = mnGClef) then
    begin
      CBNoteKey.Items.Add('Row ' + IntToStr(a+1) + ' Note Key = G Clef');
      Inc(NoteKeyG);
    end
    else
    begin
      CBNoteKey.Items.Add('Row ' + IntToStr(a+1) + ' Note Key = F Clef');
      Inc(NoteKeyF);
    end;
  end;
  if (CBThreshold.Items.Count > 0) then
    CBThreshold.ItemIndex := 0;
  if (CBNoteKey.Items.Count > 0) then
    CBNoteKey.ItemIndex := 0;
  if MusicScore.RowCount > 0 then
  begin
    CBMusicNotesRows.ItemIndex := 0;
    CBMusicNotesRowsChange(nil);
  end;
  // Hitung Y rata-rata
  averageDistance := Round(totalDistanceAll / musicNoteTotal);
  LabelNoteValue.Caption := 'Y = '+IntToStr(averageDistance);
   if MusicScore.RowCount > 0 then
    averageThreshold := Round(averageThreshold / MusicScore.RowCount);
  LabelVoiceType2.Lines.Add('Y >= ' + IntToStr(averageThreshold));
  LabelVoiceType2.Lines.Add('Y < ' + IntToStr(averageThreshold));
  if (NoteKeyG >= NoteKeyF) then
  begin
    LabelVoiceType1.Lines.Add('(G Clef) Sopran');
    LabelVoiceType1.Lines.Add('Alto');
    if (averageDistance >= averageThreshold) then
      LabelVoiceResult.Caption := 'Sopran'
    else LabelVoiceResult.Caption := 'Alto';
  end
  else begin
    LabelVoiceType1.Lines.Add('(F Clef) Tenor');
    LabelVoiceType1.Lines.Add('Bass');
    if (averageDistance >= averageThreshold) then
      LabelVoiceResult.Caption := 'Tenor'
    else
      LabelVoiceResult.Caption := 'Bass';
  end;
  PanelResult.Show; ButtonResult.Enabled := True;
end;
```

Gambar 4.18 *Sourcecode fine classification*

Citra hasil fine classification dan music syntax check ditampilkan pada gambar 4.19 dan 4.20

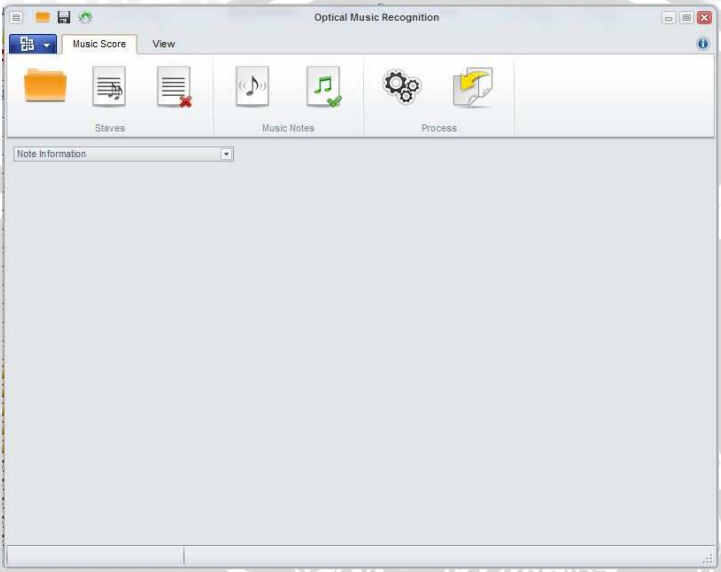
Gambar 4.19 Citra hasil *fine classification*



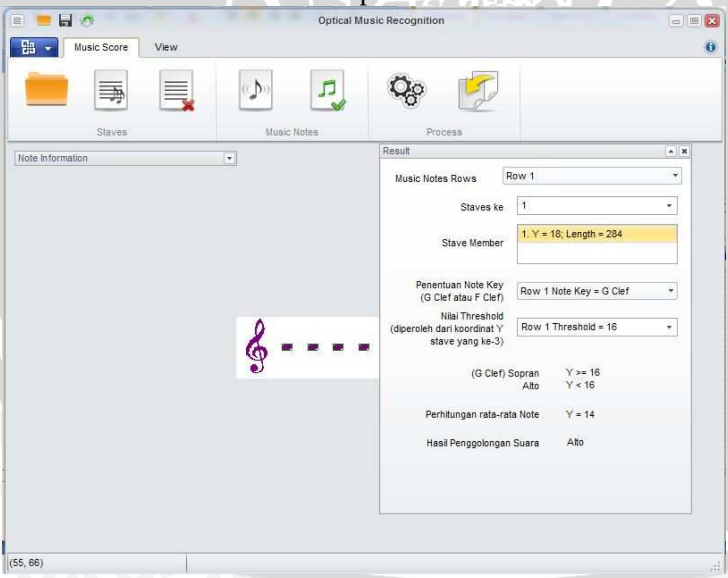
Gambar 4.20 Citra hasil *music syntax check*

4.3 Implementasi *Interface*

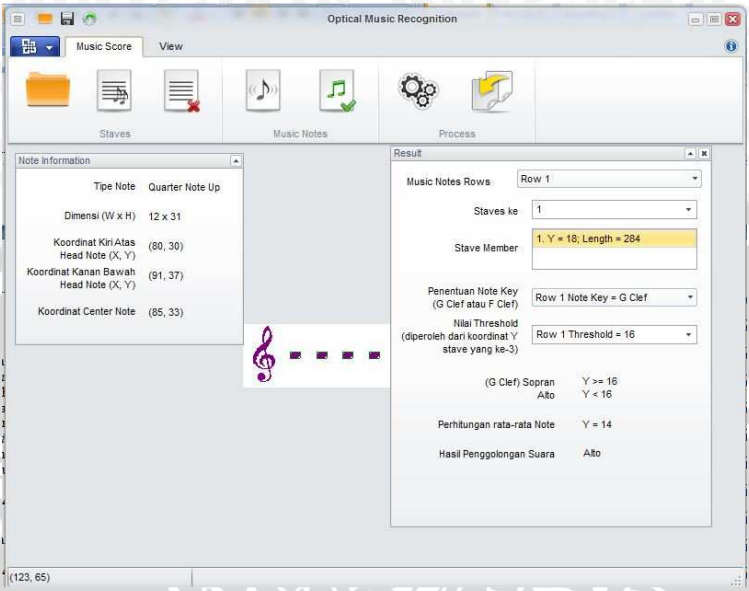
Perancangan *interface* model perangkat lunak pada ditunjukkan pada Gambar 4.21, Gambar 4.22, dan Gambar 4.23



Gambar 4.21 Tampilan *Form Utama*



Gambar 4.22 Tampilan hasil penggolongan suara



Gambar 4.23 Tampilan informasi simbol musik

Pada *user interface*, terdapat 2 bagian *Tab*. *Tab* pertama terdiri dari *Panel* yang berisi citra masukan partitur dan hasil penggolongan suara, dan *Tab* kedua berisi informasi lain-lain. Pada *Tab* pertama terdapat 7 tombol proses, yang terdiri dari; tombol *Open Image* untuk membuka dan *mengenhance image*, tombol *Detect Staves* untuk mendeteksi *staves*, tombol *Remove Staves* untuk menghilangkan *staves*, tombol Detect Music Notes dan tombol Classify Music Notes untuk memperoleh informasi simbol, serta tombol Run Process untuk memperoleh hasil penggolongan suara dan menampilkan informasi.

4.4 Implementasi Uji Coba

Pada subbab ini akan dilakukan pembahasan mengenai pengujian yang dilakukan pada sistem dan hasil yang dikeluarkan sistem.

4.4.1 Evaluasi Hasil

Uji coba diterapkan pada 100 citra partitur. Citra diambil dari 5 partitur vokal yang berbeda. Tiap partitur diambil 5 potongan citra dari masing-masing golongan suara, yaitu alto, bass, sopran, dan tenor.

Berikut merupakan hasil uji coba terhadap 100 citra partitur.

Tabel 4.1 Hasil uji coba perangkat lunak

Citra Partitur	*Stave* Terbawah	Penggolongan Kunci	*Threshold* Suara	Jarak Rata-rata	Golongan Suara
1	y = 50	G	sopran y>=16 alto y<16	14	Alto
2	y = 51	G	sopran y>=16 alto y<16	3	Alto
3	y = 52	G	sopran y>=16 alto y<16	16	Sopran
4	y = 50	G	sopran y>=16 alto y<16	5	Alto
5	y = 53	G	sopran y>=16 alto y<16	19	Sopran
6	y = 42	F	tenor y>=16 bass y<16	13	Bass
7	y = 44	F	tenor y>=16 bass y<16	8	Bass
8	y = 40	F	tenor y>=16 bass y<16	15	Bass
9	y = 43	F	tenor y>=16 bass y<16	11	Bass
10	y = 41	F	tenor y>=16 bass y<16	6	Bass
11	y = 50	G	sopran y>=16 alto y<16	18	Sopran
12	y = 54	G	sopran y>=16 alto y<16	4	Alto
13	y = 50	G	sopran y>=16 alto y<16	20	Sopran
14	y = 50	G	sopran y>=16 alto y<16	25	Sopran
15	y = 52	G	sopran y>=16 alto y<16	28	Sopran
16	y = 44	F	tenor y>=16 bass y<16	30	Tenor
17	y = 42	F	tenor y>=16 bass y<16	19	Tenor
18	y = 46	F	tenor y>=16 bass y<16	24	Tenor
19	y = 43	F	tenor y>=16 bass y<16	32	Tenor
20	y = 44	F	tenor y>=16 bass y<16	29	Tenor

21	y = 50	G	sopran y>=16 alto y<16	10	Alto
22	y = 51	G	sopran y>=16 alto y<16	6	Alto
23	y = 49	G	sopran y>=16 alto y<16	4	Alto
24	y = 54	G	sopran y>=16 alto y<16	10	Alto
25	y = 53	G	sopran y>=16 alto y<16	13	Alto
26	y = 42	F	tenor y>=16 bass y<16	12	Bass
27	y = 45	F	tenor y>=16 bass y<16	14	Bass
28	y = 43	F	tenor y>=16 bass y<16	6	Bass
29	y = 47	F	tenor y>=16 bass y<16	17	Tenor
30	y = 43	F	tenor y>=16 bass y<16	12	Bass
31	y = 53	G	sopran y>=16 alto y<16	23	Sopran
32	y = 51	G	sopran y>=16 alto y<16	19	Sopran
33	y = 51	G	sopran y>=16 alto y<16	25	Sopran
34	y = 49	G	sopran y>=16 alto y<16	21	Sopran
35	y = 52	G	sopran y>=16 alto y<16	18	Sopran
36	y = 40	F	tenor y>=16 bass y<16	30	Tenor
37	y = 46	F	tenor y>=16 bass y<16	25	Tenor
38	y = 43	F	tenor y>=16 bass y<16	22	Tenor
39	y = 45	F	tenor y>=16 bass y<16	32	Tenor
40	y = 45	F	tenor y>=16 bass y<16	24	Tenor
41	y = 50	G	sopran y>=16 alto y<16	3	Alto
42	y = 49	G	sopran y>=16 alto y<16	1	Alto

43	y = 52	G	sopran y>=16 alto y<16	6	Alto
44	y = 51	G	sopran y>=16 alto y<16	4	Alto
45	y = 52	G	sopran y>=16 alto y<16	3	Alto
46	y = 45	F	tenor y>=16 bass y<16	14	Bass
47	y = 46	F	tenor y>=16 bass y<16	17	Tenor
48	y = 43	F	tenor y>=16 bass y<16	6	Bass
49	y = 41	F	tenor y>=16 bass y<16	4	Bass
50	y = 45	F	tenor y>=16 bass y<16	6	Bass
51	y = 55	G	sopran y>=16 alto y<16	15	Alto
52	y = 53	G	sopran y>=16 alto y<16	18	Sopran
53	y = 57	G	sopran y>=16 alto y<16	16	Sopran
54	y = 53	G	sopran y>=16 alto y<16	19	Sopran
55	y = 53	G	sopran y>=16 alto y<16	14	Alto
56	y = 51	F	tenor y>=16 bass y<16	32	Tenor
57	y = 44	F	tenor y>=16 bass y<16	25	Tenor
58	y = 50	F	tenor y>=16 bass y<16	26	Tenor
59	y = 46	F	tenor y>=16 bass y<16	20	Tenor
60	y = 46	F	tenor y>=16 bass y<16	20	Tenor
61	y = 51	G	sopran y>=16 alto y<16	8	Alto
62	y = 52	G	sopran y>=16 alto y<16	1	Alto
63	y = 53	G	sopran y>=16 alto y<16	12	Alto
64	y = 51	G	sopran y>=16 alto y<16	7	Alto

65	y = 54	G	sopran y>=16 alto y<16	13	Alto
66	y = 43	F	tenor y>=16 bass y<16	11	Bass
67	y = 47	F	tenor y>=16 bass y<16	15	Bass
68	y = 50	F	tenor y>=16 bass y<16	12	Bass
69	y = 46	F	tenor y>=16 bass y<16	9	Bass
70	y = 45	F	tenor y>=16 bass y<16	7	Bass
71	y = 51	G	sopran y>=16 alto y<16	26	Sopran
72	y = 53	G	sopran y>=16 alto y<16	18	Sopran
73	y = 51	G	sopran y>=16 alto y<16	16	Sopran
74	y = 52	G	sopran y>=16 alto y<16	25	Sopran
75	y = 54	G	sopran y>=16 alto y<16	27	Sopran
76	y = 47	F	tenor y>=16 bass y<16	28	Tenor
77	y = 45	F	tenor y>=16 bass y<16	20	Tenor
78	y = 47	F	tenor y>=16 bass y<16	28	Tenor
79	y = 41	F	tenor y>=16 bass y<16	26	Tenor
80	y = 57	F	tenor y>=16 bass y<16	20	Tenor
81	y = 50	G	sopran y>=16 alto y<16	13	Alto
82	y = 54	G	sopran y>=16 alto y<16	9	Alto
83	y = 51	G	sopran y>=16 alto y<16	1	Alto
84	y = 54	G	tenor y>=16 bass y<16	6	Alto
85	y = 52	G	tenor y>=16 bass y<16	12	Alto
86	y = 41	F	tenor y>=16 bass y<16	18	Tenor

87	y = 46	F	tenor y>=16 bass y<16	9	Bass
88	y = 45	F	tenor y>=16 bass y<16	9	Bass
89	y = 45	F	tenor y>=16 bass y<16	15	Bass
90	y = 46	F	tenor y>=16 bass y<16	16	Tenor
91	y = 52	G	sopran y>=16 alto y<16	12	Alto
92	y = 52	G	sopran y>=16 alto y<16	22	Sopran
93	y = 53	G	sopran y>=16 alto y<16	17	Sopran
94	y = 52	G	sopran y>=16 alto y<16	28	Sopran
95	y = 51	G	sopran y>=16 alto y<16	20	Sopran
96	y = 47	F	tenor y>=16 bass y<16	18	Tenor
97	y = 47	F	tenor y>=16 bass y<16	34	Tenor
98	y = 45	F	tenor y>=16 bass y<16	26	Tenor
99	y = 46	F	tenor y>=16 bass y<16	32	Tenor
100	y = 49	F	tenor y>=16 bass y<16	24	Tenor

Kolom pertama pada tabel digunakan untuk mencatat indeks citra partitur yang digunakan untuk pengujian. Kolom kedua digunakan untuk mencatat posisi koordinat y *stave* ke 5. Kolom ketiga untuk mencatat hasil pembacaan kunci yang digunakan. *Threshold* pada kolom keempat ditentukan dari posisi koordinat y *stave* ke 5 dikurangi dengan posisi *stave* y ke 3. Nilai yang diperoleh akan digunakan sebagai nilai threshold untuk menentukan golongan suara. Kolom kelima digunakan untuk mencatat nilai rata-rata jarak not pada citra. Kolom terakhir digunakan untuk mencatat golongan suara yang diperoleh dari hasil pada kolom ke 5 dibandingkan dengan kolom ke 4.

4.4.2 Analisa Hasil Uji Coba

Untuk menghitung akurasi dari hasil penggolongan suara pada citra partitur, maka dilakukan perbandingan antara citra partitur yang telah diketahui golongan suaranya, dengan hasil keluaran dari perangkat lunak. Bila cocok maka dianggap valid, dan bila tidak cocok dianggap invalid.

Rumus yang digunakan yaitu :

$$\text{Persentase akurasi} = \frac{jumlah\ data\ valid}{jumlah\ percobaan} * 100\% \quad (4.1)$$

Dari perhitungan tabel 4.1, didapatkan hasil akurasi yang tertera pada tabel 4.2

Tabel 4.2 Validitas hasil uji coba

Citra Partitur	Golongan Suara	Hasil Perhitungan	Hasil
1	Alto	Alto	Valid
2	Alto	Alto	Valid
3	Alto	Sopran	Invalid
4	Alto	Alto	Valid
5	Alto	Sopran	Invalid
6	Bass	Bass	Valid
7	Bass	Bass	Valid
8	Bass	Bass	Valid
9	Bass	Bass	Valid
10	Bass	Bass	Valid
11	Sopran	Sopran	Valid
12	Sopran	Alto	Invalid
13	Sopran	Sopran	Valid
14	Sopran	Sopran	Valid
15	Sopran	Sopran	Valid
16	Tenor	Tenor	Valid
17	Tenor	Tenor	Valid
18	Tenor	Tenor	Valid
19	Tenor	Tenor	Valid
20	Tenor	Tenor	Valid
21	Alto	Alto	Valid
22	Alto	Alto	Valid
23	Alto	Alto	Valid
24	Alto	Alto	Valid
25	Alto	Alto	Valid
26	Bass	Bass	Valid
27	Bass	Bass	Valid
28	Bass	Bass	Valid

29	Bass	Tenor	Invalid
30	Bass	Bass	Valid
31	Sopran	Sopran	Valid
32	Sopran	Sopran	Valid
33	Sopran	Sopran	Valid
34	Sopran	Sopran	Valid
35	Sopran	Sopran	Valid
36	Tenor	Tenor	Valid
37	Tenor	Tenor	Valid
38	Tenor	Tenor	Valid
39	Tenor	Tenor	Valid
40	Tenor	Tenor	Valid
41	Alto	Alto	Valid
42	Alto	Alto	Valid
43	Alto	Alto	Valid
44	Alto	Alto	Valid
45	Alto	Alto	Valid
46	Bass	Bass	Valid
47	Bass	Tenor	Invalid
48	Bass	Bass	Valid
49	Bass	Bass	Valid
50	Bass	Bass	Valid
51	Sopran	Alto	Invalid
52	Sopran	Sopran	Valid
53	Sopran	Sopran	Valid
54	Sopran	Sopran	Valid
55	Sopran	Alto	Invalid
56	Tenor	Tenor	Valid
57	Tenor	Tenor	Valid
58	Tenor	Tenor	Valid
59	Tenor	Tenor	Valid
60	Tenor	Tenor	Valid
61	Alto	Alto	Valid
62	Alto	Alto	Valid
63	Alto	Alto	Valid
64	Alto	Alto	Valid
65	Alto	Alto	Valid
66	Bass	Bass	Valid
67	Bass	Bass	Valid
68	Bass	Bass	Valid
69	Bass	Bass	Valid
70	Bass	Bass	Valid
71	Sopran	Sopran	Valid

72	Sopran	Sopran	Valid
73	Sopran	Sopran	Valid
74	Sopran	Sopran	Valid
75	Sopran	Sopran	Valid
76	Tenor	Tenor	Valid
77	Tenor	Tenor	Valid
78	Tenor	Tenor	Valid
79	Tenor	Tenor	Valid
80	Tenor	Tenor	Valid
81	Alto	Alto	Valid
82	Alto	Alto	Valid
83	Alto	Alto	Valid
84	Alto	Alto	Valid
85	Alto	Alto	Valid
86	Bass	Tenor	Invalid
87	Bass	Bass	Valid
88	Bass	Bass	Valid
89	Bass	Bass	Valid
90	Bass	Tenor	Invalid
91	Sopran	Alto	Invalid
92	Sopran	Sopran	Valid
93	Sopran	Sopran	Valid
94	Sopran	Sopran	Valid
95	Sopran	Sopran	Valid
96	Tenor	Tenor	Valid
97	Tenor	Tenor	Valid
98	Tenor	Tenor	Valid
99	Tenor	Tenor	Valid
100	Tenor	Tenor	Valid

Kolom pertama digunakan untuk mencatat indeks data pengujian. Kolom kedua untuk mencatat golongan suara yang telah diketahui dari citra partitur asli. Kolom ketiga untuk mencatat hasil penggolongan suara yang diperoleh dari perangkat lunak. Hasil dari kolom kedua dan ketiga dibandingkan. Bila hasilnya sama maka data dianggap valid, bila tidak sama dianggap invalid. Hasil yang diperoleh dicatat pada kolom keempat.

Dari tabel 4.2 diperoleh jumlah data dengan rincian sebagai berikut :

a). Sopran dengan hasil valid sebanyak 21 dari 25 data uji coba.
b). Alto dengan hasil valid sebanyak 23 dari 25 data uji coba.
c). Tenor dengan hasil valid sebanyak 25 dari 25 data uji coba.
d). Bass dengan hasil valid sebanyak 21 dari 25 data uji coba.

Adanya data invalid pada hasil uji coba disebabkan pada data citra uji terdapat not-not dengan nada yang terlalu tinggi atau terlalu rendah dibandingkan dengan nilai threshold yang telah ditentukan sebelumnya untuk masing-masing golongan suara.

Dengan menggunakan rumus 4.1 maka didapatkan nilai persentase akurasi masing-masing golongan suara sebagai berikut:

a). Persentase akurasi sopran = $\frac{21}{25} * 100\% = 84\%$

b). Persentase akurasi alto = $\frac{23}{25} * 100\% = 96\%$

c). Persentase akurasi tenor = $\frac{25}{25} * 100\% = 100\%$

d). Persentase akurasi bass = $\frac{21}{25} * 100\% = 84\%$

.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil pembahasan dan analisis yang telah dilakukan pada bab sebelumnya, dapat disimpulkan bahwa:

1. Telah berhasil diimplementasikan modifikasi algoritma *Kim's Optical Music Recognition* dalam model penggolongan suara pada citra partitur, dan tersimulasikan dalam perangkat lunak yang telah dibangun.
2. Persentase keakuratan perangkat lunak penggolongan suara manusia dalam citra partitur pada percobaan sebanyak 25 data untuk masing-masing golongan suara yaitu sopran sebesar 84%, alto sebesar 96%, tenor sebesar 100%, dan bass sebesar 84%. Perbedaan persentase disebabkan pada data uji coba, terdapat not-not yang melebihi nilai threshold pada golongan suara sopran, alto, dan bass.

5.2 Saran

Beberapa saran lebih lanjut yang dapat diberikan adalah :

1. Menguji kecepatan dari penggolongan suara manusia pada citra partitur dengan menggunakan modifikasi algoritma *Kim's Optical Music Recognition*.
2. Memperbanyak simbol musik yang dapat dikenali dari model perangkat lunak yang telah dibuat, sehingga didapatkan hasil penggolongan yang lebih valid.

DAFTAR PUSTAKA

Achmad, B dan Firdausy, K. 2005. *Teknik Pengolahan Citra Digital Menggunakan Delphi*. Ardhi Publishing. Yogyakarta

Ahmad, Usman. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya*. Graha Ilmu. Yogyakarta

Feldman, D. 2007. *Apa Gajah Bisa Lompat dan Apa-Mengapa Lainnya*. PT Gramedia Pustaka Utama. Jakarta

Gonzales, R. dan Woods, R. 1992. *Digital Image Processing*. Addison-Wesley Publishing Company. UK

Kamien, Roger. 1988. *Music An Appreciation*. McGraw-Hill Book Company. United States of America

Knauss, Harold P. 2003. *Ilmu Pengetahuan Populer*. Grolier International Inc. Jakarta

Kim, W.J.; Chung M. J.; Bien, Z. 1987. *Recognition System for Printed Music Score*. Dept. Of EE, KAIST. Seoul

Redixta, Tim. 2007. *Ensiklopedi Ilmu Pengetahuan Alam Fisika*. Aneka Ilmu. Semarang.

Simanungkalit, N. 2008. *Teknik Vokal Paduan Suara*. PT Gramedia Pustaka Utama. Jakarta

Temperley, David. 2001. *The Cognition of Basic Musical Structure*. The Massachussets Institute of Technology Press. London

Thai, R. 2003. *Fingerprint Image Enhancement and Minutiae Extraction*. The University Western Australia. Australia

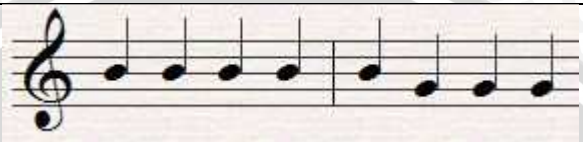
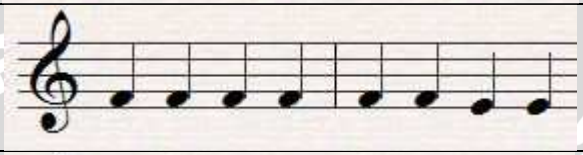
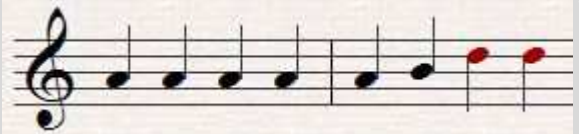
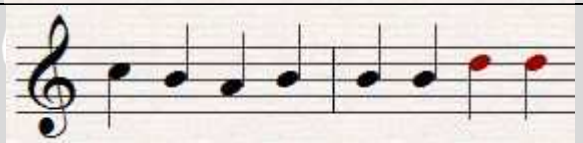
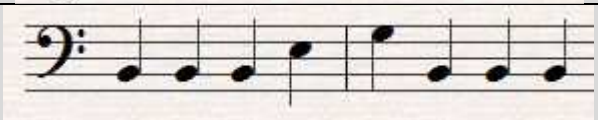
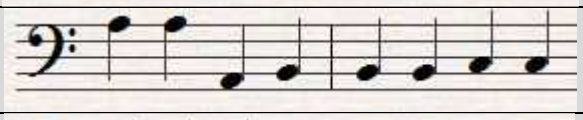
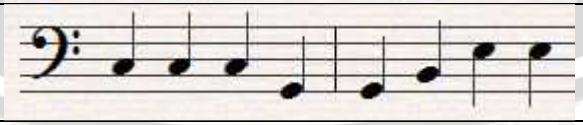
Lampiran 1

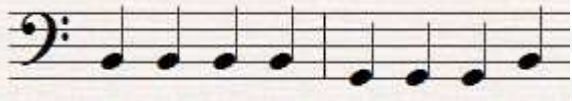
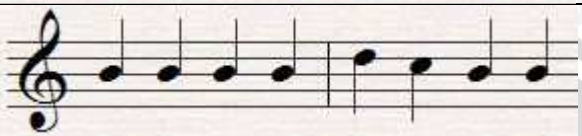
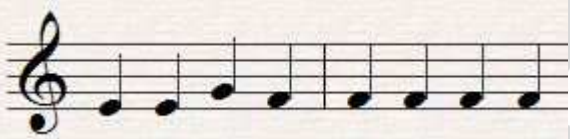
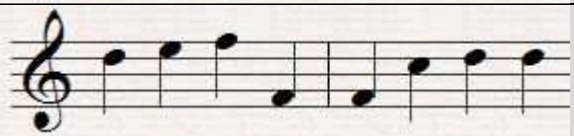
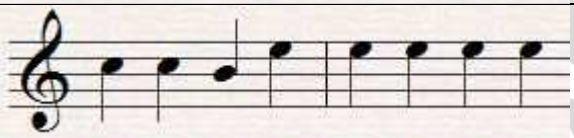
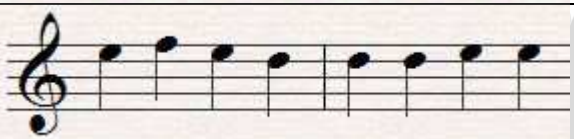
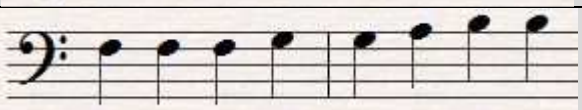
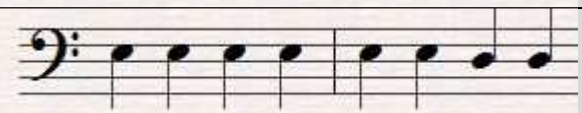
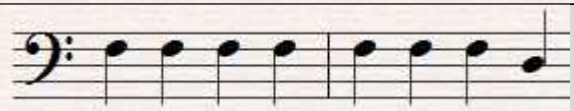
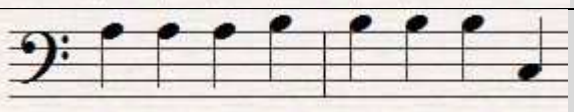
Template Simbol Musik

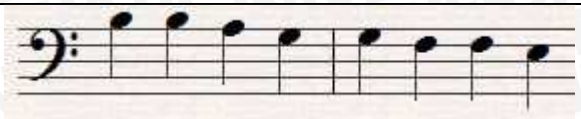
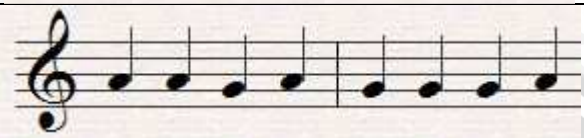
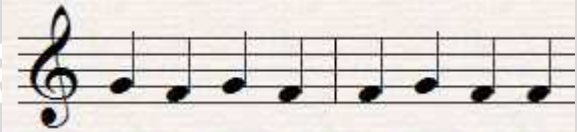
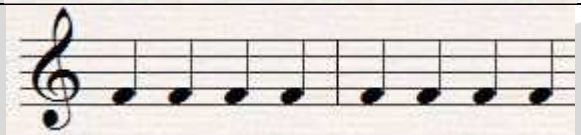
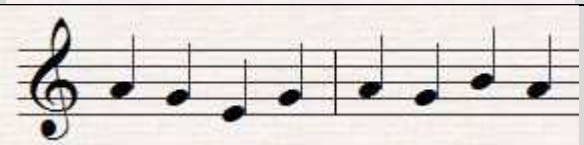
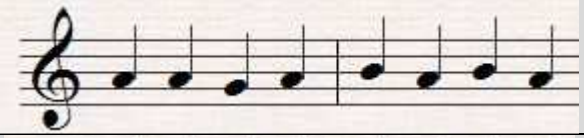
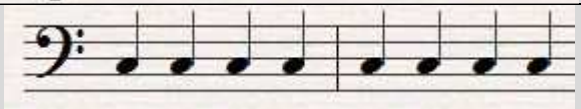
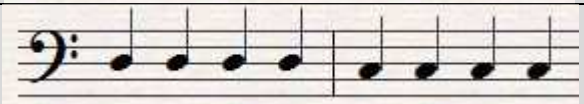
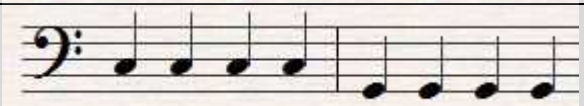
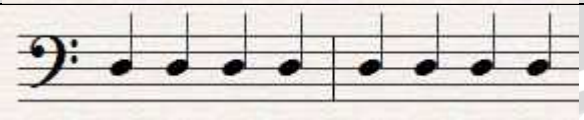
Template	Citra
GClef	
FClef	
Quarter Note Up	
Quarter Note Down	

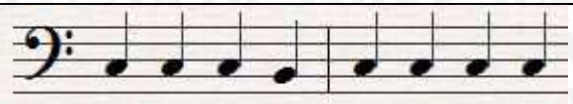
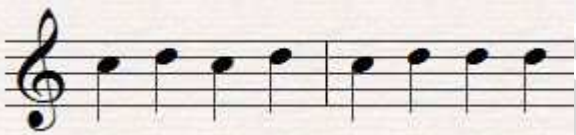
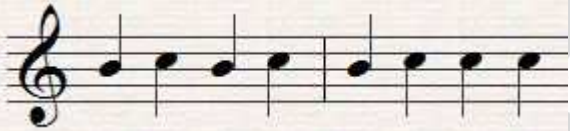
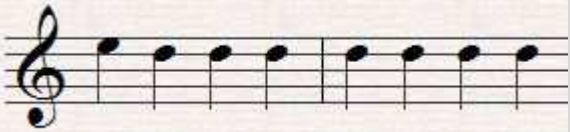
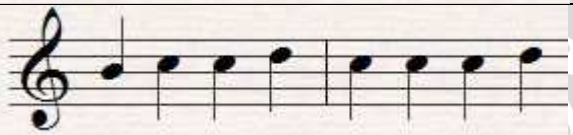
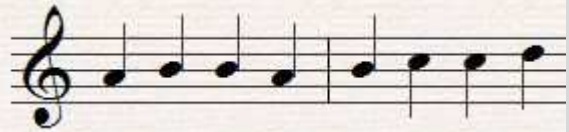
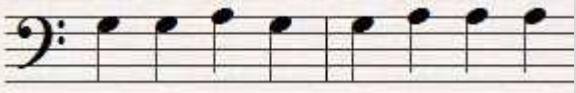
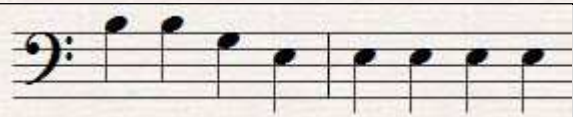
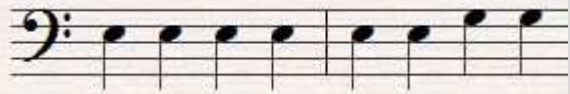
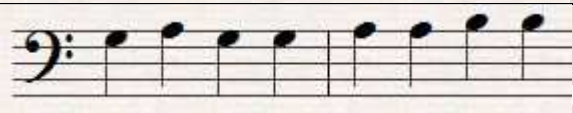
Lampiran 2

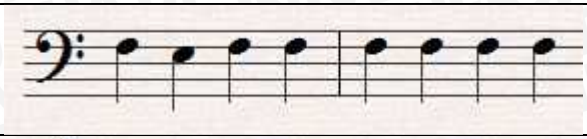
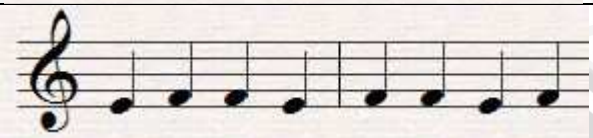
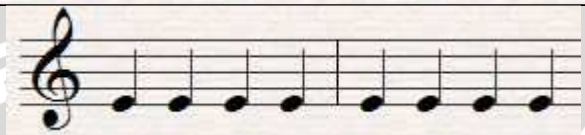
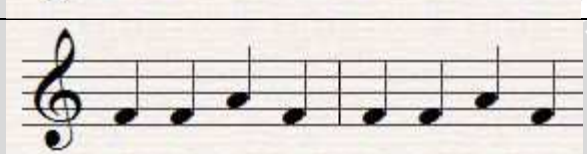
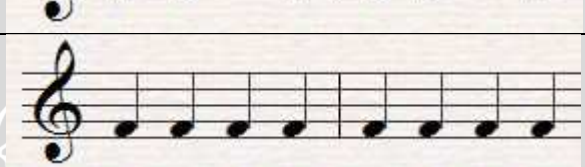
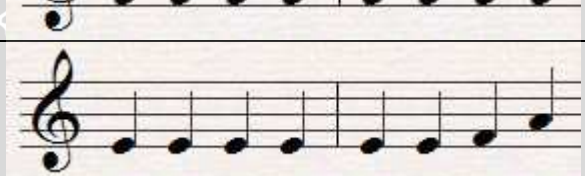
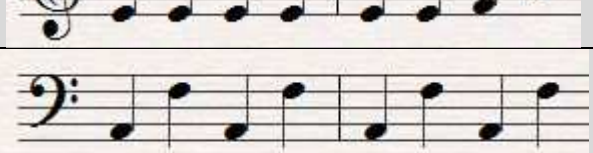
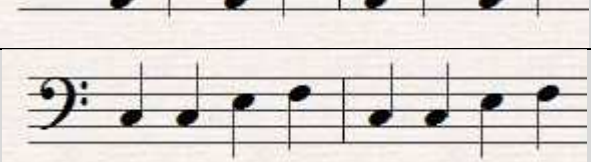
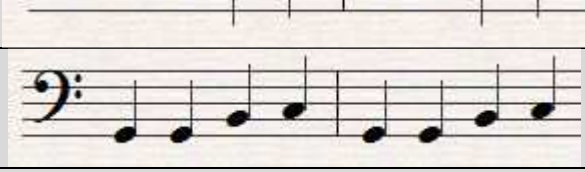
Data Pengujian

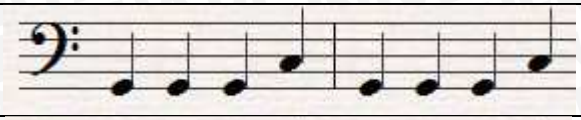
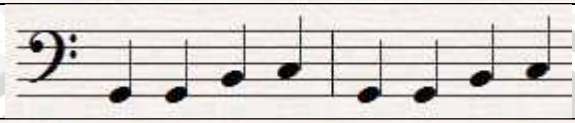
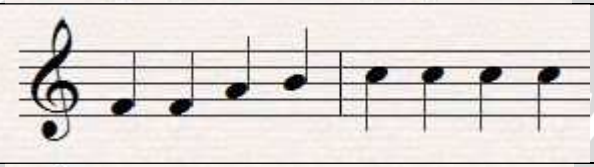
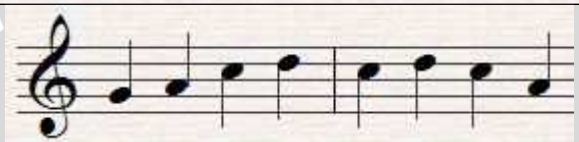
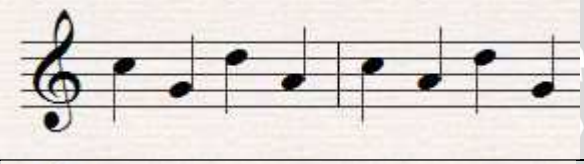
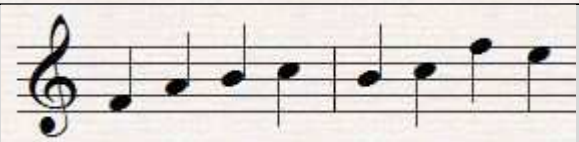
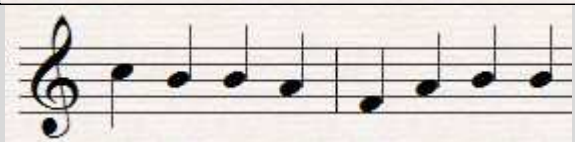
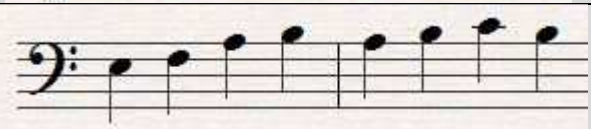
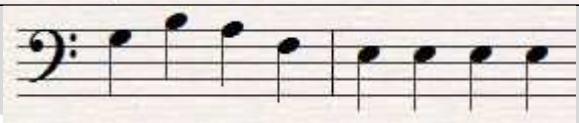
Data Uji	Citra
1	
2	
3	
4	
5	
6	
7	
8	
9	

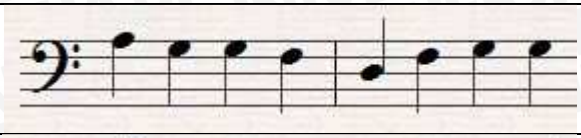
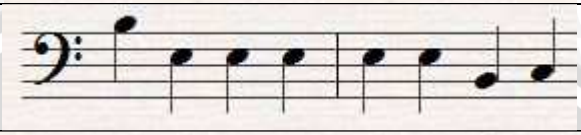
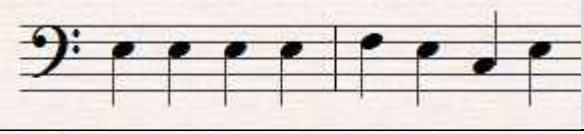
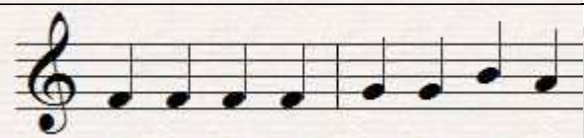
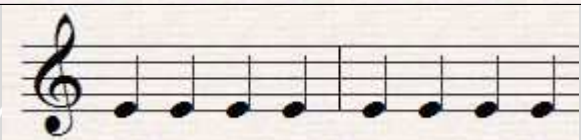
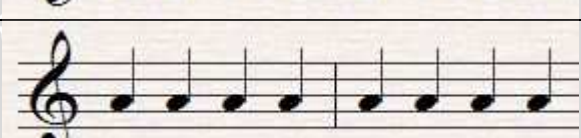
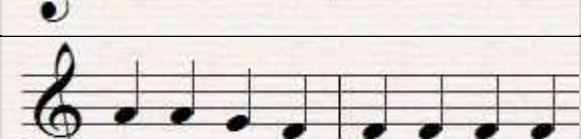
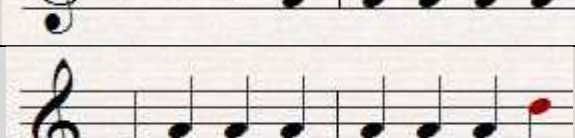
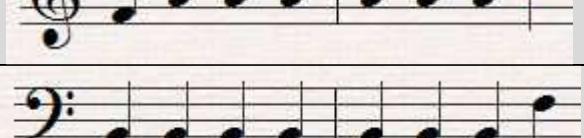
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	

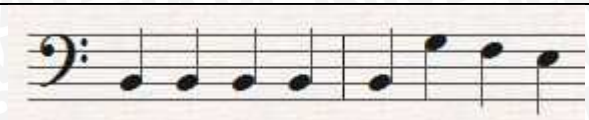
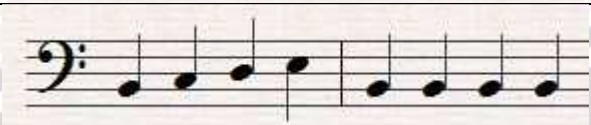
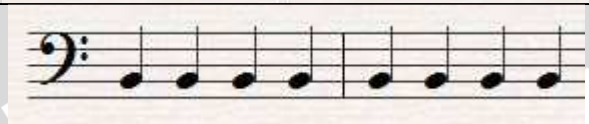
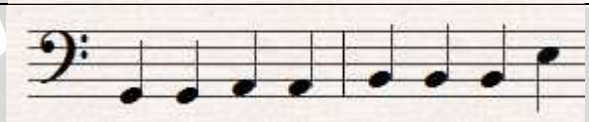
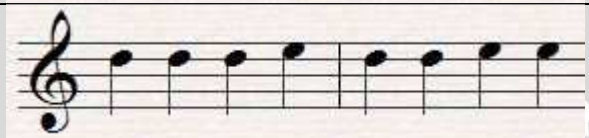
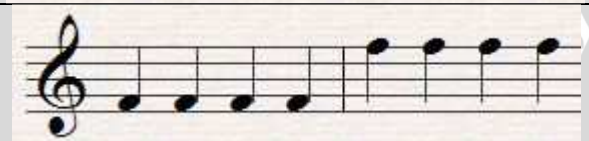
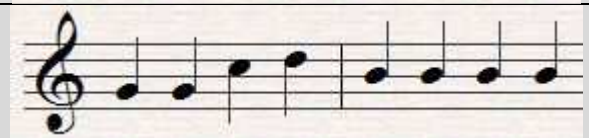
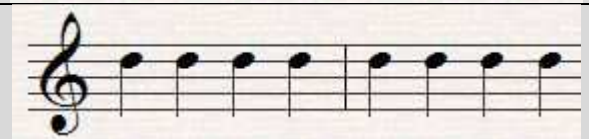
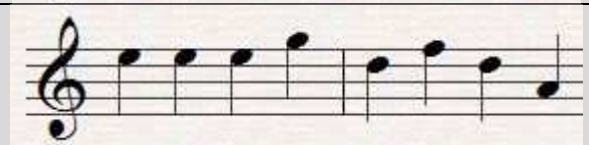
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	

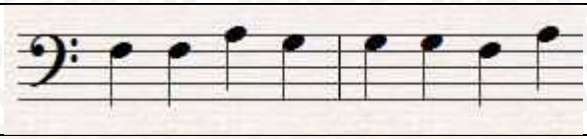
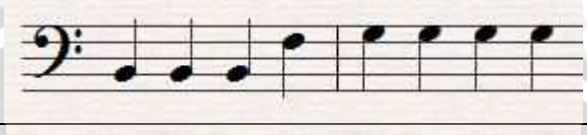
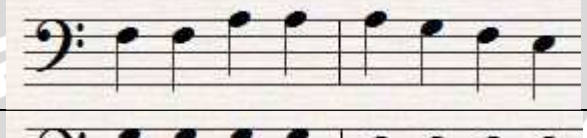
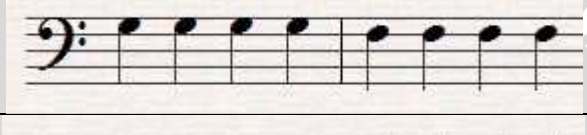
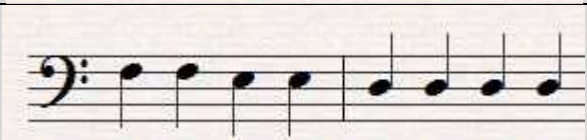
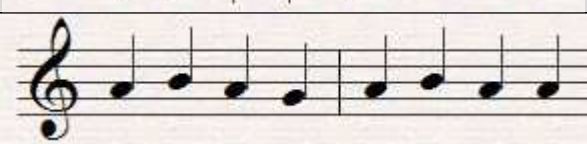
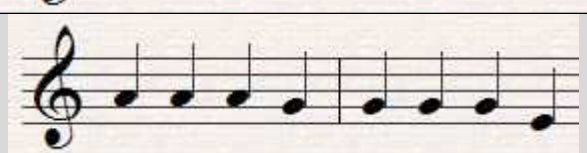
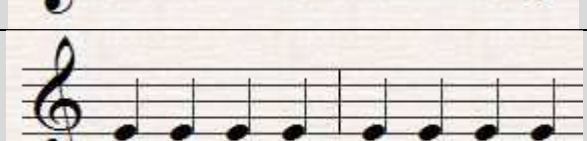
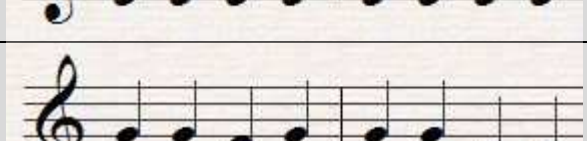
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	

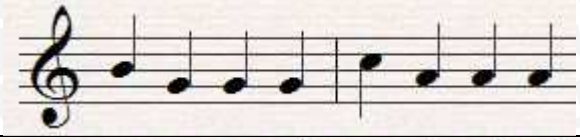
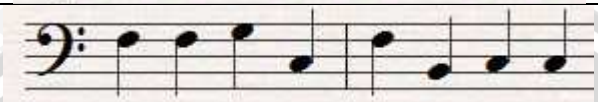
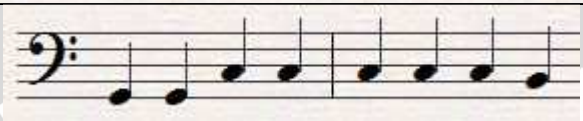
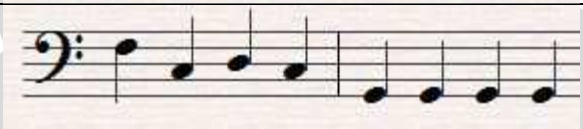
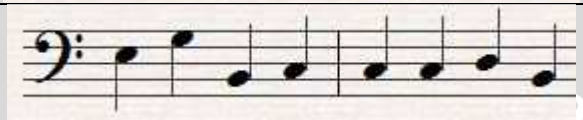
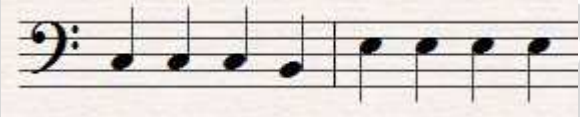
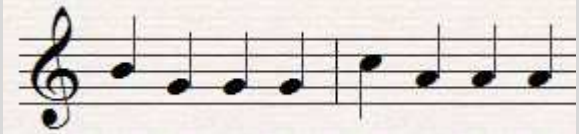
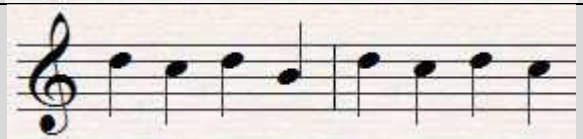
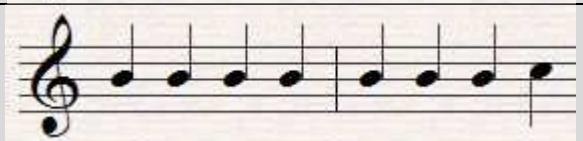
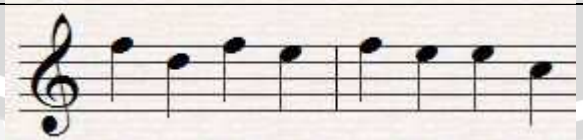
40	
41	
42	
43	
44	
45	
46	
47	
48	

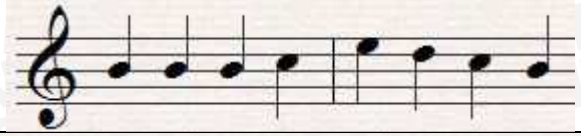
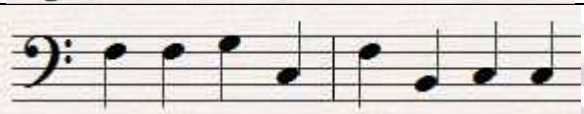
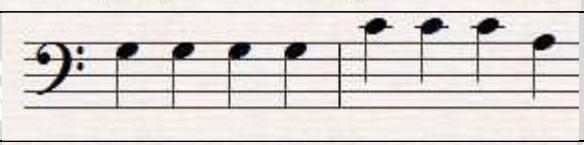
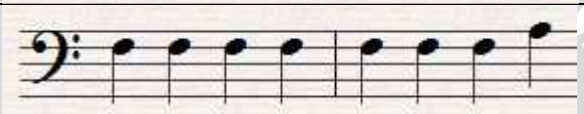
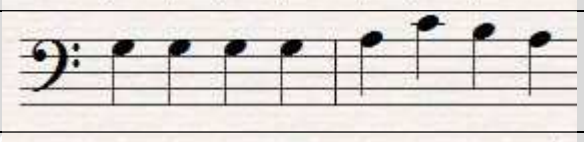
49	
50	
51	
52	
53	
54	
55	
56	
57	

58	
59	
60	
61	
62	
63	
64	
65	
66	

67	
68	
69	
70	
71	
72	
73	
74	
75	

76	
77	
78	
79	
80	
81	
82	
83	
84	

85	
86	
87	
88	
89	
90	
91	
92	
93	
94	

95	
96	
97	
98	
99	
100