**TUGAS AKHIR – KI141502**

# Reversible Multi-Layer Steganography pada Audio Menggunakan Metode Reduced Difference Expansion

**MUHAMMAD BAGUS ANDRA**
**NRP 5111100086**

**Dosen Pembimbing**
**Tohari Ahmad S.Kom., MIT., Ph.D.**

**JURUSAN TEKNIK INFORMATIKA**
**FAKULTAS TEKNOLOGI INFORMASI**
**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**
**SURABAYA**
**2015**


**FINAL PROJECT– KI141502**

# Reversible Multi-Layer Steganography in Audio With Reduced Difference Expansion Method

**MUHAMMAD BAGUS ANDRA**
**NRP 5111100086**

**Advisor**
**Tohari Ahmad S.Kom., MIT., Ph.D.**

**DEPARTMENT OF INFORMATICS**
**FACULTY OF INFORMATION TECHNOLOGY**
**SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY**
**SURABAYA**
**2015**

# LEMBAR PENGESAHAN

## REVERSIBLE MULTI-LAYER STEGANOGRAPHY PADA AUDIO MENGGUNAKAN METODE REDUCED DIFFERENCE EXPANSION

**Tugas Akhir**

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
**MUHAMMAD BAGUS ANDRA**
NRP. 5111 100 086

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Tohari Ahmad S.Kom., MIT., PHD. .....
NIP: 19750525 200313 1 002 (pembimbing 1)

**SURABAYA**
**JUNI, 2015**

**REVERSIBLE MULTI-LAYER STEGANOGRAPHY PADA AUDIO MENGGUNAKAN METODE REDUCED DIFFERENCE EXPANSION**

Nama Mahasiswa : Muhammad Bagus Andra
NRP : 51111 100 086
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Tohari Ahmad S.Kom., MIT., Ph.D.

## ABSTRAK

*Steganografi adalah sebuah praktik mengubah data secara tidak terdeteksi untuk menyisipkan sebuah pesan pada data tersebut. Secara umum, sistem steganografi terdiri dari dua komponen yaitu Encoder dan Decoder. Saat ini penelitian mengenai steganografi telah banyak dilakukan, namun kebanyakan metode yang ada hanya dapat diterapkan pada berkas citra. Metode ini juga memiliki kelemahan dimana berkas yang digunakan sebagai media penyisipan tidak dapat dikembalikan seperti semula. Untuk itu perlu digunakan metode yang bersifat reversible.*

*Pada berkas audio akan disisipkan pesan dengan menggunakan metode steganografi yang bersifat reversible yaitu RDE. Parameter seperti panjang segmen, nilai threshold, dan jumlah layer yang ditentukan dalam proses akan mempengaruhi kapasitas penyimpanan serta kualitas dari berkas audio yang diproses. Pada akhirnya pesan yang disisipkan dan berkas audio yang digunakan dapat diperoleh kembali tanpa adanya informasi yang hilang.*

*Tujuan dari pembuatan tugas akhir ini adalah untuk menerapkan dan menganalisa hasil dari RDE yang diterapkan pada berkas audio.*

***Kata kunci**: Steganografi, Reversible, Berkas Audio***

**Reversible Multi-Layer Steganography in Audio With Reduced Difference Expansion Method**

Student Name : Muhammad Bagus Andra
NRP : 51111 100 086
Major : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Tohari Ahmad S.Kom., MIT., Ph.D.

## ABSTRACT

*Steganography is a practice to change a data without being detected to embed some message in the said data. Generally, the steganography system is composed from two components, the Encoder component and the Decoder component. Current Research in the field of steganography has been done extensively, but many of the methods can only be applied to an image file. These methods also has some flaws where the file that's used for the cover cannot be retrieved completely again. To achieve this, we need to use a reversible method.*

*The message will be embedded in an audio file with a reversible steganography method that is RDE.Parameters such as segment length, threshold value, and number of layers will affect the capacity and the quality of the audio file that being processed. In the end, both the embedded message and the audio file used for cover can be retrieved without losing even a single information.*

*The aim of this final project is to apply and to analyze the result of using RDE as an steganography method for embedding data in an audio file.*

***Keywords*:*Steganography, Reversible, Audio File***

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji dan syukur, kehadirat Allah Subhanahu wa ta'ala yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "**Reversible Multi-Layer Steganography Pada Audio Menggunakan Metode Reduced Difference Expansion**".

Bagi penulis pribadi, tugas akhir ini bukanlah sekedar syarat untuk lulus, namun juga merupakan sebuah karya sebagai bukti kecintaan penulis terhadap bidang informatika yang penulis harap dapat berguna bagi kemajuan bidang informatika di Indonesia, khususnya pada bidang sekuritas jaringan.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak. Melalui lembar ini, penulis ingin secara khusus menyampaikan ucapan terima kasih kepada:

1. Allah Subhanahu wa ta'ala yang telah melimpahkan rahmat, hidayah, dan inayah-Nya sehingga penulis, Alhamdulillah mampu menyelesaikan tugas akhir dengan baik.
2. Junjungan kita Nabi Muhammad Shallahu Alayhi Wasallam yang telah menjadi inspirasi dan tauladan.
3. Ibu dari penulis yang telah mencurahkan cinta dan kasih sayang nya yang tak terhingga.
4. Ibu dari penulis yang selalu mendukung dan memberikan doa nya tanpa henti kepada penulis.
5. Ibu dari penulis yang selalu tegar, bersabar dan bekerja keras demi penulis.
6. Bapak Tohari Ahmad S.Kom., MIT., Ph.D. selaku dosen pembimbing yang telah memberikan bimbingan, motivasi, dan meluangkan waktu untuk membantu pengerjaan tugas akhir ini.
7. Ibu Prof. Ir. Handayani Tjandrasa, M.Sc.,Ph.D selaku dosen wali yang telah memberikan perhatian dan motivasi kepada

penulis selama menjadi mahasiswa di lingkungan Teknik Informatika ITS.

8. Ibu Nanik Suciati, S.Kom., M.Kom., Dr.Eng. selaku ketua jurusan Teknik Informatika ITS, Bapak Radityo Anggoro, S.Kom, M.Sc. selaku koordinator TA dan koordinator KP dan segenap Bapak/Ibu dosen Teknik Informatika yang telah memberikan ilmunya kepada penulis.
9. Yuujou sebagai sahabat-sahabat setia penulis yang selalu menemani penulis di masa suka dan duka. Kepada rahadian di seberang laut yang selalu menghibur peulis, kepada wisnu yang selalu iseng, kepada elriandi yang selalu membantu penulis, kepada fandy yang selalu seru, kepada asep yang selalu kreatif dan kepada prabu yang selalu humoris.
10. Hashfi, Askary, Aldy, Ilmi yang tidak pernah bosan untuk penulis ajak ngobrol, keluarga IBS(KCV) yang telah memberikan tempat saya bernaung dan teman-teman 2011 yang telah menemani penulis selama 4 tahun ini.
11. Hayam, Sindu, Haidar, Faldi, Uthe orang-orang hebat yang menjadi bukti nyata dari pepatah "seperti ilmu padi, semakin berisi, semakin merunduk".
12. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya tugas akhir ini, penulis mengucapkan terima kasih.

Penulis telah berusaha sebaik mungkin dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Juni 2015
Penulis

Muhammad Bagus Andra

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Berkembangnya teknologi jaringan terutama *internet* memberikan dampak besar pada distribusi dan pengiriman berkas *digital.* Ada kalanya berkas *digital* yang dikirimkan bersifat rahasia dan *confidential* seperti, data catatan kesehatan pasien atau pesan militer, sehingga harus dipastikan isi dari berkas hanya diketahui oleh pihak yang berhak. Salah satu metode yang dapat digunakan utnuk menjaga kerahasiaan adalah steganografi.

Steganografi adalah sebuah praktik mengubah data secara tidak terdeteksi untuk menyisipkan sebuah pesan [1]. Secara umum, sistem steganografi terdiri dari dua komponen yaitu *Encoder* dan *Decoder*. Bagian *Encoder* akan mempunyai dua buah masukan, yaitu data rahasia atau yang biasa disebut *payload* dan *cover,* yaitu berkas yang akan digunakan sebagai media untuk menyisipkan data rahasia. Kedua masukan ini kemudian diproses untk menghasilkan *encoded data,* yaitu data yang telah disisipkan pesan rahasia. Pada bagian *Decocer*, *encoded data* akan menjadi masukan yang kemudian diproses untuk didapatkan kembali pesan rahasia yang tersimpan di dalamnya[1].

Penelitian pada bidang keamanan informasi dan steganografi telah banyak dilakukan dan berkembang pada beberapa tahun terakhir ini. Namun sepanjang pengetahuan saya, kebanyakan metode yang telah ada hanya dapat diterapkan pada berkas citra, sedangkan metode yang dapat digunakan pada berkas bertipe audio masih sangat terbatas. Beberapa metode yang telah ada mencakup, *Low Bit Encoding, Echo Hiding, Spread Spectrum, Phase Coding* [2] , dan juga menggunakan transformasi *wavelet* [3]. Namun, metode ini memiliki kelemahan dimana berkas audio *cover* dan *secret message* tidak dapat dikembalikan persis seperti semula dikarenakan adanya modifikasi pada domain spasial ataupun domain frekuensi. Modifikasi tersebut bersifat *irreversible* atau tidak dapat dikembalikan. Hal ini tentu harus

dihindari karena informasi pada *secret message* bersifat sensitif dan harus dikembalikan tanpa adanya sedikitpun informasi yang hilang.

Untuk mendapatkan kembali data *cover* dan *secret message* tanpa kehilangan informasi, harus dilakukan modifikasi yang bersifat *reversible* pada berkas. Hal ini dapat dilakukan dengan menggunakan metode transformasi *Difference Expansion* yang pertama kali diusulkan oleh Tian [4]. Banyak metode yang diajukan sebagai pengembangan metode ini seperti, *Reduced Difference Expansion* dan *Quad Smoothness* [5], tetapi metode ini hanya dapat diaplikasikan pada berkas citra.

Dalam tugas akhir ini, metode *Reduced Difference Expansion* yang awalnya hanya dapat diterapkan pada berkas citra akan dicoba diterapkan pada berkas audio dengan ditambah beberapa modifikasi untuk memaksimalkan kapasitas serta kualitas dari berkas audio. Sebelum metode dapat diaplikasikan harus dilakukan beberapa tahap *pre-processing* agar metode tersebut dapat diaplikasikan pada berkas audio. Tahap ini mencakup pembentukan segmen sampel audio dan paritisi bit pada sampel audio.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara memproses berkas audio dan mengaplikasikan metode RDE pada berkas audio?
2. Pengembangan metode apa yang dapat dilakukan untuk memaksimalkan kapasitas penyimpanan dan kualitas dari berkas audio?
3. Bagaimana cara menentukan parameter pada proses steganografi yaitu besar segmen, nilai *threshold* dan jumlah layer sehingga hasilnya dapat memenuhi kriteria atau kebutuhan dari pengguna?

## 1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Metode hanya diterapkan pada audio dengan *format* WAV
2. Berkas audio hanya terdiri dari *single channel* atau *mono*
3. *Bit-depth* atau panjang sampel dari audio yang digunakan sebagai *cover* adalah *signed 16-bit*
4. *Payload* yang digunakan adalah teks dengan *format* TXT

## 1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah untuk menerapkan metode steganografi yang bersifat *reversible* yaitu RDE pada berkas audio serta menganalisis performa dan hasil dari metode tersebut.

## 1.5 Manfaat

Manfaat dari tugas akhir ini adalah memberikan alternatif metode baru untuk melakukan steganografi pada berkas audio yang bersifat *reversible* dengan mencoba memaksimalkan kapasitas dan kualitas dari hasil *encoding*.

## 1.6 Metodologi

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

A. Studi literatur
   Pada tahap ini dilakukan pencarian dan pemahaman literatur yang diperlukan untuk melakukan pemrosesan berkas audio serta mengaplikasikan metode RDE pada berkas audio yang mencakup buku, jurnal serta publikasi *paper*.

B. Perancangan sistem dan pengembangan metode
   Pada tahap ini dilakukan perancangan sistem secara keseluruhan serta proses studi dan pengembangan metode yang telah ada untuk mendapatkan hasil yang lebih baik

C. Implementasi Dan Pembuatan Sistem
Pada tahap ini dilakukan implementasi metode yang diusulkan dari rancangan yang telah dibuat pada tahap sebelumnya

D. Uji Coba Dan Evaluasi
Pada tahap ini dilakukan uji coba hasil steganografi. Adapun hal-hal yang diuji dan dievaluasi antara lain adalah kebenaran metode, kapasitas penyimpanan *payload* serta nilai PSNR(*Peak Noise to Signal Ratio*) yang menggambarkan kualitas dari hasil steganografi,uji coba *reversibility* dan uji coba proses GA.

E. Penyusunan Laporan Tugas Akhir
Pada tahap ini dilakukan penyusunan laporan yang berisi dasar teori, dokumentasi dari sistem, hasil yang didapat serta analisis dan perbandingan yang dilakukan.

## 1.7 Sistematika Penulisan

Buku tugas akhir ini terdiri dari beberapa bab, yang dijelaskan sebagai berikut.

**BAB I PENDAHULUAN**
Bab ini berisi latar belakang masalah, rumusan dan batasan permasalahan, tujuan dan manfaat pembuatan tugas akhir, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

**BAB II TINJAUAN PUSTAKA**
Bab ini membahas dasar pembuatan dan beberapa teori penunjang yang berhubungan dengan pokok pembahasan yang mendasari pembuatan tugas akhir ini.

**BAB III ANALISIS DAN PERANCANGAN**
Bab ini membahas analisis dari sistem yang dibuat meliputi analisis permasalahan, deskripsi umum sistem dan rancangan dari sistem yang dibuat meliputi rancangan arsitektur, data dan antarmuka.

**BAB IV IMPLEMENTASI**

Bab ini membahas implementasi dari rancangan sistem yang dilakukan pada tahap perancangan. Penjelasan implementasi meliputi implementasi metode steganografi dan antarmuka sistem.

**BAB V PENGUJIAN DAN EVALUASI**

Bab ini membahas pengujian dari aplikasi yang dibuat dengan melihat keluaran yang dihasilkan oleh sistem dan analisa dari hasil steganografi.

**BAB VI PENUTUP**

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan serta saran untuk pengembangan sistem dan metode selanjutnya.

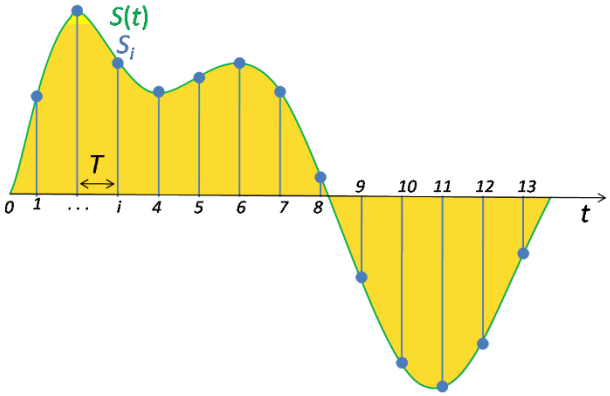
# BAB II
# TINJAUAN PUSTAKA

## 2.1 Pemrosesan Berkas Audio

Pada komputer, berkas audio direpresentasikan oleh sinyal digital. Sinyal digital didapatkan dari sinyal analog yang diproses melalui tahap *sampling* dan *quantization*. Proses ini dimaksudkan untuk mengubah sifat sinyal analog yang aslinya bersifat kontinu menjadi bentuk diskrit.

a. *Sampling*

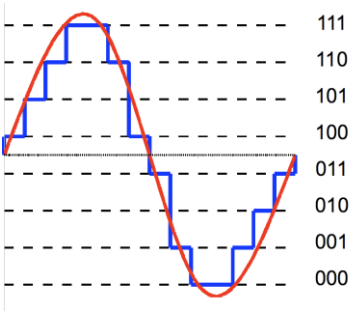
*Sampling* merupakan proses reduksi gelombang sinyal kontinu menjadi sinyal diskrit dengan cara mengambil suatu titik sampel per satuan waktu. *Sampling rate* menggambarkan berapa banyak sampel yang diambil setiap detiknya. Sebuah sinyal digital dengan *sampling rate* 44100Hz berarti dalam 1 detik dilakukan pengambilan sampel sebanyak 44100 kali [7]. Proses *sampling* ditunjukkan pada Gambar 2.1.




**Gambar 2.1 Proses Sampling [7]**

b. *Quantization*

*Quantization* merupakan proses pemetaan amplitudo dari setiap sampel yang telah diambil kedalam satuan diskrit. Setiap sampel pada berkas audio akan direpresentasikan dalam bilangan baik itu berbentuk bilangan bulat (*integer*) maupun *float*. Bilangan ini mewakili nilai amplitudo dari masing-masing sampel [7]. Proses *quantization* ditunjukkan pada Gambar 2.2.




**Gambar 2.2 Proses Quantization [7]**

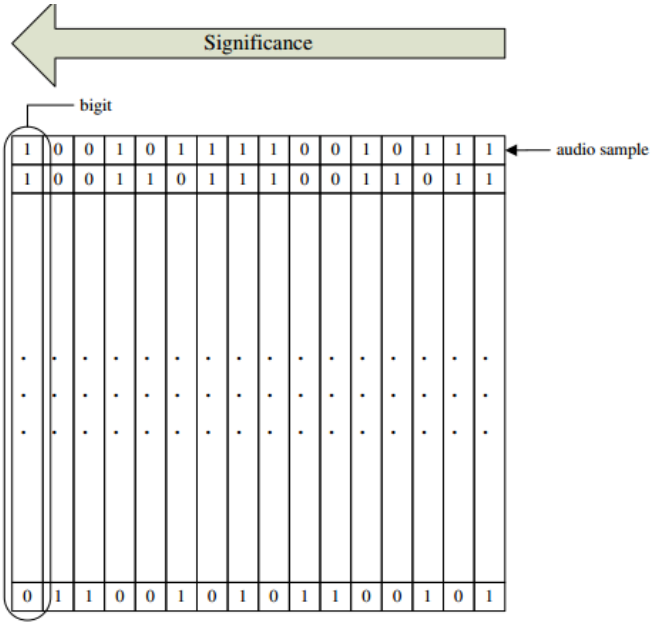
Setelah dilakukan proses *sampling* dan *quantization*, setiap nilai sampel akan disimpan dalam bentuk bit. Besar berkas audio akan bergantung oleh jumlah sampling yang dilakukan dan juga kedalaman atau *depth* dari amplitudo yang disimpan.

## 2.2 *Intelligent Partitioning*

Adanya perbedaan besar sampel pada berkas citra dan audio dimana pada umumnya berkas audio memiliki besar sampel sebesar 16 bit sedangkan piksel pada citra memiliki besar 8 bit menyebabkan tidak dapatnya diaplikasikan beberapa metode steganografi citra seperti RDE pada berkas audio. Untuk mengatasi hal tersebut perlu dilakukan tahap *Pre-Processing* pada berkas audio agar metode tesebut dapat diaplikasikan.

Untuk mengubah besar sampel berkas audio yang memiliki besar 16 bit menjadi 8 bit maka dilakukan partisi pada sampel audio untuk membaginya menjadi dua bagian yang masing-masing memiliki besar 8 bit. Kemudian penyisipan data *payload* akan dilakukan pada masing-masing bagian tersebut [8].

Partisi dilakukan dengan cara merepresentasikan gelombang audio sebagai 16 buah *array* dimana masing-masing array ini disebut *bigit.* Representasi *array* ini dapat dilihat pada Gambar 2.3.




**Gambar 2.3 Representasi *Array* Sampel Audio**

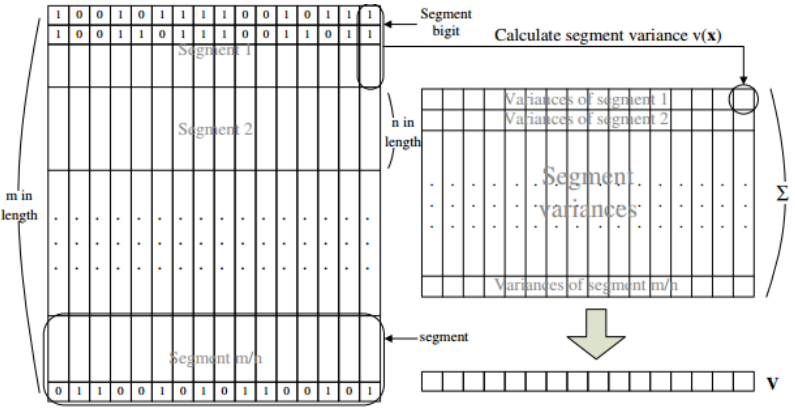
Setiap kolom merepresentasikan sebuah *bigit* dan setiap baris merepresentasikan satu sampel pada berkas audio. Proses partisi *array* dengan panjang 16 bit akan dilakukan dengan membentuk dua grup *array* M1 dan M2 yang memiliki 8 *bigit* sehingga panjang dari

baris akan menjadi 8 bit. Penempatan dari tiap *bigit* akan ditentukan oleh nilai *variance* dari tiap *bigit.*

Untuk menentukan pembagian kelompok dari tiap bigit maka kita perlu menghitung nilai *variance* dari tiap *bigit* pada sampel audio. Untuk sebuah array sampel M, nilai *variance* V didapatkan dengan proses yang ditunjukkan pada Gambar 2.4, dimana *m* adalah panjang dari keseluruhan sampel audio, dan *n* adalah panjang dari segmen. *Variance* V adalah vektor yang menunjukkan jumlah dari *variance* pada tiap segmen dari setiap *bigit* pada M. Untuk setiap segmen dari setiap bigit pada M , *variance segment v(x)* didapatkan dengan (1).

$$(x) = \sum_{j=1}^{n} (x_j - a(x))^2 \qquad \textbf{(1)}$$

Dimana *x* adalah vektor ( $x_1, x_2, x_3 \ldots . x_n$ ) yang merepresentasikan satu buah segmen bigit dan *a(x)* adalah rata-rata dari *x* yang dibulatkan. Untuk mencari V, *variance* dari semua segmen untuk setiap *bigit* harus ditemukan terlebih dahulu, kemudian semua *variance* tersebut dijumlahkan yang akah menghasilkan V.




**Gambar 2.4 Proses Penghitungan *Variance* [8]**

Setelah V ditemukan, kita akan membentuk sebuah *array* yang berisikan *variance* dari tiap bigit yang diurutkan secara menurun. Tiap indeks dari *array* berkorespondensi dengan tiap *bigit*. Kemudian tiap *bigit* akan ditempatkan pada M1 atau M2 sesuai dengan skema berikut:

1. Tempatkan 8 *Most Significant Bit* (MSB) dari *array variance* yang telah diurutkan pada tahap sebelumnya pada M1 dan M2 secara bergantian (bit pertama pada M1, bit kedua pada M2, bit ketiga pada M1 dan seterusnya).
2. Tempatkan *bigit* indeks pertama dan ketiga pada M1 dan *bigit* indeks kedua dan keempat pada M2. *Bigit* ini adalah *bigit* selain dari *bigit* yang telah dioperasikan pada tahap sebelumnya.
3. Tempatkan 4 *bigit* berikutnya pada M1 dan M2 masing-masing sebanyak dua *bigit*.

Setelah prosedur selesai, *array* sampel M akan terbagi menjadi M1 dan M2 yang masing-masing memiliki panjang 8 bit. *Array variance* yang telah disorting akan disimpan sebagai *array* partisi yang akan digunakan kembali saat proses *decoding.*

## 2.3 *Reduced Difference Expansion*

*Reduced Difference Expansion* merupakan pengembangan dari metode *Difference Expansion*. Pada DE, proses *embedding* atau penyisipan menyebabkan adanya kenaikan dari selisih nilai piksel hampir sebesar dua kali, oleh karena itu pada RDE diusulkan persamaan baru yang dapat memperkecil selisih pada proses embedding tersebut [5]. Pada RDE tahap penyisipan dibagi menjadi dua tahap yaitu tahap reduksi dan tahap penyisipan atau *embedding*.

a. Reduksi

Pada proses RDE, sebelum dilakukan penyisipan data selisih antara piksel direduksi terlebih dahulu, hal ini dilakukan agar

nilai akhir setelah proses penyisipan tidak terlalu jauh dengan nilai sampel awal dari berkas audio sehingga akan dicapai kualitas audio yang lebih baik setelah proses *embedding* selesai. Persamaan yang digunakan utnuk melakukan reduksi ditunjukkan oleh (2.1).

$$\check{v} = \begin{cases} v, & if\ v < 2 \\ v - 2^{\lfloor log_2 v \rfloor - 1}, & if\ v \geq 2 \end{cases} \quad \textbf{(2.1)}$$

Dimana $\bar{v}$ adalah selisih setelah reduksi dan *v* adalah selisih awal piksel.

Setiap piksel yang dilakukan reduksi akan diberi tanda yang disimpan di dalam *reduced map*. Piksel yang dilakukan reduksi akan diberi tanda dengan nilai 1 sedangkan piksel yang tidak dioperasikan akan diberi tanda dengan nilai 0. Saat proses *decoding* dilakukan, *reduced map* akan digunakan untuk mendapatkan nilai selisih asli dari piksel. Persamaan (2.2) akan digunakan untuk mendapatkan nilai selisih asli untuk piksel yang memiliki tanda 1 sedangkan (2.3) akan digunakan untuk piksel dengan tanda 0.

$$v = \bar{v} + 2^{\log_2 |\bar{v}|} \quad \textbf{(2.2)}$$

$$v = \bar{v} + 2^{\log_2 |\bar{v}|} - 1 \quad \textbf{(2.3)}$$

Dimana $\bar{v}$ adalah selisih setelah reduksi dan *v* adalah selisih piksel awal yang akan dikembalikan.

b. Penyisipan Data

Setelah selisih piksel direduksi, penyisipan data dilakukan dengan proses yang sama dengan metode *Difference Expansion* penyisipan setiap bit data *payload* ditunjukkan oleh persamaan (2.4).

$$\check{v} = \ 2\ x\ v + b \qquad \textbf{(2.4)}$$

Dimana $\check{v}$ adalah selisih setelah dilakukan penyisipan dan $v$ adalah selisih awal dan b adalah data bit *payload* yang akan disisipkan (0 atau 1). Setelah itu nilai akhir piksel dapat didapatkan melalui (2.5).

$$\check{u} = \ u + \check{v} \qquad \textbf{(2.5)}$$

## 2.4 *Improved Reduced Difference Expansion*

Metode RDE melakukan reduksi pada selisih piksel untuk mendapatkah hasil nilai piksel akhir yang lebih mendekati nilai piksel awal. Dengan demikian kualitas berkas audio akan lebih baik dan mendekati berkas audio aslinya.

Namun metode RDE masih dapat dikembangkan untuk mendapatkan hasil yang lebih baik. Dengan menggunakan persamaan baru, selisih antara piksel dapat direduksi lebih jauh lagi, sehingga hasil akhir piksel setelah proses penyisipan akan lebih mendekati nilai piksel awal. Persamaan ini ditunjukkan oleh (2.6).

$$\check{v} = \begin{cases} v & jika\ v < 4 \\ v - 2^{|\log_2 v|-1} - 2^{|\log_2 v|-2} & jika\ v \geq 4 \end{cases} \qquad \textbf{(2.6)}$$

Dimana $\check{v}$ adalah selisih piksel setelah direduksi dan $v$ adalah nilai selisih awal. Setiap piksel yang direduksi akan diberi tanda sesuai kondisi piksel. Tanda tersebut akan disimpan pada *reduced map*. Tanda untuk setiap piksel yang di proses dijabarkan pada Tabel 2.1.

| kondisi | Tanda |
|---|---|
| Kondisi standar | 0 |
| Jika $2^{\lfloor \log_2 v \rfloor} / 2^{\lfloor \log_2 \bar{v} \rfloor} = 4$ | 1 |
| Jika $2^{\lfloor \log_2 v \rfloor} / 2^{\lfloor \log_2 \bar{v} \rfloor} = 1$ | 2 |
| Nilai tidak berubah ($v$<4) | 3 |

**Tabel 2.1. Ketentuan Tanda Pada *Reduced Map***

Pada saat proses *decoding*, selisih asli piksel akan dikembalikan sesuai dengan tanda yang diberikan pada *reduced map*. Untuk piksel bertanda 0, nilai selisih akan dikembalikan dengan (2.7). Untuk piksel dengan tanda 1, selisih akan dikembalikan dengan (2.8). Untuk piksel dengan tanda 2, selisih akan dikembalikan dengan (2.9). Sedangkan untk piksel dengan tanda 3, selisih akan dikembalikan dengan (2.10).

$$\check{v} = v + 2^{\lfloor \log_2 v \rfloor - 1} + 2^{\lfloor \log_2 v \rfloor} \; jika \; tanda = 0 \qquad \textbf{(2.7)}$$

$$\check{v} = v + 2^{\lfloor \log_2 v \rfloor} + 2^{\lfloor \log_2 v \rfloor + 1} \; jika \; tanda = 1 \qquad \textbf{(2.8)}$$

$$\check{v} = v - 2^{\lfloor \log_2 v \rfloor - 1} - 2^{\lfloor \log_2 v \rfloor - 2} \; jika \; tanda = 2 \qquad \textbf{(2.9)}$$

$$\check{v} = v \; jika \; tanda = 3 \qquad \textbf{(2.10)}$$

## 2.5 Pustaka *SciPy Stack*

*SciPy Stack* adalah kumpulan dari beberapa perangkat lunak berbasis *open-source* yang digunakan untuk kepentingan komputasi dan sains dengan menggunakan bahasa python [9]. Secara garis besar *SciPy* terdiri atas beberapa perangkat inti yaitu:

- Python sebagai basis bahasa pemrograman

- NumPy, modul python untuk komputasi numerik
- SciPy Library, pustaka yang berisikan kumpulan algoritma dan kakas kerja, termasuk kakas untuk pemrosesan sinyal
- Matplotlib, modul yang berguna untuk merepresentasikan plot dan graf
- Pandas, antarmuka untuk struktur data
- Sympy, modul untuk matematika simbolis dan aljabar komputer
- iPython, antarmuka interaktif untuk mengetes kode
- nose, kerangka kerja untuk pengetesan kode python

## 2.6 *Peak Signal To Noise Ratio (PSNR)*

*Peak Signal to Noise Ratio* atau PSNR adalah sebuah metrik yang umum digunakan untuk menilai kualitas dari sebuah sinyal dengan cara membandingkan rasio *noise* yang ada pada sinyal dengan sinyal aslinya. PSNR digunakan secara luas sebagai metrik untuk mengukur kemiripan suatu sinyal dengan sinyal aslinya karena PSNR memiliki sifat tidak sensitif terhadap variasi angka yang kecil serta mudah dihitung [10].

Untuk menghitung PSNR pertama kita harus menghitung *Mean Squared Error* (MSE) dari kedua sinyal yaitu sinyal asli dan sinyal yang telah di rekonstruksi. MSE didapatkan dengan persamaan (2.11).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(P_i - Q_i)^2 \qquad \textbf{(2.11)}$$

Dimana *P* adalah sinyal asli, *Q* adalah sinyal hasil rekonstruksi dan *n* adalah jumlah sampel keseluruhan dari sinyal. PSNR kemudian didapatkan melalui (2.12).

$$PSNR = 20_{log_{10}} \frac{\max(|P_i|)}{\sqrt{MSE}} \qquad \textbf{(2.12)}$$

Dimana MSE adalah *Mean Squared Error* yang telah didapatkan dari perhitungan sebelumnya dan $\max(|P_i|)$ adalah nilai maksimal dari panjang bit pada sampel. Jika sampel memiliki panjang 8 bit maka nilainya adalah 255 sedangkan untuk berkas audio, dimana panjang sampelnya adalah 16 bit nilainya adalah 65536.

Semakin besar nilai PSNR menunjukkan bahwa sinyal yang direkonstruksi semakin identik dengan sinyal awal. Jika sinyal awal dan sinyal rekonstruksi identik, maka PSNR akan bernilai tidak terhingga.

## 2.7 *Genetic Algorithm* (GA)

*Genetic Algorithm* atau GA adalah salah satu bagian dari *Evolutionary Computation* atau komputasi yang berevolusi, yaitu sebuah paradigma dimana sebuah komputasi dapat berevolusi dan menyesuaikan diri dengan permasalahan yang diberikan untuk mencapai solusi optimal [11]. GA didasari oleh fenomena biologis evolusi makhluk hidup dimana pada proses tersebut terjadi variasi genetik dan seleksi alam untuk menghasilkan individu yang kuat dan dapat beradaptasi dengan alam. Operasi-operasi pada GA didasari pada sebuah populasi kromosom yang direpresentasikan oleh *string of bits*, kromosom ini berisikan parameter-parameter yang akan digunakan dalam sebuah model penyelesaian masalah. Pada populasi ini kemudian dilakukan operasi genetik seperti *crossover* atau mutasi untuk mendapatkan individu atau kromosom baru yang memiliki parameter untuk menyelesaikan masalah dengan hasil yang lebih optimal.

Pada umumnya proses iterasi GA adalah sebagai berikut [10] :

- Mulai dengan membuat populasi dengan sejumlah *n* kromosom yang dibangkitkan secara acak yang menjadi kandidat untuk solusi dari sebuah permasalahan. Tabel 2.2 menunjukkan contoh dari populasi GA.

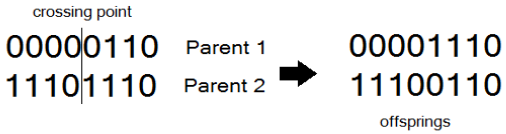
| Label Kromosom | *String* Kromosom | *Fitness Value* |
|---|---|---|
| A | 00000110 | 2 |
| B | 11101110 | 6 |
| C | 00100000 | 1 |
| D | 00110100 | 3 |

**Tabel 2.2. Populasi GA**

- Hitung *fitness value f(x)* untuk setiap kromosom *x* pada populasi. *Fitness Value* adalah nilai yang merepresentasikan keunggulan dari suatu kromosom. Nilai ini bergantung pada permasalahan yang ingin dicari nilai optimal nya, untuk permasalahan steganografi, *fitness value* yang digunakan adalah kapasitas penyimpanan.

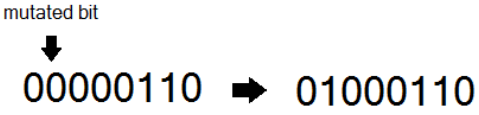
- Lakukan hal berikut hingga jumlah turunan sebanyak *n* dibuat:
    1. Pilih sepasang kromosom dari populasi saat ini. Probabilitas dari kromosom yang dipilih akan sesuai dengan *fitness value* yang dimiliki oleh kromosom tersebut, dimana semakin besar *fitness value* nya maka semakin besar probabilitas untuk dipilih.

    2. Dengan sebuah probabilitas $P_c$ yaitu probabilitas untuk *crossover* lakukan operasi *crossover* pada pasangan kromosom yang dipilih untuk menghasilkan dua buah keturunan. Jika tidak terjadi *crossover* maka keturunan yang dihasilkan adalah replika yang identik dengan orang tuanya. Gambar 2.5 menunjukkan *crossover* dari dua kromosom yang dipilih.




**Gambar 2.5 Proses *Crossover***

3. Mutasikan kedua keturunan pada bit tertentu dengan sebuah probabilitas $P_m$ yaitu probabilitas untuk mutasi. Kemudian tambahkan kedua keturunan tadi pada populasi baru. Gambar 2.6 menunjukkan proses mutasi dari kromosom.




**Gambar 2.6 Proses Mutasi**

- Ganti populasi yang lama dengan populasi baru
- Kembali ke langkah ke 2

Setiap iterasi ini disebut dengan generasi. Iterasi ini dilakukan sebanyak yang perlu dilakukan atau sampai menemukan kromosm yang *fitness value* nya telah memenuhi kebutuhan atau telah dianggap optimal.

# BAB III
# ANALISIS DAN PERANCANGAN

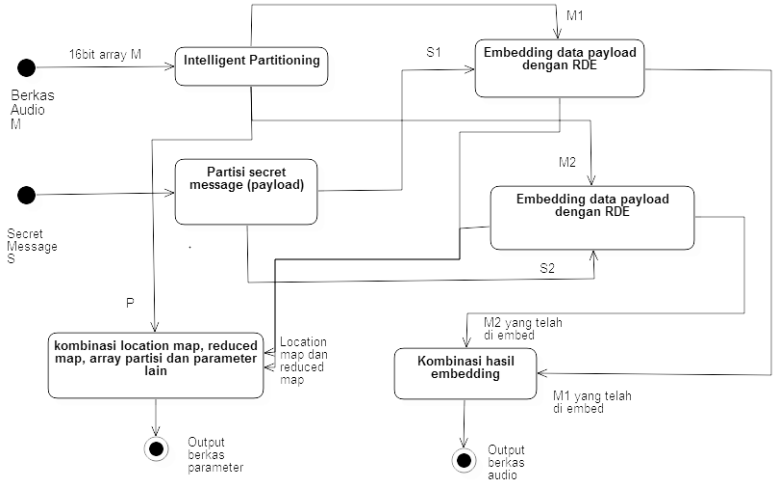
Bab ini menjelaskan tentang analisis dan perancangan aplikasi sistem untuk mencapai tujuan dari tugas akhir. Perancangan ini meliputi perancangan data, perancangan proses dan perancangan antar muka, serta juga akan dijelaskan mengenai analisis implementasi metode secara umum pada sistem

## 3.1 Analisis Sistem Secara Umum

Pada tugas akhir ini akan dibangun sebuah sistem steganografi pada berkas audio yang menggunakan metode *Reduced Difference Expansion* dengan menggunakan bahasa python dan *Scipy* sebagai pustaka bantuan. Secara garis besar proses dalam sistem ini terbagi menjadi dua yaitu proses *encoding* atau penyisipan data *payload* ke dalam berkas audio yang digunakan sebagai *cover* dan proses *decoding* yaitu proses untuk mendapatkan kembali data *payload* serta berkas audio yang dijadikan *cover* tanpa adanya informasi yang hilang.

Proses *Encoding* meliputi beberapa tahap yaitu membaca sampel pada berkas audio dan merepresentasikannya dalam bentuk *integer*, melakukan partisi pada sampel dengan *intelligent partitioning* untuk membagi sampel menjadi dua bagian sampel yang memiliki besar 8 bit, kemudian pada masing-masing bagian sampel tersebut dibagi menjadi beberapa segmen lalu pada segmen tersebut barulah dilakukan *embedding* atau penyisipan data *payload* dengan menggunakan RDE. Proses ini dapat dilakukan berkali-kali pada satu berkas audio menggunakan skema *multi-layer* sehingga kapasitas penyimpanan pada berkas dapat dimaksimalkan. Setelah proses *embedding* selesai, sistem akan menyusun kembali dua bagian sampel menjadi satu bagian utuh. Sistem kemudian menulis kembali berkas audio hasil *encoding* bersama dengan berkas tambahan yang berisikan *location map, reduced map,* serta *parameter* lain yang dibutuhkan oleh sistem dalam melakukan proses *decoding* nantinya.

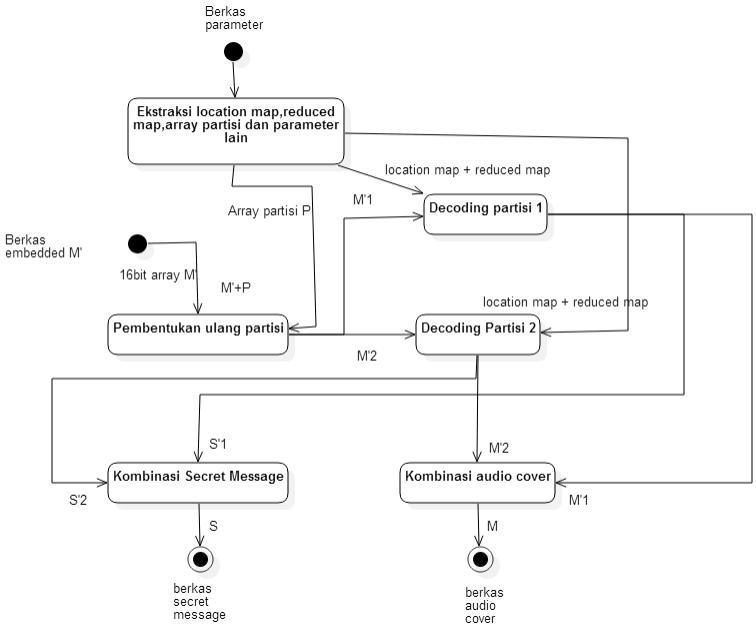
Selain melakukan *encoding* dengan parameter yang telah ditentukan, sistem dapat mencari parameter yang optimal dengan menggunakan *Genetic Algorithm.* Pencarian parameter optimal dengan GA akan dilakukan hingga menemukan parameter yang dapat memenuhi nilai target yang telah ditentukan atau sebanyak iterasi tertentu. Skema keseluruhan tahap *encoding* ditunjukkan pada Gambar 3.1.




**Gambar 3.1 Proses *Encoding***

Seperti halnya pada proses *encoding*, proses *decoding* juga terdiri dari beberapa tahap yaitu membaca sampel dari berkas audio hasil *encode* serta membaca *parameter* yang dibutuhkan untuk melakukan *decode* dari berkas yang terpisah. Parameter ini meliputi *location map*, *reduced map*, *array* partisi, jumlah *layer encoding*, dan besar segmen sampel. Kemudian sistem akan membagi kembali sampel menjadi dua bagian sesuai dengan *array* partisi yang telah dibaca. Setelah itu data *payload* akan didapatkan kembali dengan melakukan operasi pada tiap segmen sesuai dengan tanda yang dibaca pada *location map* dan *reduced map*.

Data *payload* yang telah didapatkan akan digabungkan kembali menjadi data utuh seperti semula. Kedua bagian berkas audio yang menjadi *cover* kemudian disatukan kembali membentuk berkas audio yang sama seperti semula. Skema keseluruhan proses *decoding* ditunjukkan pada Gambar 3.2




**Gambar 3.2 Proses *Decoding***

## 3.2 Perancangan Data

Pada subbab ini akan dibahas mengenai perancangan data yang merupakan bagian penting dari sistem sebagai objek yang akan menjadi masukan dan diolah oleh sistem dalam tugas akhir ini yang akhirnya akan menghasilkan keluaran.

### 3.2.1 Data Masukan

Data masukan merupakan data yang paling awal diproses oleh sistem. Pada sistem ini, data masukkan akan dibagi menjadi dua, yaitu data masukan untuk proses *encoding* dan data masukan untuk proses *decoding*. Untuk proses *encoding*, data masukan berupa berkas audio sebagai *cover*, berkas teks sebagai data *payload* dan *input* user pada aplikasi sebagai parameter. Berkas audio yang digunakan sebagai masukan memiliki ekstensi .wav tanpa adanya *tagging* karena sistem tidak dapat membaca *tag* pada berkas. Berkas audio yang menjadi masukan haruslah hanya memiliki *single channel* atau mono dan sampelnya memiliki panjang 16bit sedangkan berkas teks untuk data *payload* memiliki ekstensi .txt. Input user pada aplikasi berupa nilai-nilai yang menjadi parameter saat melakukan *Encoding* yaitu besar segmen, nilai *threshold*, dan jumlah layer.

Untuk proses *decoding*, berkas audio yang menjadi data masukan memiliki format dan spesifikasi sama dengan berkas audio untuk data masukan proses *encoding.* Berkas lainnya yang menjadi data masukan adalah berkas parameter dengan ekstensi .param yang dihasilkan pada proses *Encoding*.

### 3.2.2 Data Proses

Data proses adalah data yang digunakan selama sistem berjalan yang merupakan hasil pengolahan dari data masukan. Data yang telah diproses kemudian akan menjadi data keluaran pada tahap selanjutnya. Data proses yang digunakan pada sistem dibagi menjadi dua bagian yaitu data yang digunakan pada proses *encoding* dan data yang digunakan saat proses *decoding*.

a. Data Proses *Encoding*

Data yang digunakan saat proses *Encoding* ditunjukkan oleh Tabel 3.1.

| No. | Nama Data | Keterangan |
|---|---|---|
| 1 | Wave | Objek yang merepresentasikan berkas audio yang sedang di proses |
| 2 | Wave.samples | Data sampel dari berkas audio yang sedang di proses |
| 3 | Wave.bitrate | Bitrate dari berkas audio yang sedang di proses |
| 4 | intM1 | *Array* yang berisi sampel audio pada grup M1 setelah dilakukan partisi dalam representasi *integer* |
| 5 | intM2 | *Array* yang berisi sampel audio pada grup M2 setelah dilakukan partisi dalam representasi *integer* |
| 6 | payload_seg1 | Segmen *array* bit dari data *payload* yang akan disisipkan pada M1 |
| 7 | payload_Seg2 | segmen *array* bit dari data *payload* yang akan disisipkan pada M2 |
| 8 | threshold | Parameter batas ambang toleransi dari selisih antar piksel untuk metode RDE |
| 9 | segment_size | Parameter besar segmen yang dilakukan penyisipan saat proses RDE |

| | | |
|---|---|---|
| 10 | n_layer | Jumlah layer yang digunakan saat proses *encoding* |
| 11 | goal_value | Nilai optimal untuk GA |
| 12 | iter_limit | Batas iterasi GA |

**Tabel 3.1. Data Saat Proses Encoding**

b. Data Proses *Decoding*

Data yang digunakan saat proses *encoding* ditunjukkan oleh Tabel 3.2.

| No. | Nama Data | Keterangan |
|---|---|---|
| 1 | Wave | Objek yang merepresentasikan berkas audio yang sedang di proses |
| 2 | Wave.samples | Data sampel dari berkas audio yang sedang di proses |
| 3 | Wave.bitrate | Bitrate dari berkas audio yang sedang di proses |
| 4 | _intM1 | *Array* yang berisi sampel audio hasil *encode* pada grup M1 setelah dilakukan partisi dalam representasi *integer* |
| 5 | _intM2 | *Array* yang berisi sampel audio hasil *encode* pada grup M2 setelah dilakukan partisi dalam representasi *integer* |
| 6 | P | *Array* yang menyimpan partisi dari sampel yang akan digunakan untuk merekonstruksi ulang sampel |
| 7 | message1 | Data payload hasil *Decode* pada grup *array* _intM1 |

| | | |
|---|---|---|
| 8 | message2 | Data payload hasil *Decode* pada grup *array* _intM2 |
| 9 | Payload_sizes | *Array* yang berisi data panjang dari *payload* yang disisipkan setiap *layer* nya, akan digunakan saat melakukan *decoding* pada tiap layer |
| 10 | segment_size | Parameter besar segmen yang dibaca dari berkas parameter untuk melakukan *Decoding* |
| 11 | locmap_list | Array yang berisikan *location map* untuk tiap *layer* yang akan digunakan saat proses *decode* |
| 12 | reducemap_list | Array yang berisikan *reduce map* untuk tiap *layer* yang akan digunakan saat proses *decode* |
| 13 | n_layer | Parameter jumlah layer yang dibaca dari berkas parameter untuk melakukan *decoding* |

**Tabel 3.2. Data Saat Proses Decoding**

### 3.2.3 Data Keluaran

Data keluaran untuk proses *encoding* pada sistem ini adalah berupa berkas audio yang telah disisipi *data payload*. Ukuran dari berkas identik dengan ukuran berkas audio awal serta berkas parameter dengan format .param yang akan digunakan pada proses *decoding* nantinya. Data keluaran untuk proses *decoding* pada sistem ini adalah berkas audio yang identik dengan berkas audio sebelum dilakukan proses *encoding* tanpa ada hilangnya

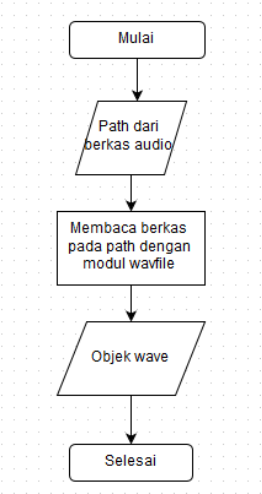
informasi sedikitpun, serta berkas dengan format .txt yang berisikan data *payload* yang disisipkan pada berkas audio.

## 3.3 Perancangan Proses

Subbab ini membahas mengenai perancangan setiap proses pada sistem untuk memberikan gambaran yang mendetil terhadap alur implementasi metode pada sistem steganografi ini.

### 3.3.1 Tahap Pembacaan Berkas Audio

Pada tahap pembacaan berkas audio digunakan bantuan modul wavfile pada pustaka *SciPy*. Modul akan membaca berkas audio dari *path* masukan user dan merubahnya ke dalam representasi objek Wave. sampel yang ada pada berkas akan dirubah tipenya dari *signed integer* 16bit menjadi *unsigned integer* 16 bit. Hal ini dilakukan agar tidak ada nilai negatif pada sampel sehingga mempermudah pemrosesan pada tahap selanjutnya. Gambar 3.3 menunjukkan diagram alir untuk tahap ini.



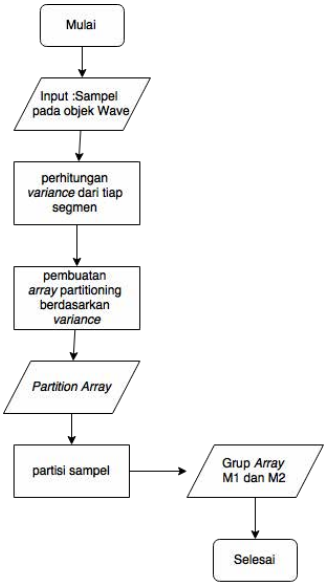

**Gambar 3.3 Diagram Alir Tahap Pembacaan Berkas Audio**

### 3.3.2 Tahap Pembacaan Data *Payload*

Pada tahap pembacaan data *payload*, data yang memiliki format .txt akan dibaca dengan menggunakan kelas *Helper* dan disimpan dalam variabel *payload* dalam bentuk string.

### 3.3.3 Tahap Partisi Sampel

Berkas audio yang telah dibaca dan direpresentasikan menjadi objek Wave memiliki atribut *sample* yang berisikan keseluruhan sampel. Sampel ini kemudian akan dipartisi dengan menggunakan metode *intelligent partition* untuk mendapatkan dua bagian *array* yang masing-masing memiliki panjang 8 bit. Diagram alir untuk tahap partisi dapat dilihat pada Gambar 3.4.




**Gambar 3.4 Diagram Alir Tahap Partisi**

### 3.3.4 Tahap Partisi *Payload*

Berkas *payload* yang telah dibaca akan dipartisi menjadi dua bagian dengan ukuran yang sama. *String* dari data *payload* akan dibagi menjadi dua buah *string* dengan ukuran sama. Masing-masing *string* ini kemudian akan disisipkan pada grup *array* yang berbeda.

### 3.3.5 Tahap *Encoding*

Di dalam tahap encoding ini dilakukan penyisipan data *payload* pada tiap grup *array* sampel yang dihasilkan dari tahap partisi sebelumnya. Bagian pertama data *payload* akan disisipkan pada grup pertama (M1) dan data *payload* sisanya akan disisipkan pada grup kedua (M2). Penyisipan dilakukan pada tiap segmen pada *array*. Besar segmen ditentukan oleh *parameter* yang didapatkan dari masukan pengguna saat menjalankan program. Kapasitas tiap segmen dipengaruhi oleh *parameter threshold* yang mnejadi ambang batas selisih tiap piksel. Kapasitas tiap segmen dapat ditentukan oleh persamaan (3.1) .

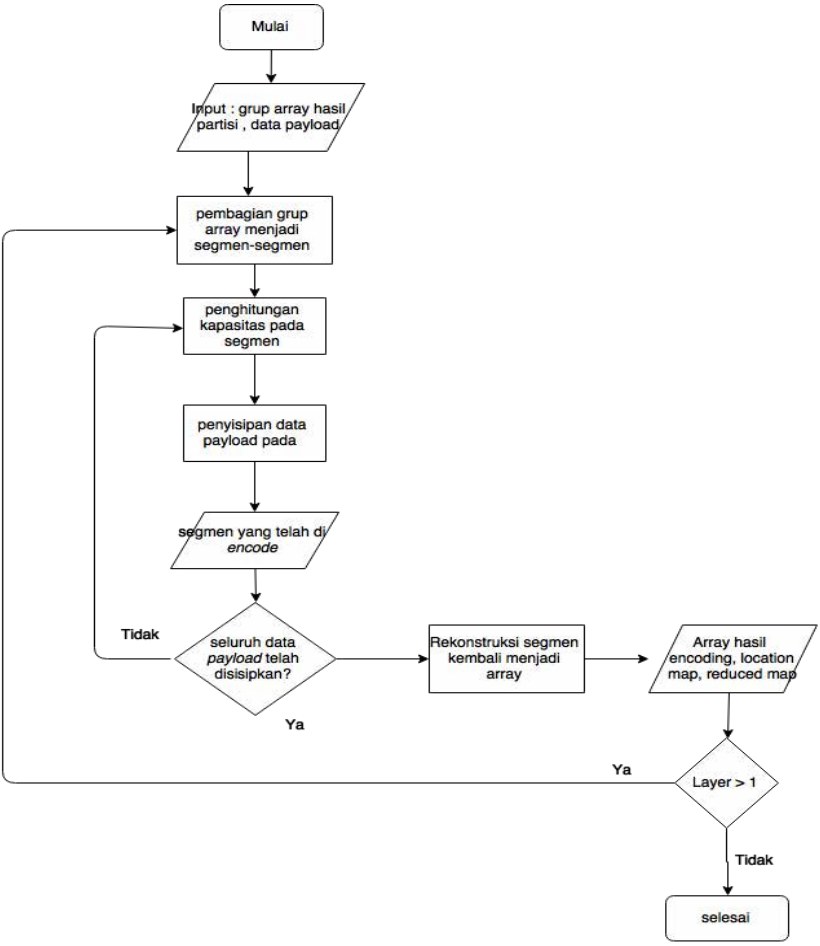
$$Cap_s = \sum_{i=2}^{n} x_i \mid 0 \leq v \leq t \; dan \; 0 \leq RDE(v) \leq 255$$

dimana **(3. 1)**

$$v = x_i - x_1$$

Dimana *n* adalah panjang segmen, *x* adalah sampel pada segmen, *v* adalah selisih antara sampel ke *i* dan sampel pertama dan *t* adalah nilai dari *threshold*, sehingga kapasitas maksimum pada tiap segmen terjadi ketika pada tiap sampel dapat dilakukan penyisipan kecuali pada sampel pertama yaitu sebesar *n-1*. Sampel pertama pada tiap segmen tidak digunakan untuk penyisipan

karena akan digunakan sebagai acuan dalam proses pengembalian nilai tiap sampel saat proses *decoding*. Gambar 3.5 menunjukkan diagram alir untuk tahap *encoding.*



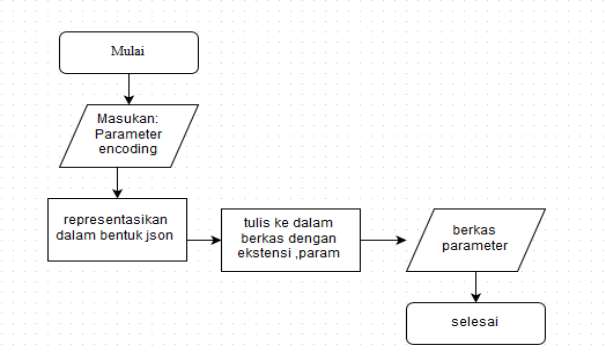

**Gambar 3.5 Tahap Encoding**

Proses penyisipan dilakukan dengan menyisipkan 1 bit dari data *payload* pada tiap sampel menggunakan metode RDE sampai kapasitas pada segmen itu terpenuhi. Proses ini dilakukan hingga semua data

*payload* disisipkan. Setiap sampel yang disisipkan data ataupun tidak akan diberi tanda yang akan disimpan pada *location map*, begitu juga dengan sampel yang selisihnya dilakukan reduksi ataupun tidak akan diberi tanda pada *reduced map*. Keseluruhan proses *encoding* ini dapat dilakukan beberapa kali dengan memanfaatkan *multi-layer encoding* yaitu pada sampel hasil penyisipan dilakukan proses *encoding* kembali. Pengguna dapat menentukan jumlah dari *layer* yang akan digunakan saat proses *encoding.*

### 3.3.6 Tahap Penulisan Berkas Parameter

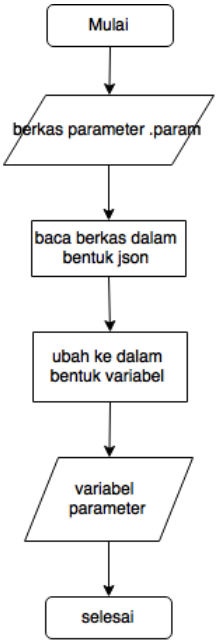
Pada tahap ini *parameter* yang digunakan dalam proses *encoding* dituliskan kedalam berkas dengan ekstensi .param. Parameter yang akan ditulis kedalam berkas meliputi besar segmen dari sampel, jumlah layer yang digunakan, besar data *payload* pada tiap layer, *location map* pada tiap layer, *reduced map pada tiap layer* dan *array* partisi Diagram alir untuk tahap penulisan berkas parameter ditunjukkan pada Gambar 3.6.




**Gambar 3.6 Tahap Penulisan Berkas Parameter**

### 3.3.7 Tahap Pembacaan Data Parameter

Pada tahap ini, berkas dengan ekstensi .param akan dibaca untuk dijadikan masukan pada proses *decoding*. Data-data *parameter* pada berkas ini diperlukan agar sistem dapat mengembalikan data *payload* maupun *cover* seperti semula tanpa adanya informasi yang hilang. Data parameter yang telah dibaca dari berkas memiliki format json, format ini kemudian diubah kembali menjadi bentuk variabel dalam bahasa pemrograman python. Gambar 3.7 menunjukkan diagram alir untuk tahap ini.



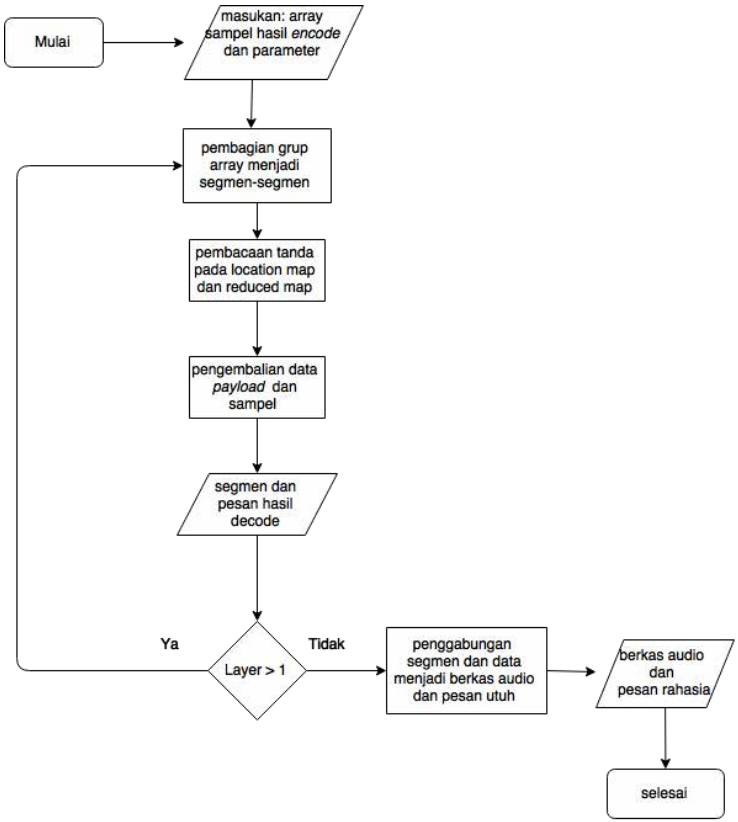

**Gambar 3.7 Tahap Pembacaan Data Parameter**

### 3.3.8 Tahap *Decoding*

Pada tahap ini data *payload* dan berkas audio asli akan dikembalikan dari berkas audio hasil tahap *encoding*, sedangkan parameter yang diperlukan untuk melakukan proses *decoding* didapatkan dari berkas parameter yang dihasilkan pula pada tahap *encoding*. Parameter yang digunakan dalam tahap *decoding* ini adalah besar segmen, jumlah *layer, array* partisi, *list location map* pada tiap *layer*, *list reduce map* pada tiap *layer* dan besar *payload* yang disisipkan pada tiap *layer*.

Menggunakan parameter yang telah didapatkan dari berkas parameter, *array* sampel pada berkas yang telah di *encode* akan dipartisi dengan menggunakan *array* partisi sehingga terbentuk kembali dua kelompok *array* M1 dan M2 yang urutannya sama dengan kelompok *array* saat tahap *encoding*. Setiap grup *array* kemudian akan dibagi menjadi segmen-segmen dengan ukuran sesuai dengan parameter yang telah didapatkan.

Data *payload* lalu didapatkan dari tiap sampel pada segmen dengan sebelumnya dilihat terlebih dahulu tanda untuk sampel tersebut pada *location map* dan *reduced map*, dimana pada tiap sampel akan dilakukan operasi sesuai dengan tanda yang ada pada *location map* dan *reduced map*. Tiap bit data *payload* yang telah didapatkan akan digabungkan menjadi satu keutuhan pesan dan sampel yang telah dikembalikan nilainya akan digabungkan kembali menjadi satu keutuhan *array* sampel, Proses ini dilakukan sebanyak jumlah layer yang nilainya telah didapatkan pada proses pembacaan data parameter dari berkas. Berkas parameter yang digunakan pada proses ini haruslah sama dengan berkas yang dihasilkan dari proses *encoding* karena jika parameter yang digunakan berbeda dengan parameter yang digunakan saat proses *encoding*, proses ini tidak akan dapat berjalan secara sempurna. Diagram alir tahap ini dapat dilihat pada Gambar 3.8.



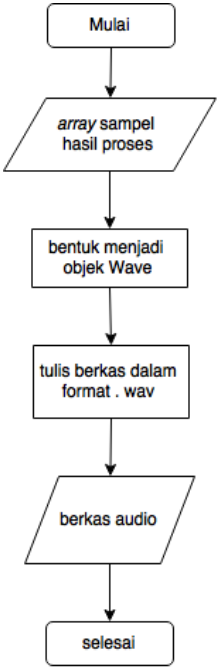

**Gambar 3.8. Tahap *Decoding***

### 3.3.9 Tahap Penulisan Data *Payload*

Pada tahap ini dilakukan penulisan data payload hasil dari tahap *decoding*. Data payload pada sistem yang bertipe *string* akan dituliskan ke dalam sebuah berkas dengan format .txt, karena tidak adanya informasi yang hilang maka isi maupun representasi data akan tetap terjaga dan identik dengan berkas asli.

### 3.3.10 Tahap Penulisan Berkas Audio

Pada tahap ini, berkas audio hasil *encoding* ataupun *decoding* dituliskan kembali ke dalam berkas audio dengan ekstensi .wav. Berkas audio baru yang ditulis akan memiliki *bitrate* dan jumlah sampel yang sama dengan berkas audio asli. Sebelum dilakukan penulisan, *array* sampel hasil proses yang memilikir tipe data *unsigned integer* 16 bit akan dirubah menjadi *signed integer* 16 bit. *Array* sampel kemudian dibentuk menjadi objek Wave yang kemudian ditulis dengan menggunakan bantuan modul wavfile pada pustaka *SciPy*. Diagram alir untuk tahap ini ditunjukkan oleh Gambar 3.9.



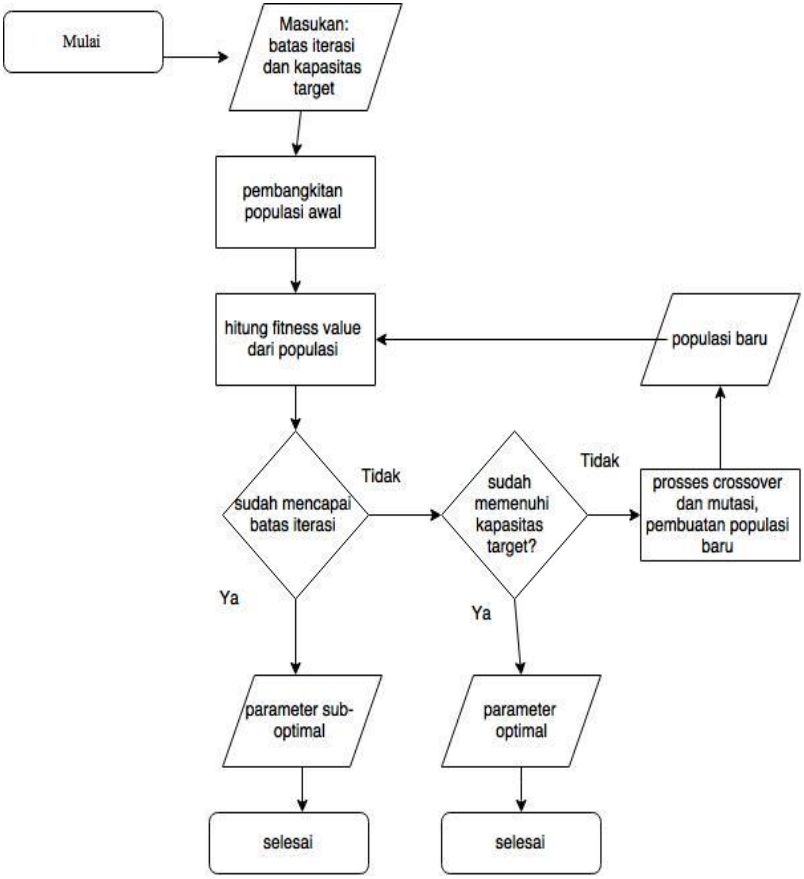

**Gambar 3.9 Tahap Penulisan Berkas Audio**

### 3.3.11 Tahap Pencarian Parameter Optimal dengan GA

Tahap ini adalah tahap opsional jika parameter yang digunakan dalam tahap *encoding* tidak didapatkan secara manual melalui masukan user melainkan didapatkan oleh sistem secara otomatis menggunakan GA. Pada tahap ini sistem akan membangkitkan populasi untuk GA yang berisikan parameter untuk proses *encoding* yaitu besar segmen, nilai threshold dan jumlah layer. Parameter ini direpresentasikan dalam bentuk *string* dimana masing-masing parameter memiliki batas nilai yaitu, parameter besar segmen memiliki panjang 8 bit yang berarti memiliki nilai maksimal 255, parameter jumlah layer memiliki panjang 3 bit yang berarti memiliki nilai maksimal 8 dan parameter besar segmen memiliki panjang 7 bit yang berarti memiliki nilai maksimal 128. String kromosom yang merepresentasikan semua parameter ini pada akhirnya akan memiliki panjang 20 bit.

Sistem kemudian akan membangkitkan populasi dengan cara melakukan pembuatan *string* kromosom hingga mencapai jumlah populasi. Sistem kemudian mencoba melakukan *encoding* dengan parameter yang telah dibangkitkan dan membandingkan hasilnya. Jika hasilnya belum mencapai nilai optimal maka sistem akan melakukan proses *crossover* dan mutasi dari populasi yang ada untuk membentuk populasi baru yang memiliki *fitness value* lebih baik dari populasi awal. *Fitness value* dihitung dengan membandingkan kapasitas dari hasil *encoding* menggunakan parameter dari masing-masing kromosom.

Seteleh populasi baru terbentuk, sistem akan melakukan *encoding* kembali. Hal ini terus dilakukan hingga hasil dari *encoding* mencapai nilai optimal atau mencapai batas iterasi. Nilai optimal yang ingin dicapai dan batas iterasi didapatkan melalui masukan user. Diagram alir untuk tahap ini ditunjukkan oleh Gambar 3.10.




**Gambar 3.10 Tahap Pencarian Parameter Optimal dengan GA**

# BAB IV
# IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan sistem. Bahasa pemrograman yang digunakan untuk implementasi adalah python dengan pustaka *SciPy*.

## 4.1 Lingkungan Implementasi

Lingkungan implementasi tugas akhir dijelaskan pada Tabel 4.1.

| | |
|---|---|
| Perangkat Keras | Prosesor :<br>- Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz<br>Memori :<br>- 6 GB |
| Perangkat Lunak | Sistem Operasi :<br>- Microsoft Windows 7 Ultimate 64-bit<br>Perangkat Pengembang :<br>- Visual Studio 2013 dengan Python tools for Visual Studio |

**Tabel 4.1 Lingkungan Implementasi Perangkat Lunak**

## 4.2 Implementasi Proses

Implementasi Proses dilakukan berdasarkan perancangan proses yang telah dijelaskan pada bab analisis dan perancangan.

### 4.2.1 Implementasi Tahap Pembacaan Berkas Audio

Pada subbab ini akan dibahas implementasi untuk tahap pembacaan berkas audio, untuk membaca berkas audio digunakan kelas Helper.WavIO yang di dalamnya terdapat implementasi dari modul wavfile pada pustaka *scipy*. Pada modul wavfile telah tersedia fungsi untuk membaca berkas audio dengan format wav menjadi sebuah array *numpy* yang berisikan nilai dari setiap sampel

pada berkas fungsi ini kemudian akan mengembalikan objek Wave. Kode sumber untuk implementasi tahap ini dapat dilihat pada Gambar 4.1.

```
FUNCTION open_wav (String path)
    SET wavefile AS Wave_Object
    READ object FROM path
    wavefile.sample = object.sample
    wavefile.bitrate = object.bitrate
    RETURN wavefile
END FUNCTION
```

**Gambar 4.1 *Pseudocode* Implementasi Pembacaan Berkas Audio**

### 4.2.2 Implementasi Tahap Pembacaan Data Payload

Pada subbab ini akan dibahas implementasi untuk tahap pembacaan data payload. Untuk membaca data payload digunakan kelas Helper.PayloadIO pada kelas ini terdapat fungsi untuk membaca berkas *payload* dengan ekstensi .txt kedalam sebuah *string*. Kode sumber pada Gambar 4.2 menunjukkan implementasi dari tahap ini.

```
FUNCTION open_payload (String path)
    SET payload AS string
    READ object FROM path
    payload = object.text
    RETURN payload
END FUNCTION
```

**Gambar 4.2 *Pseudocode* Implementasi Pembacaan Data Payload**

### 4.2.3 Implementasi Tahap Partisi Sampel

Subbab ini akan membahas implementasi dari tahap partisi sampel audio yang dilakukan sebelum proses *encoding*. Sebelum dilakukan partisi, sampel yang tadinya berbentuk *unsigned integer* 16 bit harus direpresentasikan dalam bentuk biner hal ini ditunjukkan pada kode sumber pada Gambar 4.3.

```
FUNCTION num_to_binary (array numArr)
    SET binaries AS array
    FOR ALL number IN numArr
        bin = asBinary(number)
        PUSH bin TO binaries
    RETURN binaries
END FUNCTION
```

**Gambar 4.3 *Pseudocode* Implementasi Konversi Integer ke Biner**

*Array* sampel yang telah direpresentasikan dalam biner kemudian akan ditransformasikan ke dalam *bigit* dimana 1 bigit adalah 1 kolom dari *array* sampel. Pembentukan bigit dapat dilihat pada kode sumber pada Gambar 4.4.

```
FUNCTION make_bigits (matrice sampleArray)
    SET bigits AS matrice
    FOR ALL column IN sampleArray
        bigit = transpose(column)
        PUSH bigit TO bigits
    RETURN bigits
END FUNCTION
```

**Gambar 4.4 *Pseudocode* Implementasi Pembuatan *Bigit***

Kemudian *variance* dari setiap *bigit* akan dihitung untuk menjadi acuan pembuatan *array* partisi, setelah *array* partisi

terbentuk barulah dilakukan partisi *array* sampel menjadi dua bagian yaitu M1 dan M2. Proses penghitungan *variance* dan partisi *array* ditunjukkan oleh Kode sumber pada Gambar 4.5.

```
FUNCTION intelligent_partition (array bigits, matrice
audio_sample)
    SET variances AS array
    SET partition_array AS array
    SET groupA AS array
    SET groupB AS array

    FOR ALL bigit IN bigits
        v = calculate_variance(bigit)
        PUSH v TO variances

    SORT DESCENDING variances
    FOR bigit IN bigits
        IF idx OF bigit = 8
            PUSH bigit TO groupA
            SET partition_array INDEX OF 8 AS 0
        IF idx OF bigit = 10
            PUSH bigit TO groupA
            SET partition_array INDEX OF 10 AS 0
        IF idx OF bigit = 12
            PUSH bigit TO groupA
            SET partition_array INDEX OF 12 AS 0
        IF idx OF bigit = 14
            PUSH bigit TO groupA
            SET partition_array INDEX OF 14 AS 0
        IF idx OF bigit = 0
            PUSH bigit TO groupA
            SET partition_array INDEX OF 0 AS 0
        IF idx OF bigit = 2
            PUSH bigit TO groupA
            SET partition_array INDEX OF 2 AS 0

     FOR bigit IN bigits
        IF idx OF bigit = 9
            PUSH bigit TO groupB
```

```
            SET partition_array INDEX OF 9 AS 0
        IF idx OF bigit = 11
            PUSH bigit TO groupB
            SET partition_array INDEX OF 11 AS 0
        IF idx OF bigit = 13
            PUSH bigit TO groupB
            SET partition_array INDEX OF 13 AS 0
        IF idx OF bigit = 15
            PUSH bigit TO groupB
            SET partition_array INDEX OF 15 AS 0
        IF idx OF bigit = 1
            PUSH bigit TO groupB
            SET partition_array INDEX OF 0 AS 0
        IF idx OF bigit = 3
            PUSH bigit TO groupB
            SET partition_array INDEX OF 2 AS 0

    SPLIT remaining_bigits
    PUSH remaining_bigits_1 TO groupA
    SET partition_array INDEX OF remaining_bigits_1
AS 1
    PUSH remaining_bigits_2 TO groupB
    SET partition_array INDEX OF remaining_bigits_2
AS 0

    RETURN groupA,groupB,partition_array

END FUNCTION
```

**Gambar 4.5 *Pseudocode* Implementasi Penghitungan Variance dan Partisi Sampel**

### 4.2.4 Implementasi Tahap Partisi *Payload*.

Subbab ini menjelaskan mengenai implementasi tahap partisi *payload*. Pada tahap ini, *payload* yang telah direpresentasikan kedalam bentuk *string* akan dipartisi kedalam dua bagian yaitu *payload* segmen 1 dan *payload* segmen 2, kode

sumber pada Gambar 4.6. menunjukkan implementasi dari partisi *payload*.

```
FUNCTION split_payload(string payload)
    SET payload_seg_1 AS string
    SET payload_seg_2 AS string
    payload_len = SIZE OF payload
    payload_seg_1 = payload INDEX OF 1 TO
payload_len/2
    payload_seg_2 = payload INDEX OF payload/len2
TO payload_len
    RETURN payload_seg_1, payload_seg_2
END FUNCTION
```

**Gambar 4.6 *Pseudocode* Implementasi Partisi *Payload***

### 4.2.5 Implementasi Tahap *Encoding*

Subbab ini membahas implementasi pada tahap *encoding*. Sebelum melakukan *encoding*, *array* sampel hasil partisi yang masih direpresentasikan dalam biner akan dirubah kembali bentuknya menjadi *unsigned integer* 16 bit. Operasi konversi dilakukan untuk kedua grup M1 dan M2. Kode sumber pada Gambar 4.7. menunjukkan implementasi dari konversi biner ke integer.

```
FUNCTION binary_to_num(array binaryArr)
   SET numArr AS array

   FOR ALL bin IN  binaryArr
        num = convert_to_number(bin)
        PUSH num TO numArr
```

```
   RETURN numArr
END FUNCTION
```

**Gambar 4.7 Kode Implementasi Konversi Biner ke Integer**

#### 4.2.5.1 Implementasi Pembagian Sampel Menjadi Segmen

Langkah berikutnya adalah membagi *array* sampel menjadi segmen-segmen dengan panjang tiap segmen sesuai dengan parameter yang digunakan. Implementasi dari tahap ini dapat dilihat pada kode sumber pada Gambar 4.8.

```
FUNCTION split_to_segment(array sampleArray)
   SET sample_len AS SIZE OF sampleArray
   SET segmented AS array
   SET idx AS 0
   FOR ALL sample IN  sampleArray
        segment = sample INDEX OF idx TO
idx+segment_len
        PUSH segment TO segmented
        idx = idx+segment_len
   RETURN segmented
END FUNCTION
```

**Gambar 4.8. *Pseudocode* Implementasi Pembagian Segmen**

#### 4.2.5.2 Implementasi Pengecekan Segmen

Setiap segmen yang akan dilakukan *encoding* akan dlakukan pengecekan kapasitas terlebih dahulu untuk menentukan apakah pada segmen tersebut dapat dilakukan *encoding*.

Pengecekan Kapasitas yang dilakukan mencakup pengecekan *threshold* dan pengecekan terhadap *underflow* dan *overflow* seperti yang dijabarkan pada subbab 3.3.5. Kode sumber untuk melakukan pengecekan kapasitas ditunjukkan oleh Gambar 4.9, kode sumber untuk melakukan pengecekan *threshold* ditunjukkan oleh Gambar 4.10, dan kode sumber untuk melakukan pengecekan *overflow* dan *underflow* ditunjukkan oleh Gambar 4.11.

```
FUNCTION check_capacity(array sampleArray)
   SET capacity AS 0
   FOR ALL segment IN  sampleArray
        IF check_block(segment) IS TRUE AND
check_threshold(segment) IS TRUE
        capacity = capacity + SIZE OF segment
   RETURN capacity
END FUNCTION
```

**Gambar 4.9. *Pseudocode* Implementasi Pengecekan Kapasitas**

```
FUNCTION check_threshold(array sampleArray)
   SET flag AS TRUE
   FOR ALL segment IN  sampleArray
        diff = segment - sampleArray INDEX OF 1
        IF diff > threshold
            flag = FALSE
   RETURN flag
END FUNCTION
```

**Gambar 4.10. *Pseudocode* Implementasi Pengecekan Threshold**

```
FUNCTION check_block(array sampleArray)
   SET flag AS TRUE
   FOR ALL segment IN sampleArray
        diff = segment - sampleArray INDEX OF 1
        IF sampleArray INDEX OF 1 + diff > 255
            flag = FALSE
        IF sampleArray INDEX OF 1 + diff < 0
            flag = FALSE
   RETURN flag
END FUNCTION
```

**Gambar 4.11. *Pseudocode* Implementasi Pengecekan Overflow dan Underflow**

Setiap segmen yang telah dilakukan pengecekan akan diberi tanda pada *location map* sesuai hasilnya. Jika dapat dilakukan *encoding* pada segmen maka pada *location map* akan diberikan tanda 0, jika segmen dapat dilakukan *encode* dengan reduksi maka pada *location map* akan diberi tanda 1 dan jika segmen tidak dapat dilakukan *encode* maka pada *location map* akan diberi tanda 2.

#### 4.2.5.3 Implementasi *Improved* RDE

Pada segmen dimana dapat dilakukan *encoding* selisih antara piksel akan direduksi menggunakan RDE. Kemudian proses *encoding* akan dilanjutkan dengan menyisipkan bit pada selisih hasil RDE, operasi yang dilakukan untuk penyisipan bergantung pada tanda yang digunakan pada *reduced map* seperti yang telah dijelaskan pada subbab 2.4. Implementasi *Improved* RDE dapat dilihat pada kode sumber pada Gambar 4.12. Gambar 4.13 menunjukkan proses penyisipan bit pada selisih hasil RDE.

```
FUNCTION RDEI(integer diff)
   IF diff >3
        rd = diff -  POW(2,FLOOR(LOG(diff,2))-1)  -
POW(2,FLOOR(LOG(angka,2))-2)
        flag = 0
        IF POW(2,FLOOR(LOG(diff,2)) /
POW(2,(FLOOR(LOG(rd,2))) == 4
            flag = 1
        IF POW(2,FLOOR(LOG(diff,2)) /
POW(2,(FLOOR(LOG(rd,2))) == 1
            flag = 2
    ELSE
        rd = diff
    RETURN rd,flag
END FUNCTION
```

**Gambar 4.12. *Pseudocode* Implementasi *Improved RDE***

```
FUNCTION insert_bit(array sampleArray,array
payload)
    SET encoded AS array
    SET location_map AS array
    SET sample_len AS SIZE OF sampleArray

    FOR ALL sample IN sampleArray
        diff = sample - sampleArray INDEX OF 1
        rdei,flag = rdei(diff)
        PUSH flag TO location_map

    FOR idx IN sample_len
        IF idx < sample_len
            IF flag=0
                ri = 2*rdei+payload INDEX OF idx
            ELSE IF flag =1
                ri = 2*FLOOR(rdei/2) + payload
INDEX OF idx
            ELSE
```

```
                IF flag = 0
                    ri =  2*rdei
                ELSE IF flag =1
                    ri =  2*FLOOR(rdei/2)
            ui = sampleArray INDEX OF 1 + ri
            PUSH ui TO encoded

    RETURN encoded,location_map

END FUNCTION
```

**Gambar 4.13. *Pseudocode* Implementasi Penyisipan Bit**

#### 4.2.5.4 Implementasi Pencarian Parameter Optimal dengan GA

Pada subbab ini dibahas implementasi dari penggunaan GA untuk mencari parameter optimal untuk proses *encoding.* Selain menggunakan parameter yang didapat dari masukan pengguna, sistem dapat menggunakan GA untuk menemukan parameter yang optimal untuk proses *encoding*. Proses ini dibagi menjadi beberapa tahap yaitu pembentukan populasi awal, penghitungan *fitness value*, *crossover* populasi, mutasi populasi dan pembentukan populasi baru.

##### 4.2.5.4.1 Pembentukan Populasi Awal

Pembentukan awal pada populasi GA dilakukan dengan membangkitkan *string* acak sebanyak jumlah populasi yang diperlukan. *String* ini merepresentasikan parameter-parameter yang digunakan pada *encoding*. Gambar 4.14 menunjukkan kode sumber dari proses ini.

```
FUNCTION generate_population(integer
chromosome_length,integer population_count)
    SET population AS array
    WHILE SIZE OF population < population_count
        SET kromosom AS string
        WHILE SIZE OF kromosom < chromosome_length
            APPEND RANDOM (0,1) TO kromosom
        PUSH kromosom TO population
    RETURN population
END FUNCTION
```

**Gambar 4.14. *Pseudocode* Implementasi Pembentukan Populasi Awal**

#### 4.2.5.4.2 Perhitungan Fitness Value

Setiap populasi yang telah dibuat akan dihitung *fitness value* nya, pada kasus kali ini *fitness value* dari tiap kromosom adalah kapasitas dari hasil *encoding*. Jika salah satu dari kromosom memiliki *fitness value* yang mencapai target maka proses akan berhenti. Jika batas iterasi telah dicapai tetapi belum ditemukan kromosom *fitness value* yang mencapai target maka diambil kromosom dengan *fitness value* yang paling besar. Kode sumber pada Gambar 4.15 menunjukkan implementasi dari proses perhitungan ini.

```
FUNCTION calculate_fitness(array populasi, integer
target)
    SET best_solution 0
    SET fitness_val AS array
    FOR ALL kromosom IN populasi
      parameter = get_parameter(kromosom)
      result = encode(parameter)
      PUSH result TO fitness_val
      IF result > best_solution
        best_solution = result
      IF best_solution >= target
        BREAK
```

```
    RETURN fitness_val
END FUNCTION
```

**Gambar 4.15. *Pseudocode* Implementasi Perhitungan *Fitness Value***

#### 4.2.5.4.3 Crossover Populasi

Setelah *fitness value* untuk populasi dihitung, maka akan dipilih dua buah kromosom dari populasi untuk dilakukan *cross over*, hal ini bertujuan untuk menghasilkan kromosom baru yang diharapkan mempunyai *fitness value* yang lebih baik daripada kromosom pada generasi sebelumnya. Pemilihan kromosom yang digunakan untuk *crossover* dipengaruhi oleh *fitness value* nya. Jika *fitness valuen* nya besaar maka kemungkinan dipilih untuk *crossover* akan semakin besar. Kode sumber pada Gambar 4.16 menunjukkan implentasi dari proses *crossover.*

```
FUNCTION crossover_chromosome (string kromosom_1 ,
string kromosom_2)
   SET idx AS RANDOM(0,SIZE OF kromosom)
   REPLACE (kromsom_1 INDEX OF idx TO SIZE OF
KROMOSOM,kromsom_2 INDEX OF idx TO SIZE OF kromosom)
   REPLACE (kromsom_2 INDEX OF idx TO SIZE OF
kromosom,kromsom_1 INDEX OF idx TO SIZE OF kromosom)
   RETURN kromosom_1 , kromosom_2
END FUNCTION
```

**Gambar 4.16 *Pseudocode* Implementasi Crossover Populasi**

#### 4.2.5.4.4 Mutasi Populasi

Kromosom baru hasil *crossover* akan memiliki kemungkinan kecil untuk melakukan mutasi. Mutasi pada kromosom dilakukan dengan mengganti nilai bit pada indeks yang dipilih secara acak dari 1 menjadi 0 atau sebaliknya. Hal ini

dilakukan untuk memberikan variasi pada populasi. Kode sumber pada Gambar 4.17 menunjukkan implementasi dari proses mutasi .

```
FUNCTION mutation_chromosome (string kromosom)
   SET idx AS RANDOM(0,SIZE OF kromosom)
   FLIP BIT kromosom INDEX OF idx
   RETURN kromosom
END FUNCTION
```

**Gambar 4.17 *Pseudocode* Implementasi Mutasi Pada Kromosom**

#### 4.2.5.4.5 Pembentukan Populasi Baru

Dengan menggunakan kromosom baru hasil proses *crossover* dan *mutasi*, populasi baru akan dibentuk sampai jumlah kromosom pada populasi baru tersebut sama dengan jumlah populasi awal. Kemudian populasi baru tersebut akan dihitung kembali *fitness value* nya dan keseluruhan proses akan diulang kembali. Kode sumber pada Gambar 4.18 menunjukkan implementasi dari proses pembentukan populasi baru

```
FUNCTION create_new_population (array
old_population)
    SET new_population AS array
    WHILE SIZE OF new_population < old_population
        parents = selection(old_population)
        new_childs = crossover(parents)
        prob = RANDOM(0,1000)
        IF prob = 5
            mutate(new_childs)
        PUSH new_childs TO new_population
   RETURN kromosom
END FUNCTION
```

**Gambar 4.18 *Pseudocode* Implementasi Pembentukan Populasi Baru**

### 4.2.6 Implementasi Penulisan Berkas Parameter

Pada Subbab ini akan dibahas implementasi dari penulisan berkas parameter. Setelah proses *encoding* selesai, parameter yang digunakan untuk *encoding* akan ditulis kedalam suatu berkas dengan ekstensi .param. Setiap variabel parameter akan diformat menjadi json kemudian ditulis kedalam berkas. Kode sumber pada Gambar 4.19 menunjukkan implementasi dari penulisan berkas parameter.

```
FUNCTION write_parameter (string path)
    OPEN path IN WRITE MODE
    json_parameter = JSON_DUMP(parameter)
    WRITE(PATH,json_parameter)
END FUNCTION
```

**Gambar 4.19. *Pseudocode* Implementasi penulisan berkas parameter**

### 4.2.7 Implementasi Tahap Pembacaan Data Parameter

Pada subbab ini akan dibahas implementasi dari tahap pembacaan data parameter. Untuk melakukan *decoding*, diperlukan berkas parameter yang akan memberikan informasi mengenai parameter apa yang digunakan agar *decoding* dapat dilakukan dengan tepat. Gambar 4.20 menunjukkan Kode sumber untuk tahap pembacaan data parameter.

```
FUNCTION open_parameter (string path)
    OPEN path IN READ MODE
    parameters = JSON_LOAD(path)
    RETURN parameters
END FUNCTION
```

**Gambar 4.20. *Pseudocode* Implementasi Pembacaan Data Parameter**

### 4.2.8 Implementasi Tahap *Decoding*

Subbab ini membahas implementasi pada tahap *decoding*. Sebelum melakukan *decoding*, *array* sampel hasil partisi hasil *encode* yang masih direpresentasikan dalam biner akan dirubah kembali bentuknya menjadi *unsigned integer* 16 bit, operasi konversi dilakukan untuk kedua. Proses konversi yang dilakukan identik dengan proses konversi pada *encoding*.

#### 4.2.8.1 Implementasi Pembagian Sampel Encoded Menjadi Segmen

Langkah berikutnya adalah membagi *array* sampel yang telah di *encode* menjadi segmen-segmen dengan panjang tiap segmen sesuai dengan parameter yang telah dibaca dari berkas parameter pada proses sebelumnya. Berkas parameter yang digunakan harus sesuai karena jika besar segmen yang digunakan tidak sama maka proses *decoding* akan gagal. Implementasi dari tahap ini dapat dilihat pada kode sumber pada Gambar 4.21

```
FUNCTION split_to_segment(array sampleArray)
   SET sample_len AS SIZE OF sampleArray
   SET segmented AS array
   SET idx AS 0
   FOR ALL sample IN  sampleArray
        segment = sample INDEX OF idx TO
idx+segment_len
        PUSH segment TO segmented
        idx = idx+segment_len
   RETURN segmented
END FUNCTION
```

**Gambar 4.21. *Pseudocode* Implementasi Pembagian Sampel Encoded Menjadi Segmen**

#### 4.2.8.2 Implementasi Pembacaan Tanda Dari *Location Map*

Untuk mengetahui operasi apa yang akan digunakan untuk men-*decode* suatu segmen, maka sistem perlu membaca tanda untuk segmen tersebut dari *location map*, jika pada *location map* bertanda 0 maka segmen tersebut dapat dilakukan *decode*, jika tanda nya adalah 1 maka segmen tersebut dapat di *decode* dengan dikembalikan terlebih dahulu selisih aslinya, sedangkan jika tanda pada *location map* adalah 2 maka segmen tersebut tidak dapat di *decode* dan nilai sampelnya tidak diubah. Gambar 4.22 menunjukkan kode sumber untuk implementasi pengecekan tanda pada *location map* dan Gambar 4.23 menunjukkan kode sumber untuk pemilihan operasi yang dilakukan berdasarkan tanda pada *location map.*

```
FUNCTION read_locmap (array location_map)
    SET flag_counter AS idx
    IF location_map INDEX OF flag_counter =0
        w,m = decode_block(segment,location_map
INDEX OF flag_counter
    RETURN decoded
END FUNCTION
```

**Gambar 4.22 *Pseudocode* Implementasi Pengecekan Tanda Pada *Location Map***

```
FUNCTION check_locmap (integer flag)
    SET rdiff AS integer
    IF flag = 0
            rdiff = (diff-w)/2
    ELSE IF flag=1
            rdiff = (diff-w)
```

```
    RETURN rdiff
END FUNCTION
```

**Gambar 4.23 *Pseudocode* Implementasi Pemilihan Operasi Berdasarkan Tanda *Location Map***

#### 4.2.8.3 Implementasi Invers RDE dan Pengembalian Data Payload

Untuk setiap segmen yang dapat di *decode* maka setiap sampelnya akan dilakukan pengecekan tanda pada *reduced map* untuk mengetahui operasi yang harus dilakukan dalam mengembalikan selisihnya seperti semula seperti yang telah di jabarkan pada subbab 2.4.

Setelah selisih dari sampel telah dikembalikan maka setiap bit data *payload* dapat dikembalikan dengan proses invers RDE. Gambar 4.24 menunjukkan kode sumber untuk pengecekan tanda pada *reduced map* dan Gambar 4.25 menunjukkan proses pengembalian data *payload*.

```
FUNCTION inv_rdei (integer input,integer flag)
   IF flag =1
        rd = input +
POW(2,(FLOOR(LOG(input,2))))  +
POW(2,(FLOOR(LOG(input,2))+1))

   ELSE IF flag =2
           rd = input +
POW(2,(FLOOR(LOG(input,2))-1))  +
POW(2,(FLOOR(LOG(input,2))-2))

   ELSE IF flag =0
           rd = input +
POW(2,(FLOOR(LOG(input,2))-1))  +
POW(2,(FLOOR(LOG(input,2))))
    ELSE
           rd =input
```

```
    RETURN rd
END FUNCTION
```

**Gambar 4.24. *Pseudocode* Implementasi Pengecekan Tanda Pada *Reduced Map***

```
FUNCTION reverse_embed(array sampleArray)
   FOR ALL sample IN sampleArray
        diff = sample - sampleArray INDEX OF 1
        w = 2*f(x)-x
   RETURN w
END FUNCTION
```

**Gambar 4.25. *Pseudocode* Implementasi Pengembalian Bit Data *Payload***

### 4.2.9 Implementasi Tahap Penulisan Data Payload

Subbab ini membahas mengenai implemetasi dari tahap penulisan data *payload*. Data *payload* yang telah didapatkan dari proses *decoding* masih berbentuk biner, oleh karena itu dilakukan proses konversi dari biner kembali menjadi *string*. Setelah itu data *payload* yang telah berbentuk *string* akan ditulis kedalam berkas berekstensi .txt. Implementasi untuk konversi data *payload* ditunjukkan oleh kode sumber pada Gambar 4.26 dan implementasi untuk penulisan berkas ditunjukkan oleh kode sumber pada Gambar 4.27.

```
FUNCTION binary_to_string(array binaries)
   SET hex_array AS array
   SET message AS string
   hex_array = BINARY_TO_HEX(binaries)
   message = HEX_DECODE(hex_array
```

```
   RETURN message
END FUNCTION
```

**Gambar 4.26. *Pseudocode* Implementasi Konversi Biner ke String**

```
FUNCTION write_message(string path, string message)
   OPEN path IN WRITE MODE
   WRITE(path, message)
END FUNCTION
```

**Gambar 4.27. *Pseudocode* Implementasi Penulisan Berkas Payload**

### 4.2.10 Implementasi Tahap Penulisan Berkas Audio

Pada subbab ini akan dibahas implementasi dari tahap penulisan berkas audio. Setelah proses *encoding* maupun *decoding*, *array* sampel yang dihasilkan akan ditulis kedalam berkas dengan ekstensi .wav, Tipe data dari *array* sampel yang akan ditulis dikonversi terlebih dahulu menjadi *signed integer* 16 bit. Implementasi untuk penulisan berkas audio dapat dilihat pada Kode sumber pada Gambar 4.28.

```
FUNCTION write_message(string path, Wave waveObject)
   OPEN path IN WRITE MODE
   WRITE(path, waveObject)
END FUNCTION
```

**Gambar 4.28 *Pseudocode* Implementasi Penulisan Berkas Audio**

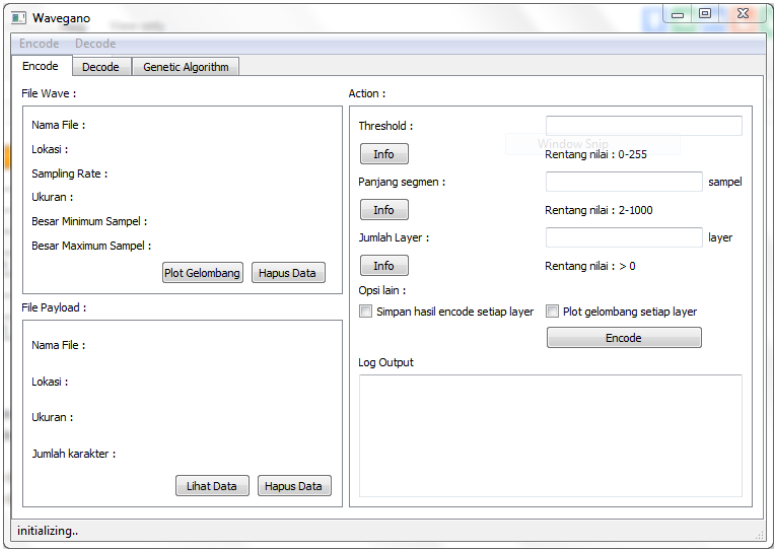
## 4.3 Implementasi Antarmuka Pengguna

Implementasi tampilan antarmuka dari sistem ini berbasis *desktop* dan berjalan pada sistem operasi Windows, Linux dan semua sistem operasi yang mendukung python dan pyqt. Antarmuka didesain menggunakan pustaka pyqt yaitu pustaka python untuk antarmuka pengguna berbasis grafis.

### 4.3.1 Menu Encoding

Menu *encoding* adalah menu yang digunakan untuk melakukan proses *encoding* pada berkas audio. Untuk melakukan *encoding*, langkah pertama yang dilakukan pengguna adalah membuka berkas audio yang akan dijadikan *cover* melalui menu Encode pada *menubar*, kemudia pengguna memilihi submenu "Open Wave File". Setelah pengguna memilih berkas yang akan digunakan pada dialog maka informasi dari berkas audio akan tampil pada aplikasi. Pengguna dapat menampilkan plot dari gelombang audio serta menghapus data berkas yang telah dibuka. Langkah selanjutnya adalah memilih berkas yang akan digunakan sebagai data *payload* dengan memilih pada submenu "Open Payload File", info dari *data* payload juga akan ditampilkan pada aplikasi.

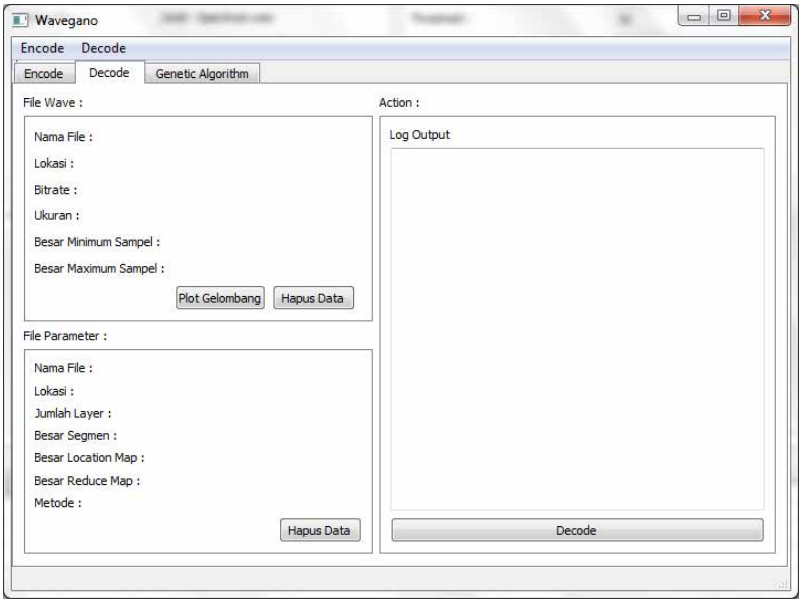
Pengguna dapat melihat data *payload* yang telah dibuka dan juga menghapus data terserbut. Pada menu ini terdapat beberapa kotak masukan untuk menentukan parameter dari proses *encoding* dan juga beberapa opsi untuk menyimpan hasil *encode* dari setiap *layer* dan menyimpan gambar plot gelombang dari setiap *layer*. Setelah *encoding* dilakukan jendela Log Output akan menampilkan pesan dan hasil dari *encoding* seperti kapasitas setiap *layer* dan PSNR dari hasil. Implementasi dari menu ini dapat dilihat pada Gambar 4.29.




**Gambar 4.29 Tampilan Menu *Encoding***

### 4.3.2 Menu Decoding

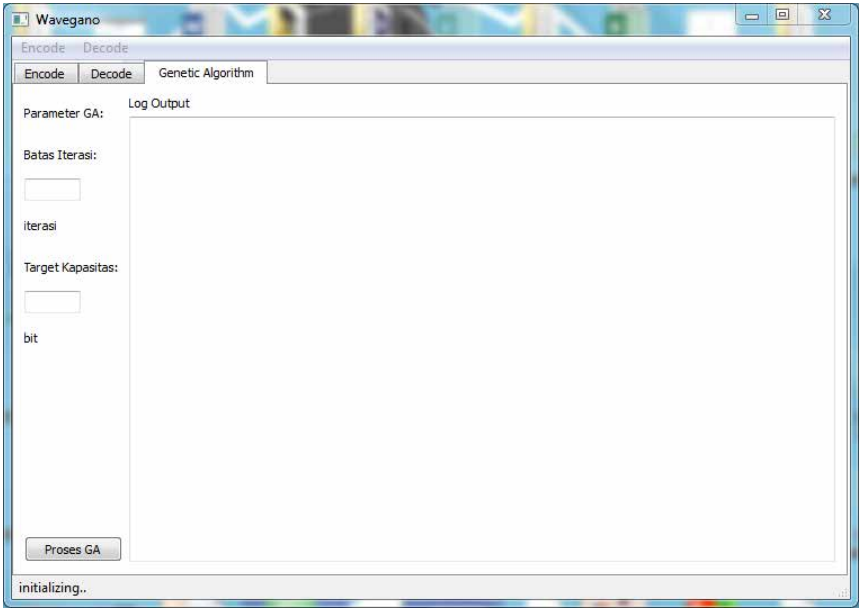
Menu ini digunakan untuk melakukan proses *decoding*. Untuk melakukan *decoding*, langkah pertama yang dilakukan pengguna adalah membuka berkas audio hasil dari proses *encoding* melalui *menu bar* Decode lalu memilih submenu "Open Encoded File". Setelah pengguna memilih berkas audio, pengguna dapat menampilkan plot dari gelombang audio atau dapat menghapus data berkas. Kemudian pengguna membuka berkas parameter yang sesuai dengan berkas audio yang akan di decode, info dari parameter akan ditampilkan pada aplikasi. Setelah itu proses *decode* akan berjalan setelah pengguna menekan tombol "Decode". Setelah proses selesai, info dari proses *decoding* akan ditampilkan pada Log Output. Implementasi dari menu ini dapat dilihat pada Gambar 4.30.




**Gambar 4.30 Tampilan Menu Decoding**

### 4.3.3 Implementasi menu GA

Menu ini digunakan untuk melakukan proses GA dalam mencari parameter yang optimal untuk proses *encoding*. Untuk melakukan proses GA pengguna perlu memasukkan jumlah iterasi dan target kapasitas yang diinginkan. Proses GA akan berjalan setelah tombol "Proses GA" ditekan. Proses akan berjalan hingga mencapai target yang ditentukan atau mencapai batas iterasi maksimal. Setelah proses GA selesai, info mengenai proses akan ditampilkan pada jendela Log Output. Implementasi dari menu GA ditunjukkan pada Gambar 4.31.




**Gambar 4.31 Tampilan Menu Proses GA**

| | |
|---|---|
| | Jumlah Layer = 7 |
| Kapasitas Hasil | 13392 |

**Tabel 5.25 Hasil Uji Coba 3 Proses GA**

Dari hasil uji coba diatas dapat dilihat bahwa proses GA tidak dapat mencapai nilai optimal yaitu target yang ditentukan karena telah mencapai batas iterasi, namun berhasil menemukan solusi sub optimal pada iterasi ke 20 yaitu dengan besar kapasitas 13392.

## 5.9 Perbandingan Hasil dengan Metode Lain

Subbab ini membahas mengenai perbandingan hasil *encoding* dari metode yang digunakan pada tugas akhir ini dengan metode yang telah ada. Hasil yang dibandingkan adalah hasil dari metode untuk melakukan steganografi pada audio yaitu antara lain *Spread Spectrum* , *Difference Expansion* , dan *Histogram Shifting.* Poin yang akan dibandingkan adalah kapasitas dalam satuan bit/detik, nilai PSNR dan reversibilitas dari metode. Parameter yang digunakan adalah panjang segmen sebesar 30 dan nilai threshold sebesar 65 Tabel 5.26 menunjukkan perbandingan dari hasil setiap metode.

| No | Nama Metode | Kapasitas (bit/detik) | PSNR | Reversible |
|---|---|---|---|---|
| 1 | Spread Spectrum | 32000 | 18,243 | TIDAK |
| 2 | Difference Expansion | 12578 | 26,45 | YA |
| 3 | Histogram Shifting | 15840 | 28,48 | TIDAK |
| 4 | Improved Reduced Difference Expansion | 56394 | 28.10508 | YA |

**Tabel 5.26 Tabel Perbandingan Hasil Metode**

Dari tabel diatas dapat dilihat bahwa metode RDE yang digunakan dapat memberikan kapasitas yang jauh lebih besar dari metode yang lain dengan nilai PSNR yang baik yaitu dengan perbedaan hanya sebesar 1.4% dengan metode *Histogram Shifting* yang memberikan nilai PSNR paling baik dari keempat metode diatas, terlebih lagi metode RDE memiliki sifat *reversible* sementara *Histogram Shifting* tidak.

## 5.10 Uji Coba dengan *Format* MP3

Pada subbab ini akan dibahas percobaan menggunakan metode pada berkas audio dengan *format* selain wav yaitu *format* MP3 dan hambatan yang ditemukan dalam melakukannya.

MP3 adalah sebuah *format* untuk audio digital yang diproses menggunakan kompresi yang bersifat *lossy*, sehingga ada informasi yang hilang saat berkas audio mengalami kompresi. MP3 telah menjadi standar dan menjadi *format* yang paling umum digunakan dalam distribusi berkas audio, namun dalam memproses berkas MP3 ada beberapa hambatan yang ditemui. Karena berkas MP3 merupakan berkas hasil kompresi yang berarti ada informasi yang hilang maka MP3 kurang merepresentasikan karakteristik asli dari suatu berkas audio, ketika metode RDE diterapkan pada berkas MP3 maka distorsi yang dihasilkan akan semakin besar sehingga kualitas dari berkas audio akan mengalami penurunan yang signifikan.

Hambatan yang lain adalah beberapa berkas MP3 merepresentasikan sampel dari berkas audio dalam bentuk data *float* sedangkan metode RDE hanya dapat diterapkan pada tipe data *integer* karena metode RDE menggunakan selisih dari sampel untuk melakukan penyisipan atau *embedding*. Ketika tipe data *float* dilakukan pembulatan menjadi data *integer*, maka pada saat proses *decoding* sampel tidak dapat dikembalikan ke nilai asalnya yang berarti akan ada informasi yang hilang sehingga sifat *reversible* dari metode RDE tidak dapat dipertahankan.

# BAB V
# PENGUJIAN DAN EVALUASI

Bab ini berisi bahasan mengenai uji coba dan evaluasi perangkat lunak yang telah dikembangkan berdasarkan implementasi dari sistem steganografi pada berkas audio menggunakan metode RDE.

## 5.1 Lingkungan Uji Coba

Proses uji coba dilakukan pada lingkungan yang telah ditentukan. Pada uji coba ini, lingkungan dibedakan menjadi lingkungan perangkat keras dan lingkungan perangkat lunak. Berikut ini dijelaskan mengenai tiap-tiap lingkungan uji coba aplikasi.

### 5.1.1 Lingkungan Perangkat Keras

Deskripsi perangkat keras untuk proses uji coba perangkat lunak ini dapat dilihat pada Tabel 5.1.

| No | Deskripsi |
|---|---|
| 1 | Asus N43SL<br><br>Spesifikasi :<br>Prosesor : Intel i5 2410M @ 2.3 Ghz<br>Memori : 6GB DDR3 RAM @1333Mhz<br>Kartu Grafis : Intel HD Graphic 4000 768 MB |

**Tabel 5.1 Lingkungan Perangkat Keras**

### 5.1.2 Lingkungan Perangkat Lunak

Deskripsi perangkat lunak untuk proses uji coba dapat dilihat pada Tabel 5.2.

| No | Deskripsi |
|---|---|
| 1 | Sistem Operasi Windows 7 Ultimate 64-bit |
| 2 | Visual Studio Professional 2013 Version 12.0.21005.1 |
| 3 | Python 2.7.8 |

**Tabel 5.2 Lingkungan Perangkat Lunak**

## 5.2 Data Uji Coba

Data yang digunakan untuk uji coba implementasi sistem steganografi ini adalah berkas audio dengan ekstensi .wav yang merupakan klip musik dengan durasi 11 detik. Berkas audio ini memiliki ukuran 345 *Kilobyte* dengan *bitrate* sebesar 256Kbps. Berkas teks yang akan digunakan sebagai *data* payload adalah berkas dengan ekstensi .txt dengan ukuran 2,19 *Kilobyte* dan jumlah karakter sebanyak 2248 karakter.

## 5.3 Skenario Pengujian

Pada subbab ini akan dijelaskan mengenai skenario uji coba yang telah dilakukan. Terdapat beberapa skenario uji coba yang telah dilakukan, diantaranya yaitu:

1. Perbandingan kapasitas *encoding* dan nilai PSNR berdasarkan variasi nilai parameter *threshold* dengan nilai parameter besar segmen yang tetap.
2. Perbandingan kapasitas *encoding* dan nilai PSNR berdasarkan variasi nilai parameter besar segmen dengan nilai parameter *threshold* yang tetap.
3. Perbandingan kapasitas *encoding* dan nilai PSNR pada skema *encoding multi-layer.*
4. Uji coba *reversibility* dari hasil proses *encoding*

5. Uji coba proses GA untuk menentukan parameter *encoding*.

## 5.4 Skenario Pengujian : Perbandingan Kapasitas *Encoding* dan Nilai PSNR Berdasarkan Variasi Nilai Parameter *Threshold*

Pada skenario pengujian ini dilakukan proses *encoding* sebanyak 6 kali dimana pada tiap proses nilai parameter besar segmen yang digunakan adalah tetap, sedangkan nilai *threshold* yang digunakan bervariasi pada rentang 100 sampai 50. Kapasitas *encoding* serta nilai PSNR dari setiap proses dengan nilai besar segmen = 10 ditunjukkan oleh Tabel 5.3.

| No. | Threshold | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 50 | 89136 | 32,2753 | 138,322 | 85,947 |
| 2 | 60 | 108684 | 30,49728 | 105,593 | 74,404 |
| 3 | 70 | 130437 | 28,5439 | 115,549 | 76,686 |
| 4 | 80 | 154179 | 27,09732 | 119,414 | 76,839 |
| 5 | 90 | 166653 | 26,46574 | 128,584 | 86,368 |
| 6 | 100 | 180612 | 25,83126 | 123,164 | 86,368 |

**Tabel 5.3 Kapasitas dan PSNR Hasil *Encoding* Dengan Besar Segmen = 10**

Dari hasil uji coba diatas, didapatkan kapasitas *encoding* terbesar pada proses dengan nilai *threshold* = 100 sebesar 180612 bit sedangkan nilai PSNR terbaik didapatkan pada hasil dengan nilai *threshold* = 50 yaitu 32,2753. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran A.1 .

Uji coba berikutnya dilakukan dengan nilai besar segmen = 20 dan nilai *threshold* yang bervariasi dalam rentang 50 sampai dengan 100 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.4.

| No. | Threshold | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 50 | 36746 | 35,83276 | 126,775 | 70,839 |
| 2 | 60 | 49115 | 33,87884 | 86,131 | 99,015 |
| 3 | 70 | 69046 | 31,33857 | 89,139 | 64,09 |
| 4 | 80 | 100415 | 28,99764 | 77,323 | 55,903 |
| 5 | 90 | 113012 | 28,18302 | 74,478 | 55,109 |
| 6 | 100 | 127129 | 27,28366 | 66,931 | 47,627 |

**Tabel 5.4 Kapasitas Dan PSNR Hasil *Encoding* Dengan Besar Segmen = 20**

Dari hasil uji coba diatas, pada nilai segmen = 20 kapasitas terbesar didapatkan pada nilai threshold = 100 yaitu sebesar 127129 bit sedangkan kapasitas terkecil didapatkan pada nilai *threshold* = 50 yaitu sebesar 36746 bit, sebaliknya nilai PSNR terbaik didapatkan pada nilai *threshold* = 50 yaitu dengan nilai 35,83276 dan nilai PSNR terkecil didapatkan pada threshold = 100 dengan nilai 27,28366. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran A.2.

Uji coba berikutnya dilakukan dengan nilai besar segmen = 30 dan nilai *threshold* yang bervariasi dalam rentang 50 sampai dengan 100 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.5.

| No. | Threshold | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 50 | 17342 | 38,99249 | 97,229 | 77,325 |
| 2 | 60 | 23751 | 36,84187 | 99,188 | 55,344 |
| 3 | 70 | 38019 | 33,98812 | 88,619 | 57,078 |
| 4 | 80 | 74617 | 30,23166 | 66,766 | 58,608 |
| 5 | 90 | 84912 | 29,38892 | 67,879 | 42,836 |
| 6 | 100 | 96048 | 28,44975 | 69,018 | 43,258 |

**Tabel 5.5 Kapasitas dan PSNR Hasil *Encoding* Dengan Besar Segmen = 30**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada nilai threshold = 100 yaitu sebesar 96048 bit sedangkan kapasitas terkecil didapatkan pada nilai *threshold* = 50 yaitu sebesar 17342 bit, sebaliknya nilai PSNR terbaik didapatkan pada nilai *threshold* = 50 yaitu dengan nilai 38,99249 dan nilai PSNR terkecil didapatkan pada threshold = 100 dengan nilai 28,44975. grafik tren untuk uji coba ini dapat dilihat pada Lampiran A.3 .

Uji coba berikutnya dilakukan dengan nilai besar segmen = 40 dan nilai *threshold* yang bervariasi dalam rentang 50 sampai dengan 100 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.6.

| No. | Threshold | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 50 | 10725 | 41,58948 | 95,117 | 60,318 |
| 2 | 60 | 14664 | 38,9043 | 93,815 | 67,891 |
| 3 | 70 | 24141 | 35,82201 | 96,226 | 59,451 |
| 4 | 80 | 60255 | 31,14846 | 83,870 | 54,115 |
| 5 | 90 | 70317 | 30,12569 | 81,748 | 60,579 |
| 6 | 100 | 79599 | 29,25059 | 86,844 | 45,382 |

**Tabel 5.6 Kapasitas dan PSNR Hasil *Encoding* Dengan Besar Segmen = 40**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada nilai threshold = 100 yaitu sebesar 79599 bit sedangkan kapasitas terkecil didapatkan pada nilai *threshold* = 50 yaitu sebesar 10725 bit, sebaliknya nilai PSNR terbaik didapatkan pada nilai *threshold* = 50 yaitu dengan nilai 41,58948 dan nilai PSNR terkecil didapatkan pada threshold = 100 dengan nilaiv 29,25059. grafik tren untuk uji coba ini dapat dilihat pada Lampiran A.4.

Uji coba berikutnya dilakukan dengan nilai besar segmen = 50 dan nilai *threshold* yang bervariasi dalam rentang 50 sampai

dengan 100 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.7.

| No. | Threshold | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 50 | 5586 | 44,64487 | 89,015 | 77,999 |
| 2 | 60 | 7644 | 41,54113 | 95,723 | 59,403 |
| 3 | 70 | 15484 | 37,4776 | 102,547 | 49,310 |
| 4 | 80 | 53557 | 31,6273 | 86,717 | 56,028 |
| 5 | 90 | 62132 | 30,69965 | 92,915 | 55,996 |
| 6 | 100 | 67081 | 30,1138 | 74,495 | 53,447 |

**Tabel 5.7 Kapasitas dan PSNR Hasil *Encoding* Dengan Besar Segmen = 50**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada nilai threshold = 100 yaitu sebesar 67081 bit sedangkan kapasitas terkecil didapatkan pada nilai *threshold* = 50 yaitu sebesar 5586 bit, sebaliknya nilai PSNR terbaik didapatkan pada nilai *threshold* = 50 yaitu dengan nilai 44,64487 dan nilai PSNR terkecil didapatkan pada threshold = 100 dengan nilaiv 30,1138. grafik tren untuk uji coba ini dapat dilihat pada Lampiran A.5.

## 5.5 Skenario Pengujian : Perbandingan Kapasitas *Encoding* Dan Nilai PSNR Berdasarkan Variasi Nilai Parameter Besar Segmen

Pada skenario pengujian ini dilakukan proses *encoding* sebanyak 5 kali dimana pada tiap proses nilai parameter *threshold* yang digunakan tetap, sedangkan nilai besar segmen yang digunakan bervariasi pada rentang 10 sampai 50. Kapasitas *encoding* serta nilai PSNR dari hasil *encoding* akan dibandingkan pada setiap variasi nilai besar segmen. Uji coba dengan nilai *threshold* = 50 ditunjukkan oleh Tabel 5.8.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 89136 | 32,2753 | 138,322 | 85,947 |
| 2 | 20 | 36746 | 35,83276 | 126,775 | 70,839 |
| 3 | 30 | 17342 | 38,99249 | 97,229 | 77,325 |
| 4 | 40 | 10725 | 41,58948 | 95,117 | 60,318 |
| 5 | 50 | 5586 | 44,64487 | 89,015 | 77,999 |

**Tabel 5.8 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 50**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 89136 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 5586 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 44,64487 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 32,2753. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.1.

Uji coba berikutnya dilakukan dengan nilai *threshold* 60 dan besar segmen yang bervariasi dalam rentang 10 sampai dengan 50 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.9.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 108684 | 30,49728 | 105,593 | 74,404 |
| 2 | 20 | 49115 | 33,87884 | 86,131 | 99,015 |
| 3 | 30 | 23751 | 36,84187 | 99,188 | 55,344 |
| 4 | 40 | 14664 | 38,9043 | 93,815 | 67,891 |
| 5 | 50 | 7644 | 41,54113 | 95,723 | 59,403 |

**Tabel 5.9 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 60**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 108684 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 7644 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 41,54113 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 30,49728. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.2.

Uji coba berikutnya dilakukan dengan nilai *threshold* 70 dan besar segmen yang bervariasi dalam rentang 10 sampai dengan 50 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.10.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 130437 | 28,5439 | 115,549 | 76,686 |
| 2 | 20 | 69046 | 31,33857 | 88,139 | 64,090 |
| 3 | 30 | 38019 | 33,98812 | 88,619 | 57,078 |
| 4 | 40 | 24141 | 35,82201 | 96,226 | 59,451 |
| 5 | 50 | 15484 | 37,4776 | 102,547 | 49,310 |

**Tabel 5.10 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 70**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 130437 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 15484 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 37,4776 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 28,5439. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.3.

Uji coba berikutnya dilakukan dengan nilai *threshold* 80 dan besar segmen yang bervariasi dalam rentang 10 sampai dengan 50

seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.11.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 154179 | 27,09732 | 119,414 | 76,839 |
| 2 | 20 | 100415 | 28,99764 | 77,323 | 55,903 |
| 3 | 30 | 74617 | 30,23166 | 66,766 | 58,608 |
| 4 | 40 | 60255 | 31,14846 | 83,870 | 54,115 |
| 5 | 50 | 53557 | 31,6273 | 86,717 | 56,028 |

**Tabel 5.11 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 80**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 154179 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 53557 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 31,6273 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 27,09732. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.4.

Uji coba berikutnya dilakukan dengan nilai *threshold* 90 dan besar segmen yang bervariasi dalam rentang 10 sampai dengan 50 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.12.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 166653 | 26,46574 | 128,584 | 86,368 |
| 2 | 20 | 113012 | 28,18302 | 74,478 | 55,109 |
| 3 | 30 | 84912 | 29,38892 | 67,879 | 42,836 |
| 4 | 40 | 70317 | 30,12569 | 81,748 | 60,579 |
| 5 | 50 | 62132 | 30,69965 | 92,915 | 55,996 |

**Tabel 5.12 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 90**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 166653 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 62132 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 30,69965 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 26,46574. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.5.

Uji coba berikutnya dilakukan dengan nilai *threshold* 100 dan besar segmen yang bervariasi dalam rentang 10 sampai dengan 50 seperti uji sebelumnya. Kapasitas dan nilai PSNR dari hasil uji coba ditunjukkan pada Tabel 5.13.

| No | Besar Segmen | Kapasitas (bit) | PSNR | Waktu *Encode* | Waktu *Decode* |
|---|---|---|---|---|---|
| 1 | 10 | 180612 | 25,83126 | 123,164 | 80,206 |
| 2 | 20 | 127129 | 27,28366 | 66,931 | 47,627 |
| 3 | 30 | 96048 | 28,44975 | 69,018 | 43,258 |
| 4 | 40 | 79599 | 29,25059 | 86,844 | 45,382 |
| 5 | 50 | 67081 | 30,1138 | 74,495 | 53,477 |

**Tabel 5.13 Kapasitas dan PSNR Hasil *Encoding* Dengan *Threshold* = 100**

Dari hasil uji coba diatas, kapasitas terbesar didapatkan pada proses dengan besar segmen 10 yaitu sebesar 180612 bit sedangkan kapasitas terkecil didapatkan pada nilai besar segmen 50 yaitu sebesar 67081 bit, sebaliknya nilai PSNR terbaik didapatkan pada proses dengan besar segmen 50 yaitu dengan nilai 30,1138 dan nilai PSNR terkecil didapatkan pada proses dengan besar segmen 10 dengan nilai 25,83126. grafik tren untuk uji coba ini dapat dilihat pada Lampiran B.6.

## 5.6 Skenario Pengujian : Perbandingan Kapasitas *Encoding* dan Nilai PSNR Pada Skema *Encoding Multi-Layer*

Pada skenario pengujian ini dilakukan proses *encoding* sebanyak 5 kali dimana pada setiap proses *encoding* dilakukan sebanyak 5 *layer* dengan nilai parameter besar segmen dan *threshold* yang bervariasi. Kapasitas dan nilai PSNR pada setiap *layer* akan dibandingkan. Tabel 5.14 menunjukkan hasil dari *encoding* dengan besar segmen 30 dan *threshold* 70.

| layer | kapasitas | PSNR |
|---|---|---|
| 1 | 38019 | 33,98812 |
| 2 | 26042 | 32,92548 |
| 3 | 22707 | 31,96761 |
| 4 | 20822 | 31,17902 |
| 5 | 20735 | 30,28532 |
| Kapasitas Total | | 128325 |
| Waktu *Encode* | | 237,387 |
| Waktu *Decode* | | 185,813 |

**Tabel 5.14 Kapasitas dan PSNR Hasil Encoding 5 *Layer* Dengan *Threshold* = 70 dan Besar Segmen =30**

Hasil uji coba diatas menunjukkan kapasitas total sebesar 128325 bit dan PSNR sebesar 30.28532 untuk *encoding* sebanyak 5 layer. Selisih dari kapasitas layer pertama dan layer terakhir sebesar 17284 atau terjadi penurunan sebesar 45.46%. grafik tren untuk uji coba ini dapat dilihat pada Lampiran C.1 dan plot dari gelomban pada tiap layer dapat dilihat pada Lampiran C.2.

Uji coba selanjutnya dilakukan dengan nilai besar segmen 10 dan *threshold* sebesar 50. Kapasitas dan nilai PSNR dari uji coba ditunjukkan pada Tabel 5.15.

| layer | kapasitas | PSNR |
|---|---|---|
| 1 | 89136 | 32,2753 |
| 2 | 83880 | 31,40305 |
| 3 | 81000 | 30,459 |
| 4 | 80658 | 29,47563 |
| 5 | 80649 | 28,47628 |
| Kapasitas Total | | 415755 |
| Waktu *Encode* | | 262.240 |
| Waktu *Decode* | | 218.095 |

**Tabel 5.15 Kapasitas dan PSNR Hasil Encoding 5 *Layer* Dengan *Threshold* = 50 dan Besar Segmen =10**

Hasil uji coba diatas menunjukkan kapasitas total sebesar 415323 bit dan PSNR sebesar 28.47628 untuk *encoding* sebanyak 5 layer. Selisih dari kapasitas layer pertama dan layer terakhir sebesar 8487 atau terjadi penurunan sebesar 9.52%. %. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran C.3 dan plot dari gelomban pada tiap layer dapat dilihat pada Lampiran C.4.

Uji coba selanjutnya dilakukan dengan nilai besar segmen 20 dan *threshold* sebesar 100. Kapasitas dan nilai PSNR dari uji coba ditunjukkan pada Tabel 5.16.

| layer | kapasitas | PSNR |
|---|---|---|
| 1 | 127129 | 27,28366 |
| 2 | 112100 | 24,62313 |
| 3 | 106951 | 22,74279 |
| 4 | 104975 | 21,10951 |
| 5 | 104785 | 19,62007 |
| Kapasitas Total | | 555940 |
| Waktu *Encode* | | 231,825 |
| Waktu *Decode* | | 174,957 |

**Tabel 5.16 Kapasitas dan PSNR Hasil Encoding 5 *Layer* Dengan *Threshold* = 100 dan Besar Segmen =20**

Hasil uji coba diatas menunjukkan kapasitas total sebesar 555940 bit dan PSNR sebesar 27.28366 untuk *encoding* sebanyak 5 layer. Selisih dari kapasitas layer pertama dan layer terakhir sebesar 22344 atau terjadi penurunan sebesar 17.57%. %. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran C.5 dan plot dari gelomban pada tiap layer dapat dilihat pada Lampiran C.6.

Uji coba selanjutnya dilakukan dengan nilai besar segmen 50 dan *threshold* sebesar 50. Kapasitas dan nilai PSNR dari uji coba ditunjukkan pada Tabel 5.17.

| layer | kapasitas | PSNR |
|---|---|---|
| 1 | 5586 | 44.64487 |
| 2 | 5047 | 43.65468 |
| 3 | 4753 | 42.45606 |
| 4 | 4655 | 41.34018 |
| 5 | 4655 | 40.20747 |
| Kapasitas Total | | 24696 |
| Waktu *Encode* | | 247,184 |
| Waktu *Decode* | | 192,335 |

**Tabel 5.17 Kapasitas dan PSNR Hasil Encoding 5 *layer* Dengan *threshold* = 50 dan besar segmen =50**

Hasil uji coba diatas menunjukkan kapasitas total sebesar 24696 bit dan PSNR sebesar 40.20747 untuk *encoding* sebanyak 5 layer. Selisih dari kapasitas layer pertama dan layer terakhir sebesar 931 atau terjadi penurunan sebesar 16.66%. %. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran C.7 dan plot dari gelomban pada tiap layer dapat dilihat pada Lampiran C.8.

Uji coba selanjutnya dilakukan dengan nilai besar segmen 40 dan *threshold* sebesar 80. Kapasitas dan nilai PSNR dari uji coba ditunjukkan pada Tabel 5.18.

| layer | kapasitas | PSNR |
|---|---|---|
| 1 | 60255 | 31.14846 |
| 2 | 56394 | 28.10508 |
| 3 | 55770 | 25.81224 |
| 4 | 55614 | 23.95956 |
| 5 | 55614 | 22.2533 |
| Kapasitas Total | | 283647 |
| Waktu *Encoding* | | 226.246 |
| Waktu *Decoding* | | 179.386 |

**Tabel 5.18 Kapasitas dan PSNR Hasil Encoding 5 *Layer* Dengan *Threshold* = 80 dan Besar Segmen =40**

Hasil uji coba diatas menunjukkan kapasitas total sebesar 283467 bit dan PSNR sebesar 22.2533 untuk *encoding* sebanyak 5 layer. Selisih dari kapasitas layer pertama dan layer terakhir sebesar 4641 atau terjadi penurunan sebesar 7.7%. %. Grafik tren untuk uji coba ini dapat dilihat pada Lampiran C.9 dan plot dari gelomban pada tiap layer dapat dilihat pada Lampiran C.10.

## 5.7 Skenario Pengujian : Uji Coba *Reversibility* Dari Hasil Proses Encoding

Pada pengujian ini akan dilakukan uji coba terhadap karakteristik *reversible* dari metode yang digunakan di dalam sistem steganografi. Pada uji coba ini akan dilakukan proses *encoding* sebanyak 4 kali dengan variasi parameter dan jumlah *layer*. kemudian hasil dari proses tersebut akan di *decode* kembali kemudian hasil *decode* akan dibandingkan kemiripannya dengan berkas asli. Perbadingan dilakukan dengan alat bantu *diff checking tool* untuk mengetahui apakah ada perubahan pada berkas.

Uji coba pertama dilakukan pada berkas audio dengan ukuran 346 Kilobyte berdurasi 11 detik. Parameter yang digunakan adalah besar segmen 40, nilai *threshold* 70, dan jumlah layer sebanyak 4.

Berkas audio dan berkas *payload* hasil *encoding* masing-masing dibandingkan dengan *diff tool* untuk mengetahui perbedaan pada berkas. Tabel 5.19 menunjukkan kemiripan antara berkas hasil *decode* dan berkas asli. Data lengkap ditunjukkan oleh Lampiran D.1.

| Berkas | Kemiripan |
|---|---|
| Audio | 100% |
| Payload | 100% |

**Tabel 5.19 Kemiripan Berkas Hasil *Decode* Uji Coba 1**

Uji selanjutnya dilakukan pada berkas audio dengan ukuran 344 Kilobyte berdurasi 11 detik. Parameter yang digunakan adalah besar segmen 20, nilai *threshold* 100, dan jumlah layer sebanyak 6. Berkas audio dan berkas *payload* hasil *encoding* masing-masing dibandingkan dengan *diff tool* untuk mengetahui perbedaan pada berkas. Tabel 5.20 menunjukkan kemiripan antara berkas hasil *decode* dan berkas asli. Data lengkap ditunjukkan oleh Lampiran D.2.

| Berkas | Kemiripan |
|---|---|
| Audio | 100% |
| Payload | 100% |

**Tabel 5.20 Kemiripan Berkas Hasil *Decode* Uji Coba 2**

Uji coba selanjutnya dilakukan pada berkas audio dengan ukuran 348 Kilobyte berdurasi 11 detik. Parameter yang digunakan adalah besar segmen 30, nilai *threshold* 50 dan jumlah layer sebanyak 3. Berkas audio dan berkas *payload* hasil *encoding* masing-masing dibandingkan dengan *diff tool* untuk mengetahui perbedaan pada berkas. Tabel 5.21 menunjukkan kemiripan antara berkas hasil *decode* dan berkas asli. Data lengkap ditunjukkan oleh Lampiran D.3.

| Berkas | Kemiripan |
|---|---|
| Audio | 100% |
| Payload | 100% |

**Tabel 5.21 Kemiripan Berkas Hasil *Decode* Uji Coba 3**

Uji coba selanjutnya dilakukan pada berkas audio dengan ukuran 344 Kilobyte berdurasi 11 detik. Parameter yang digunakan adalah besar segmen 20, nilai *threshold* 80 dan jumlah layer sebanyak 5. Berkas audio dan berkas *payload* hasil *encoding* masing-masing dibandingkan dengan *diff tool* untuk mengetahui perbedaan pada berkas. Tabel 5.22 menunjukkan kemiripan antara berkas hasil *decode* dan berkas asli. Data lengkap ditunjukkan oleh Lampiran D.4.

| Berkas | Kemiripan |
|---|---|
| Audio | 100% |
| Payload | 100% |

**Tabel 5.22 Kemiripan Berkas Hasil *Decode* Uji Coba 4**

## 5.8 Skenario Pengujian : Uji Coba Proses GA Untuk Menentukan Parameter Encoding

Pada pengujian ini akan dilakukan proses GA untuk menentukan parameter optimal yang akan digunakan dalam proses *encoding*. Berkas audio yang digunakan adalah berkas pendek dengan panjang 1 detik. Proses akan dilakukan sebanyak 3 kali, masing-masing dengan batas iterasi yang berbeda dan nilai target yang berbeda. Sedangkan jumlah populasi ditetapkan sebanyak 5 kromosom. Kemudian, kapasitas yang dihasilkan pada tiap proses akan dibandingkan. Hasil dari pengujian ditunjukkan oleh Tabel 5.23, Tabel 5.24, dan Tabel 5.25.

| | |
|---|---|
| Batas Iterasi | 5 |
| Kapasitas Target | 20000 bit |
| Parameter Hasil | Besar Segmen = 61<br>Threshold = 203<br>Jumlah Layer = 6 |
| Kapasitas Hasil | 22300 |

**Tabel 5.23 Hasil Uji Coba 1 Proses GA**

Dari hasil uji coba diatas dapat dilihat bahwa proses GA berhasil melampaui nilai optimal yaitu target yang telah ditentukan dengan kapasitas sebesar 22300 dengan melakukan proses sebanyak 5 iterasi.

| | |
|---|---|
| Batas Iterasi | 10 |
| Kapasitas Target | 40000 bit |
| Parameter Hasil | Besar Segmen = 29<br>Threshold = 111<br>Jumlah Layer = 5 |
| Kapasitas Hasil | 29502 |

**Tabel 5.24 Hasil Uji Coba 2 Proses GA**

Dari hasil uji coba diatas dapat dilihat bahwa proses GA tidak dapat mencapai nilai optimal yaitu target yang ditentukan karena telah mencapai batas iterasi, tetapi berhasil menemukan solusi sub-optimal pada iterasi ke 10 yaitu dengan besar kapasitas 29502.

| | |
|---|---|
| Batas Iterasi | 20 |
| Kapasitas Target | 80000 bit |
| Parameter Hasil | Besar Segmen = 59<br>Threshold = 251 |

# BAB VI
# KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, terdapat pula saran yang ditujukan untuk pengembangan perangkat lunak lebih lanjut.

## 6.1. Kesimpulan

Dalam proses pengerjaan tugas akhir mulai dari tahap analisis, desain, implementasi, hingga pengujian didapatkan kesimpulan sebagai berikut:

1. Metode *Reduced Difference Expansion* dapat diaplikasikan pada berkas audio dengan cara melakukan partisi pada sampel terlebih dahulu.
2. Metode *Reduced Difference Expansion* dapat dikembangkan lebih jauh untuk meningkatkan kapasitas dan kualitas dari hasil *encoding*
3. Data uji coba menunjukkan bahwa setiap kenaikan nilai parameter besar segmen sebesar 10 sampel terjadi rata-rata penurunan kapasitas *encoding* sebesar 49,375% dan terjadi rata-rata kenaikan PSNR sebesar 8,24%
4. Data uji coba menunjukkan bahwa setiap kenaikan nilai parameter *Threshold* sebesar 10 sampel terjadi rata-rata kenaikan kapasitas *encoding* sebesar 20,177% dan terjadi rata-rata penurunan PSNR sebesar 5,74%
5. Uji coba proses *encode* menggunakan skema *multi-layer* memperlihatkan peningkatan rata-rata kapasitas sebesar 17.53% dan memperlihatkan penurunan rata-rata dari nilai PSNR sebesar 2,71%

6. Kakteristik *reversible* dari metode *Reduced Difference Expansion* dapat dibuktikan melalui uji coba dimana berkas audio dan berkas *payload* berhasil dikembalikan tanpa adanya informasi yang hilang yang mana ditujunkkan oleh tingkat kemiripan 100% dengan berkas asli
7. *Genetic Algorithm* dapat digunakan sebagai metode untuk menemukan parameter yang optimal atau mendekati optimal untuk melakukan *encoding*.

## 6.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang, berdasarkan pada hasil perancangan, implementasi dan uji coba yang telah dilakukan.

1. Skema penyisipan parameter yang dibutuhkan untuk *decoding* kedalam berkas *cover* sehingga proses *decoding* dapat dilakukan tanpa perlu nya berkas tambahan.
2. Optimasi perhitungan dan paralelisasi komputasi pada proses RDE untuk mempercepat proses *encoding*.
3. Pengembangan teknik partisi sampel untuk mendapatkan blok yang memiliki selisih kecil sehingga kapasitas dan kualitas hasil dapat ditingkatkan lebih jauh lagi.
4. Pengembangan sistem agar dapat mengakomodasi berkas *payload* dengan tipe lain selain data teks.

# DAFTAR PUSTAKA


[1] Ingemar , J. Cox . 2008 .**Digital watermarking and steganography Morgan Kaufmann**, Burlington:Morgan Kaufmann

[2] Bilal, I., Roj, M.S., Kumar, R., Mishra, P.K., 11-13 Dec. 2014 ,"Recent advancement in audio steganography", **Parallel, Distributed and Grid Computing (PDGC), 2014 International Conference on** , vol., no., pp.402,405

[3] Santosa, R.A., Bao, P., 8-10 Juni 2005,"Audio-to-image wavelet transform based audio steganography" , **ELMAR, 2005. 47th International Symposium** , vol., no., pp.209,212

[4] Jun Tian, Aug. 2003 , "Reversible data embedding using a difference expansion", **Circuits and Systems for Video Technology, IEEE Transactions on** , vol.13, no.8, pp.890,896

[5] Tohari , Ahmad. , Holil, M. , 2014 , "Increasing the Performance of Difference Expansion-based Steganography for Securing Medical Data" **[unpublished].**

[6] Grundlehner, Bernard,et al. , 2005 "Performance assessment method for speech enhancement systems." **Proc. 1st annu. IEEE BENELUX/DSP valley signal process.** symp**.**

[7] Zolzer , Udo. , 2008 , **Digital Audio Signal Processing** ,Hamburg:Wiley

[8] Choi, Ka-Cheng. , Pun , Chi-Man., Chen, C.L Philip , Januari 2013 , "Application of a generalized difference expansion

based reversible audio data hiding algorithm", **Multimedia Tools and Applications** , Impact Factor: 1.06

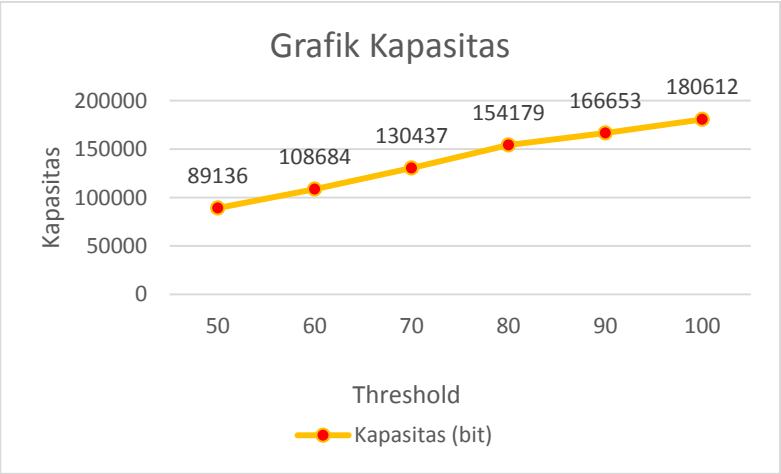
[9] SciPy Team. 2012. **Scientific Computing Tools for Python,** <URL: http://www.scipy.org/about.html>

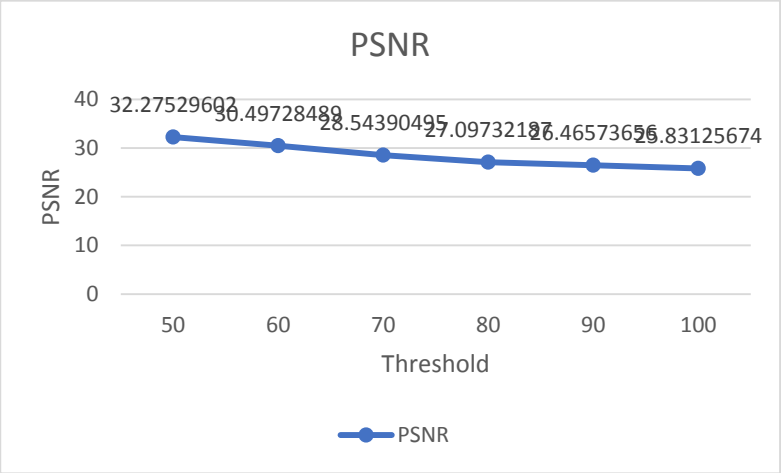
[10] Salomon, David.2007. **Data Compression: The Complete Reference** (4 ed.), Springer.

[11] Melanie, Mitchell. 1996. **An Introduction to Genetics Algorithms**, Bradford:MIT Press

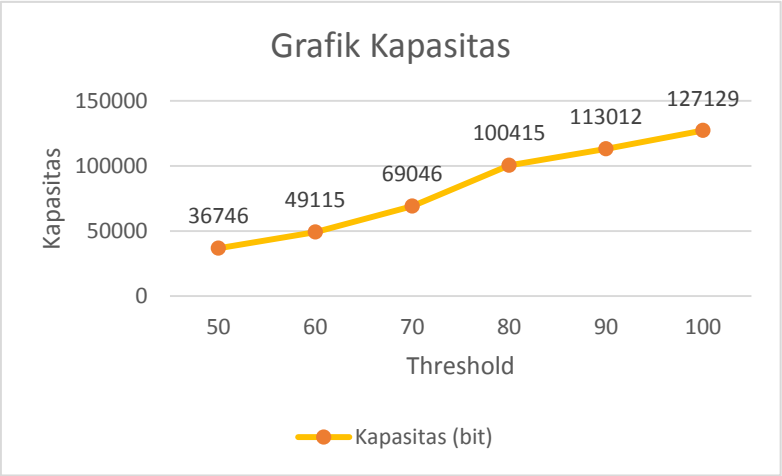
# LAMPIRAN A

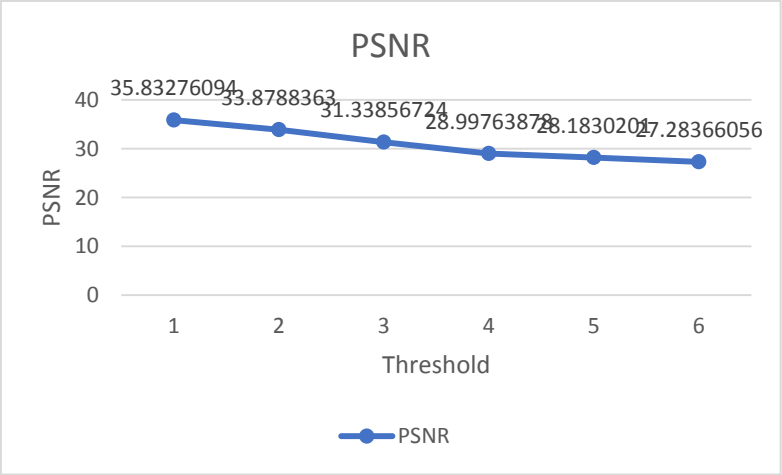
## LAMPIRAN GRAFIK TREN UJI COBA 1



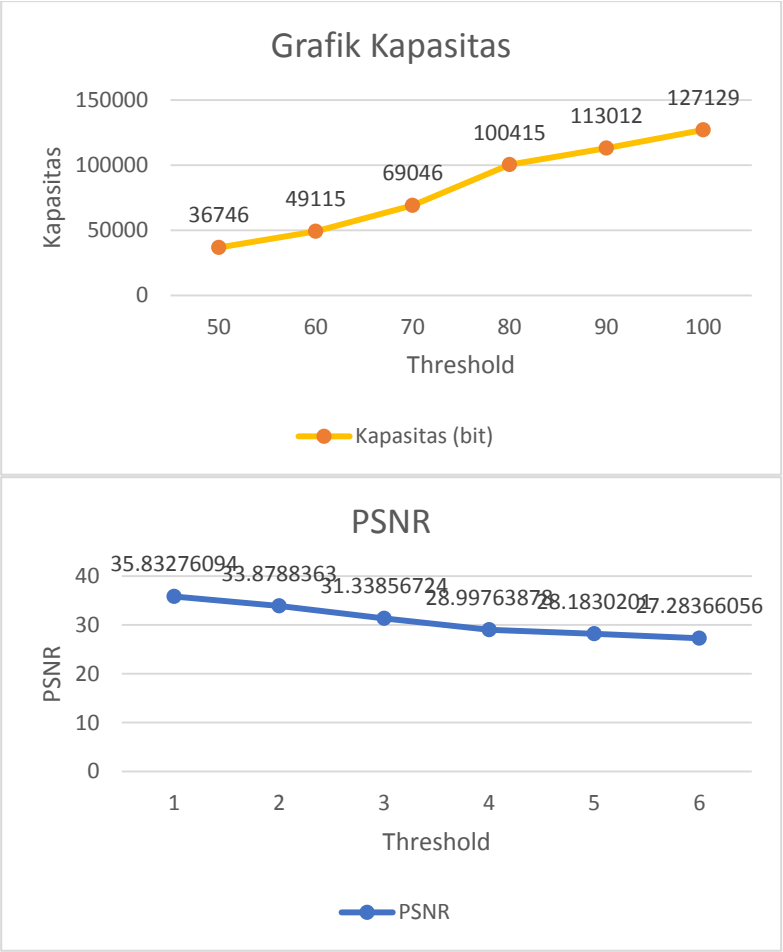




**Lampiran A.1 Grafik Tren Hasil *Encoding* dengan Besar Segmen = 10**

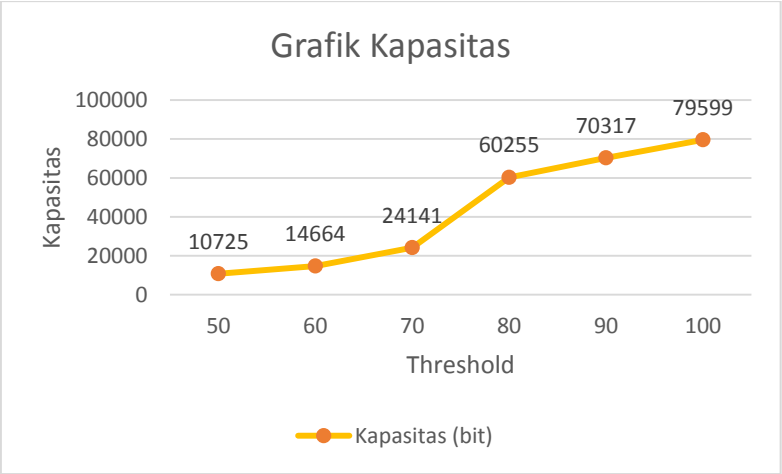


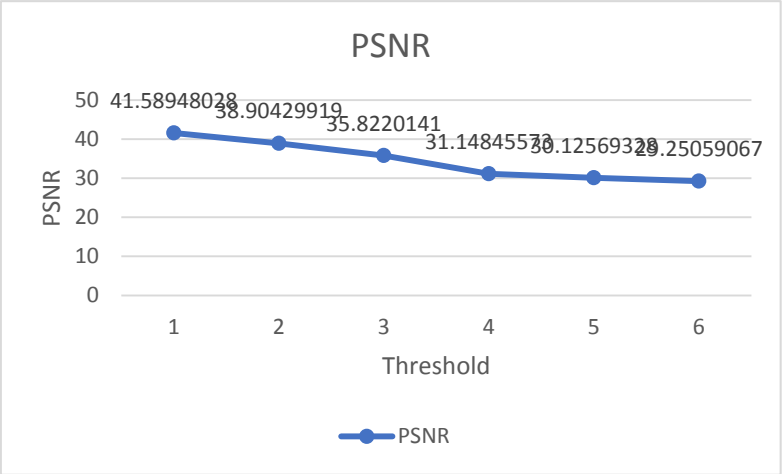




**Lampiran A.2 Grafik Tren Hasil *Encoding* dengan Besar Segmen = 20**

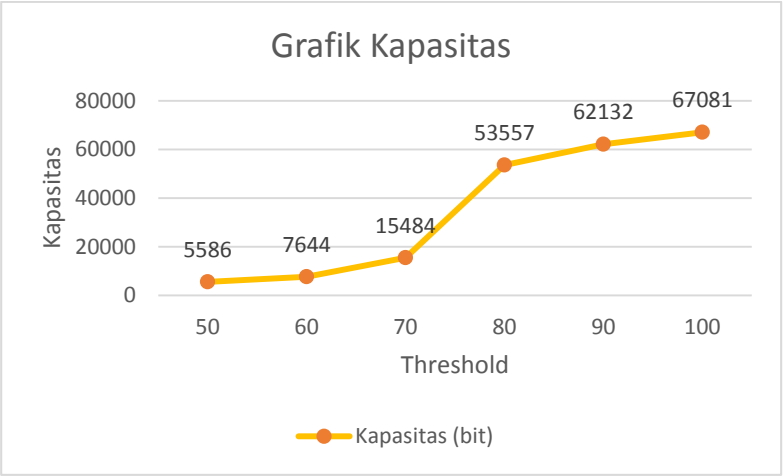


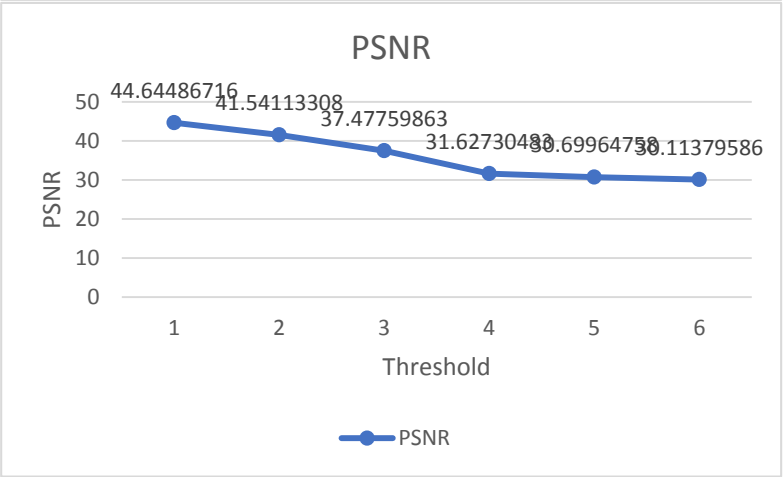

**Lampiran A.3 Grafik Tren Hasil *Encoding* dengan Besar Segmen = 30**




**Lampiran A.4 Grafik Tren Hasil *Encoding* dengan Besar Segmen = 40**



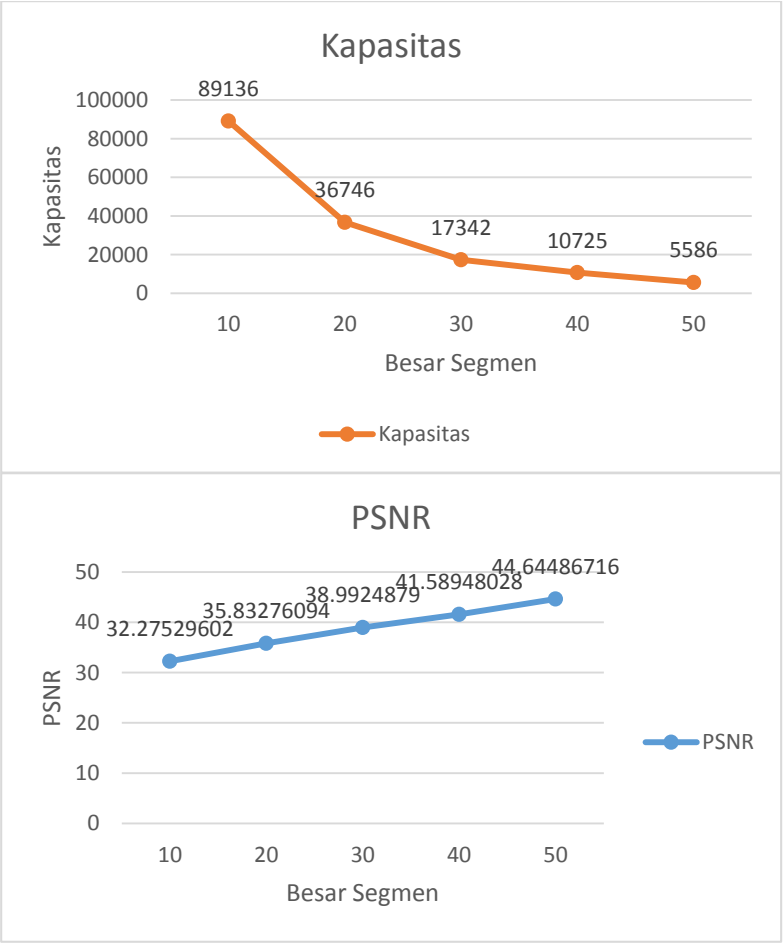




**Lampiran A.5 Grafik Tren Hasil *Encoding* dengan Besar Segmen = 50**

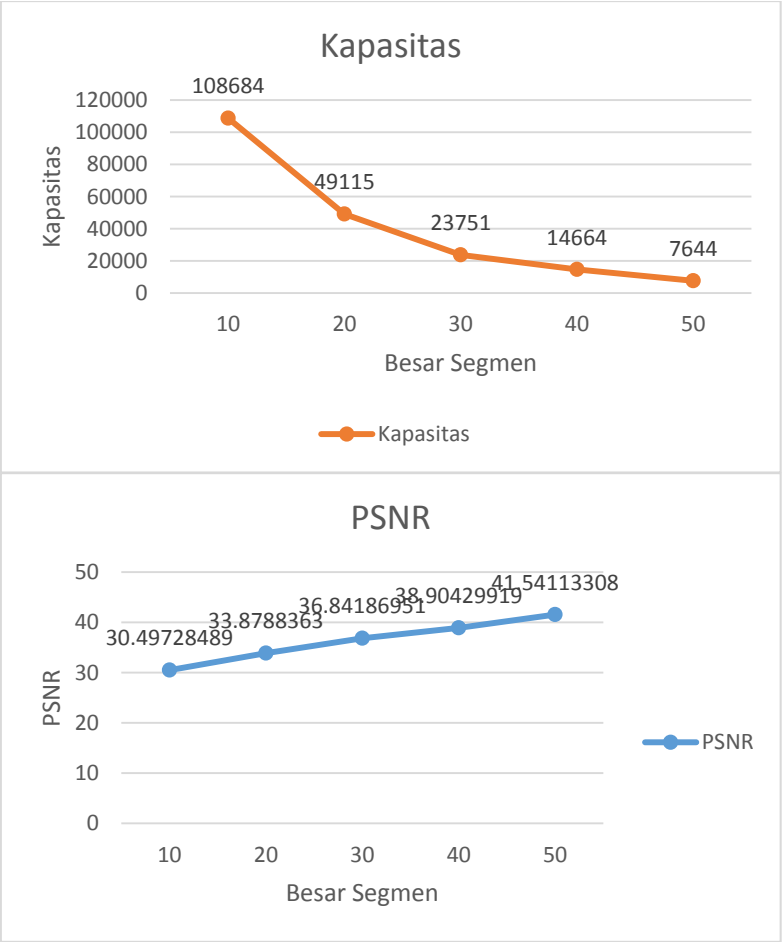
*(Halaman ini sengaja dikosongkan)*

# LAMPIRAN B

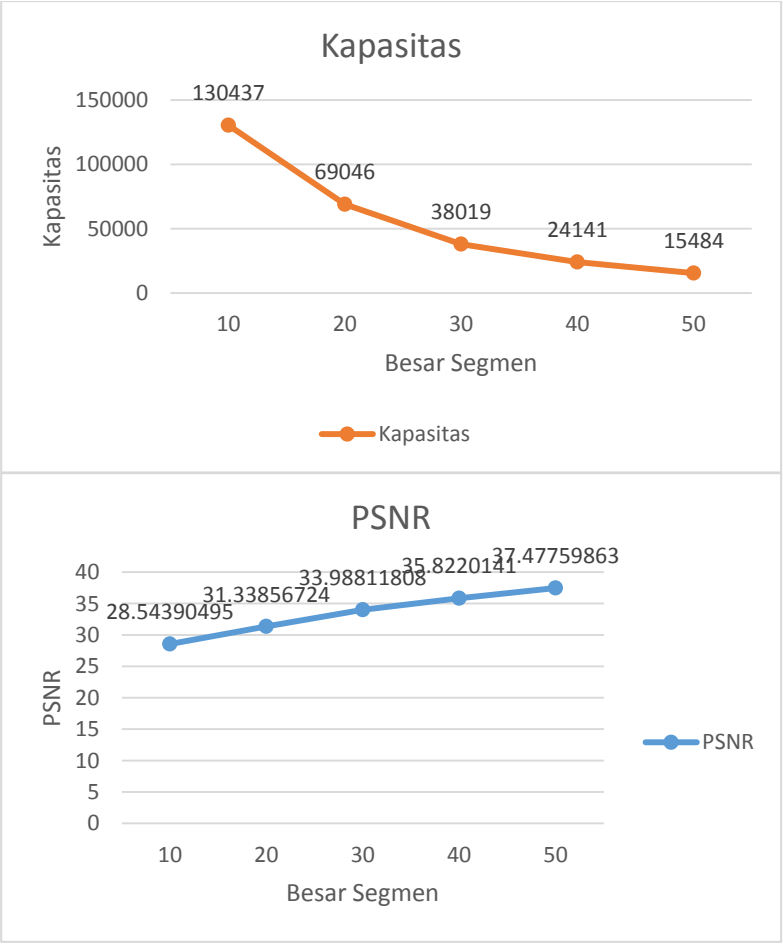
## LAMPIRAN GRAFIK TREN UJI COBA 2



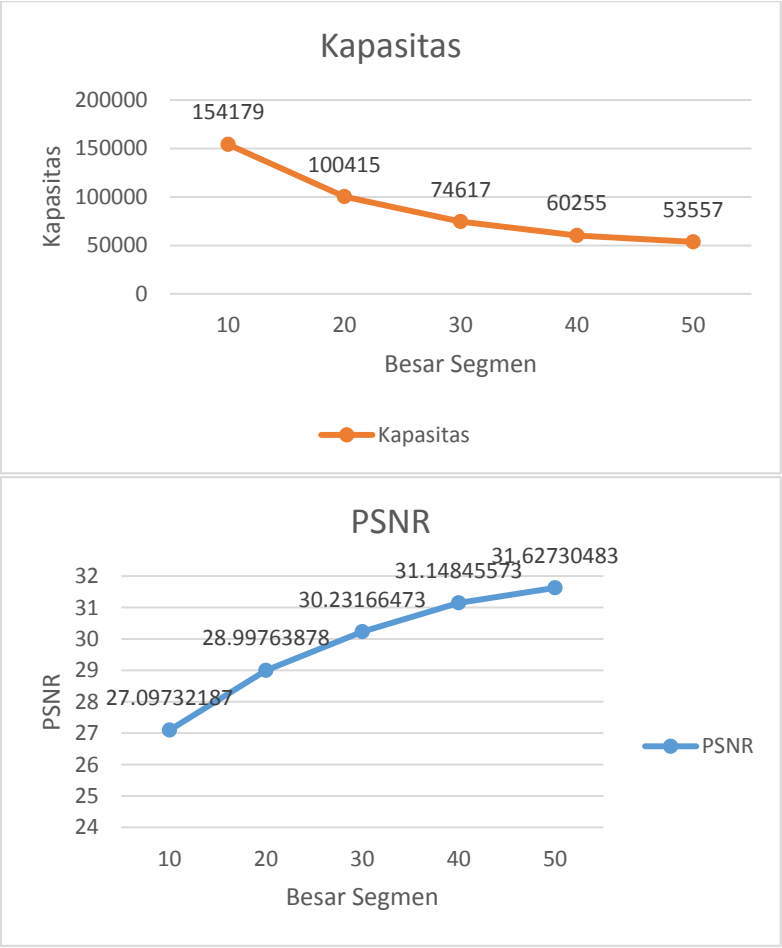

**Lampiran B.1 Grafik Tren Hasil *Encoding* dengan *Threshold* = 50**



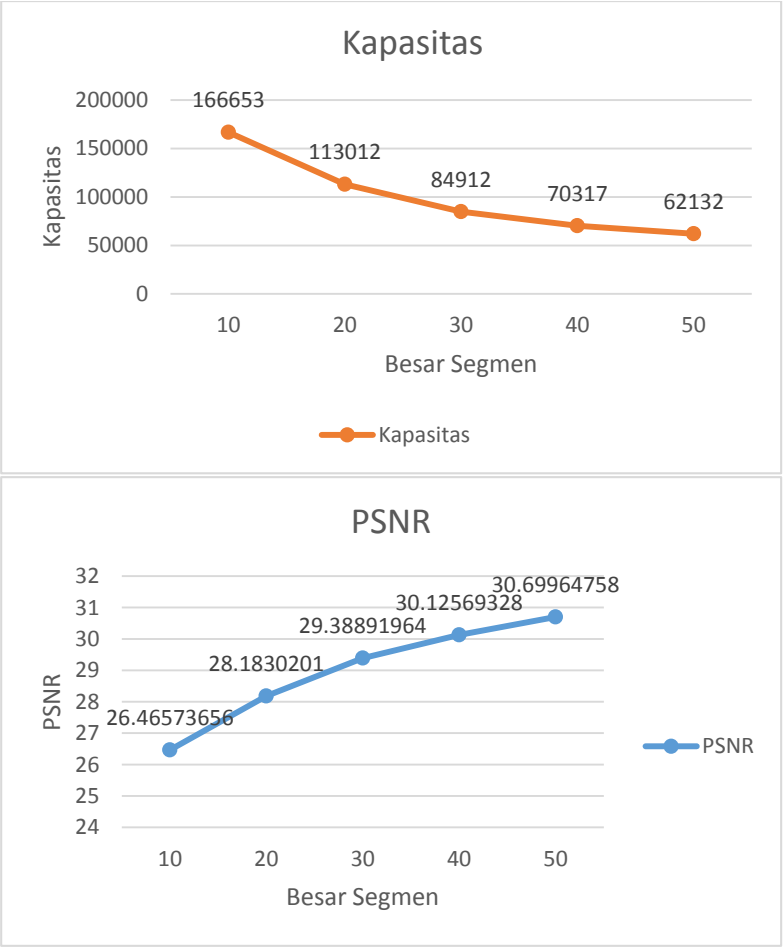

**Lampiran B.2 Grafik Tren Hasil *Encoding* dengan *Threshold* = 60**



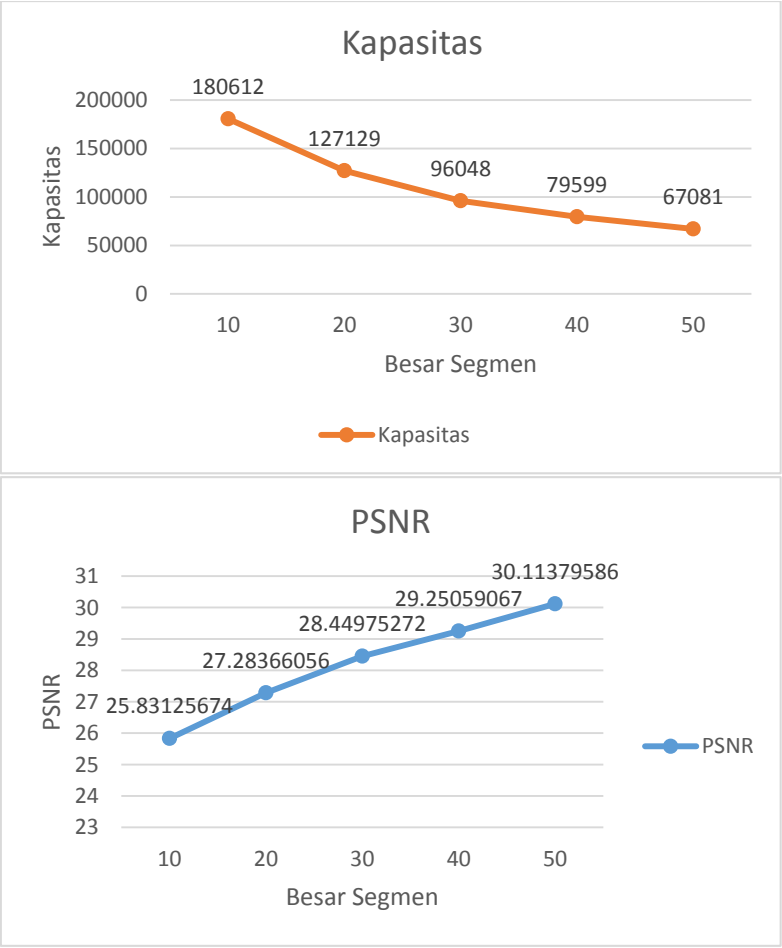

**Lampiran B.3 Grafik Tren Hasil *Encoding* dengan *Threshold* = 70**




**Lampiran B.4 Grafik Tren Hasil *Encoding* dengan *Threshold* = 80**



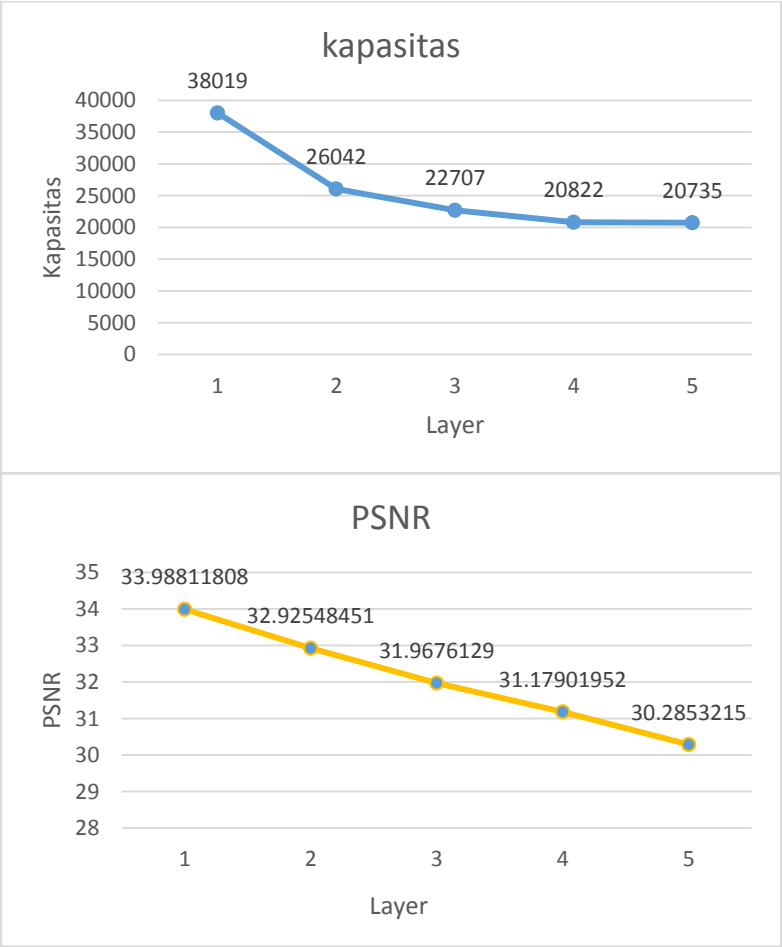

**Lampiran B.5 Grafik Tren Hasil *Encoding* dengan *Threshold* = 90**



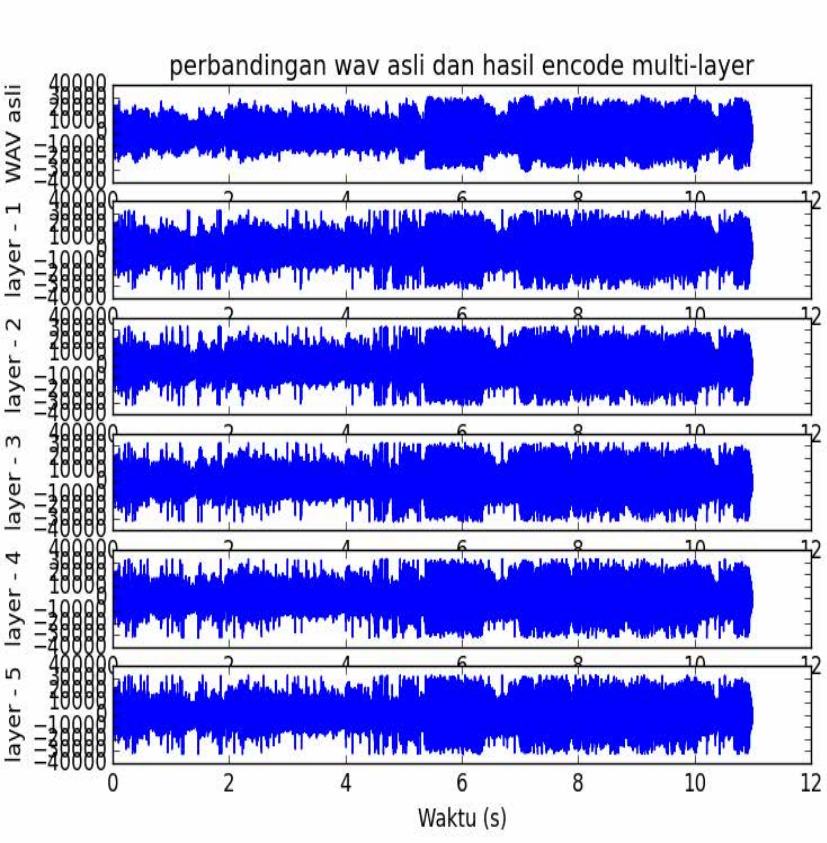

**Lampiran B.6 Grafik Tren Hasil *Encoding* dengan *Threshold* = 100**

# LAMPIRAN C

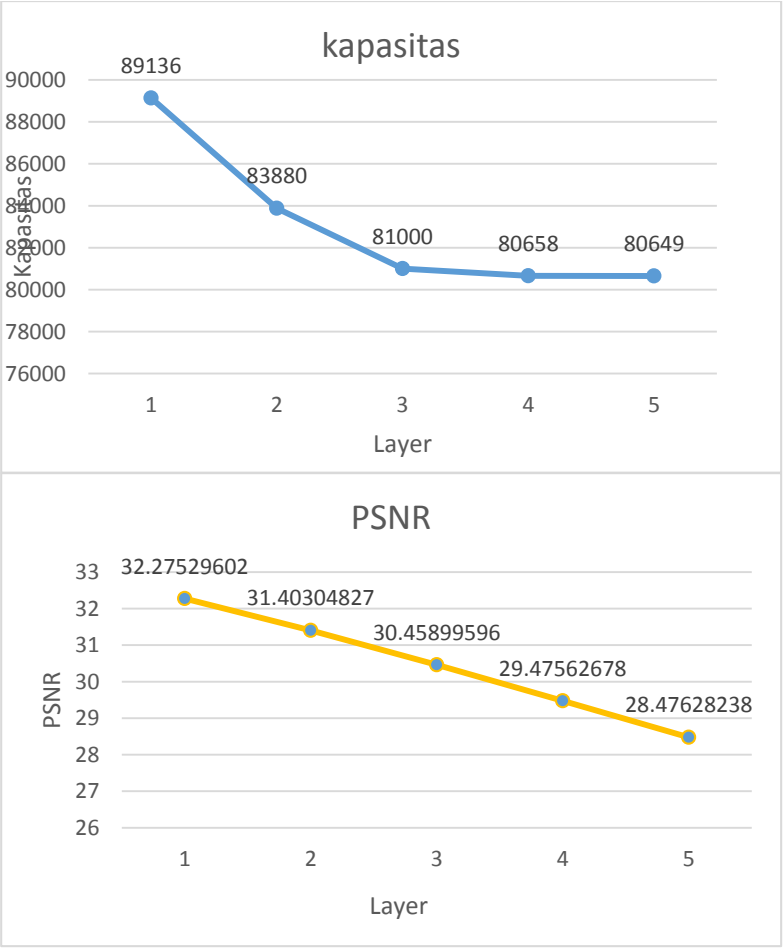
## LAMPIRAN GRAFIK DAN PLOT UJI COBA 3



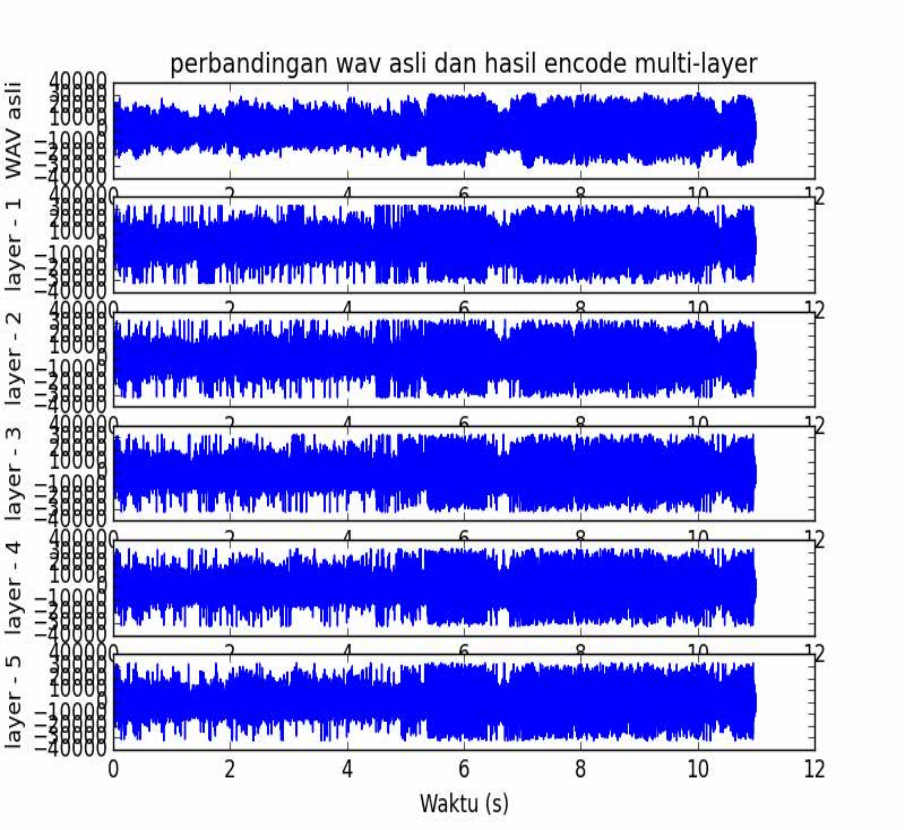

**Lampiran C.1 Grafik Tren Hasil *Encoding* 5 *Layer* dengan *Threshold* = 70 dan Besar Segmen =30**



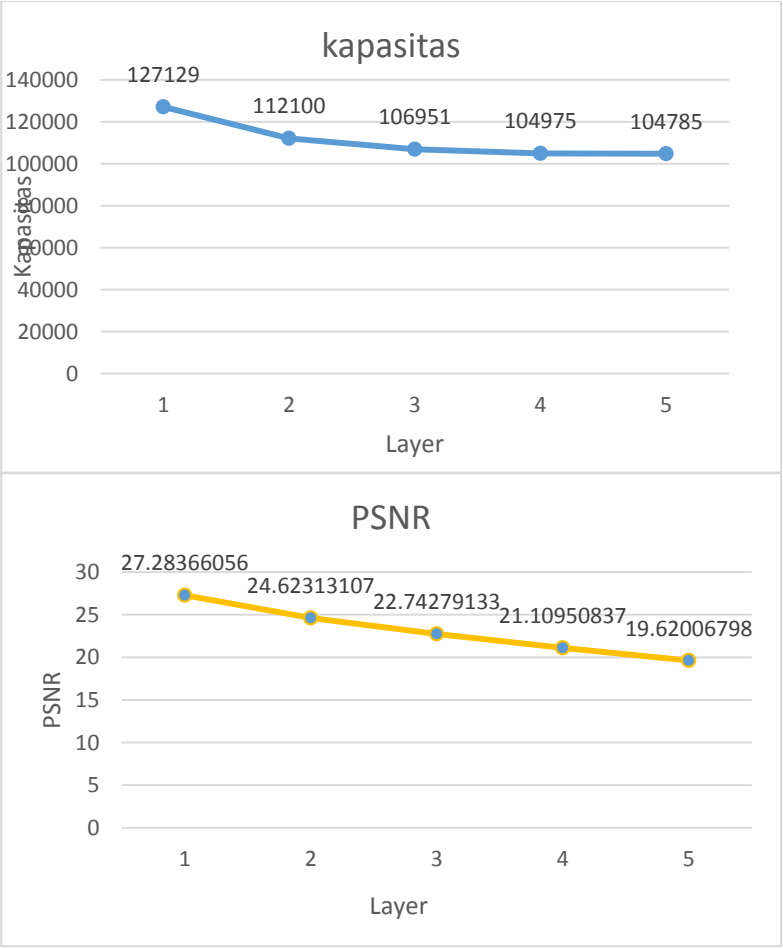

**Lampiran C.2 Plot Gelombang Audio Hasil *Encoding* 5 *Layer* dengan *Threshold* = 70 dan Besar Segmen =30**



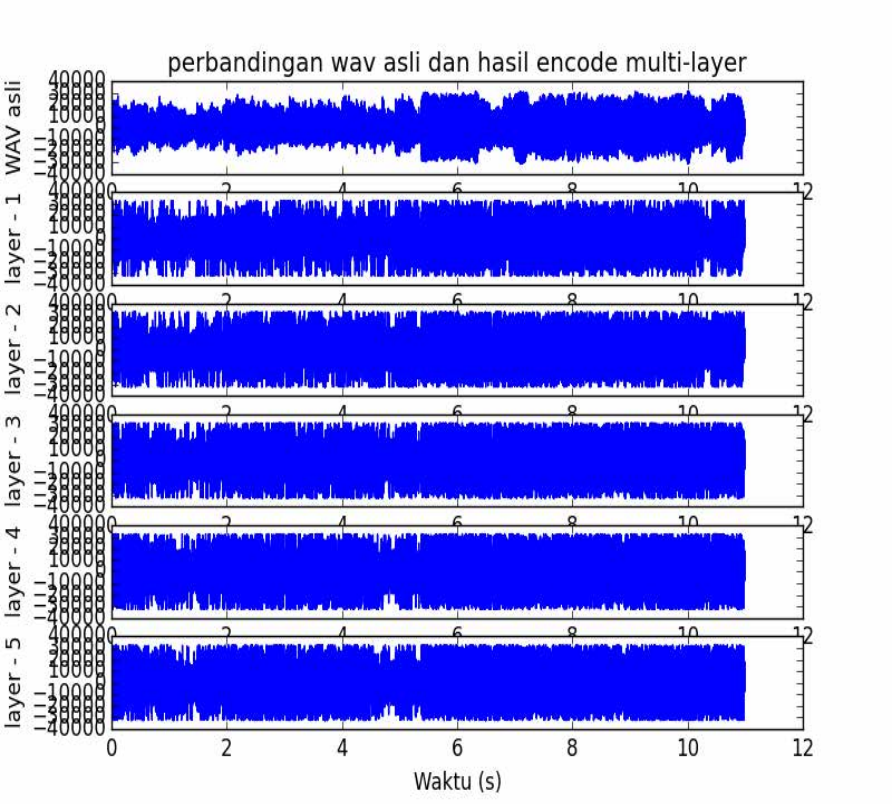

**Lampiran C.3 Grafik Tren Hasil *Encoding* 5 *Layer* dengan *Threshold* = 50 dan Besar Segmen =10**



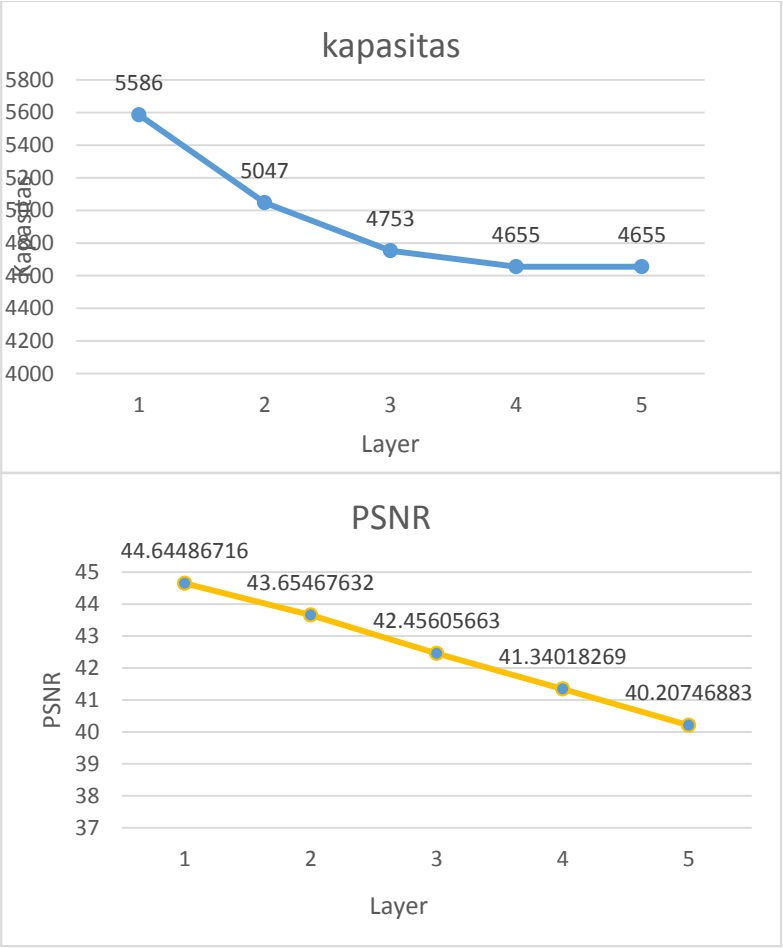

**Lampiran C.4 Plot Gelombang Audio Hasil *Encoding* 5 *Layer* dengan *Threshold* = 50 dan Besar Segmen =10**



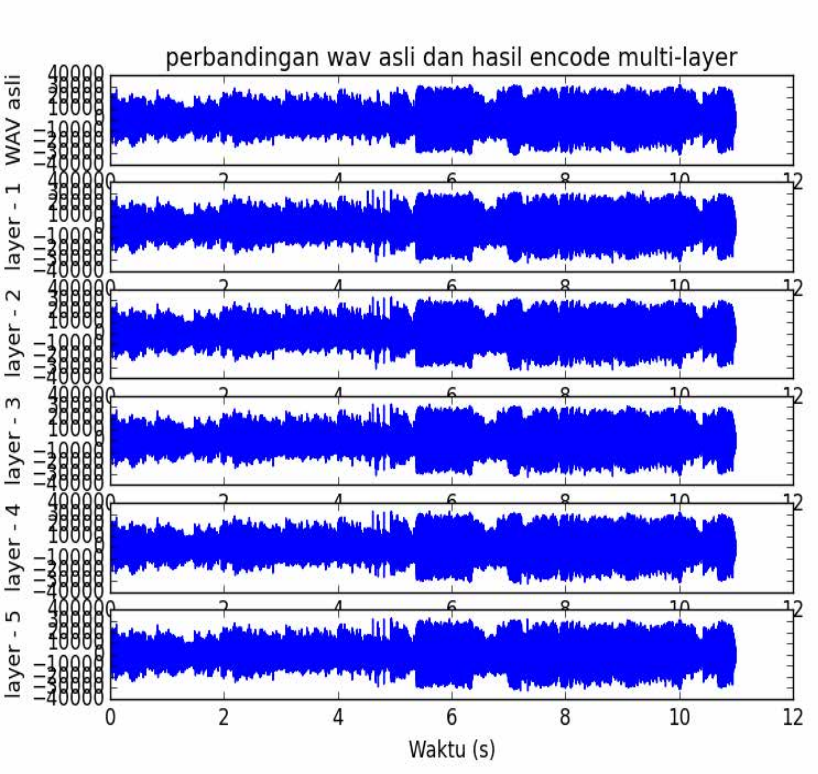

**Lampiran C.5 Grafik Tren Hasil *Encoding* 5 *Layer* dengan *Threshold* = 100 dan Besar Segmen =20**



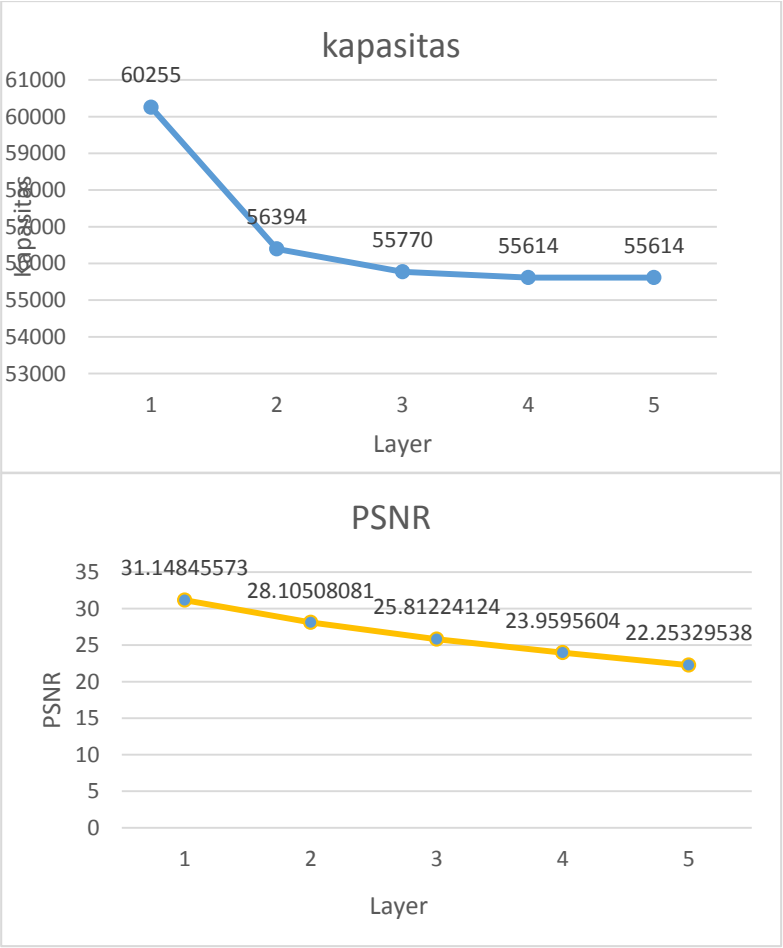

**Lampiran C.6 Plot Gelombang Audio Hasil *Encoding* 5 *Layer* dengan *Threshold* = 100 dan Besar Segmen =20**



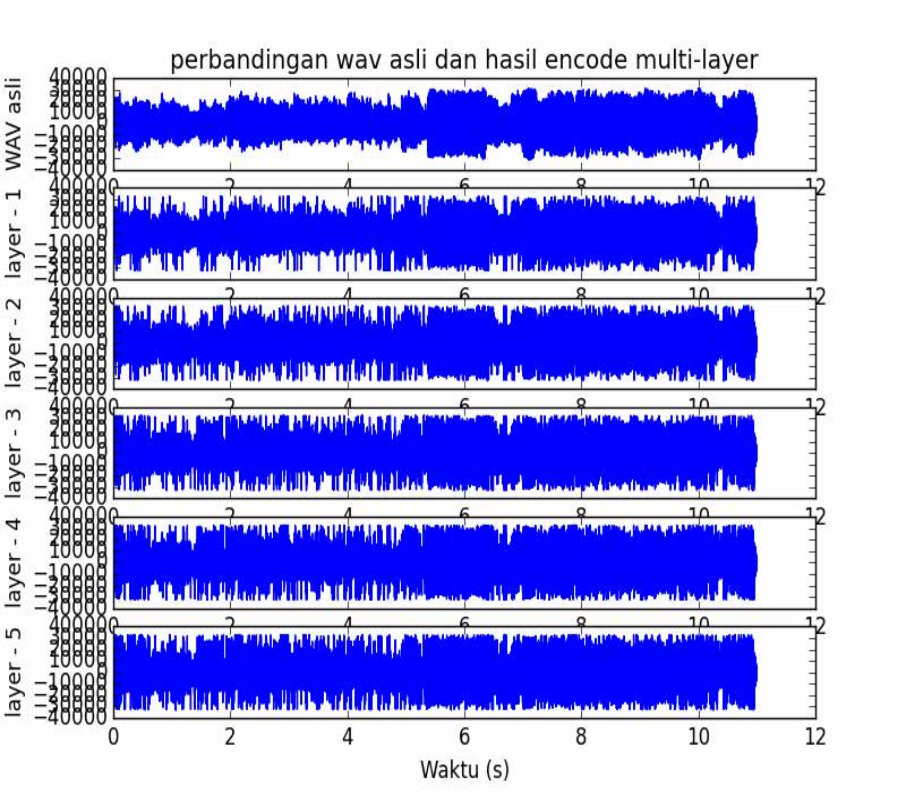

**Lampiran C.7 Grafik Tren Hasil *Encoding* 5 *Layer* dengan *Threshold* = 50 dan Besar Segmen =50**




**Lampiran C.8 Plot Gelombang Audio Hasil *Encoding* 5 *Layer* dengan *Threshold* = 50 dan Besar Segmen =50**



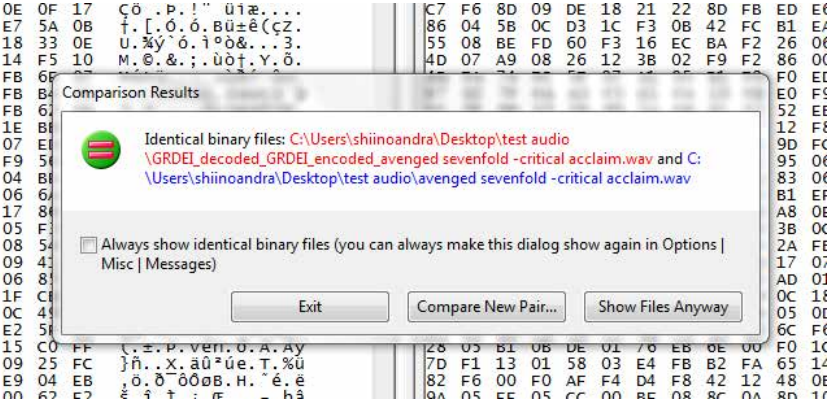

**Lampiran C.9 Grafik Tren Hasil *Encoding* 5 *Layer* dengan *Threshold* = 80 dan Besar Segmen =40**



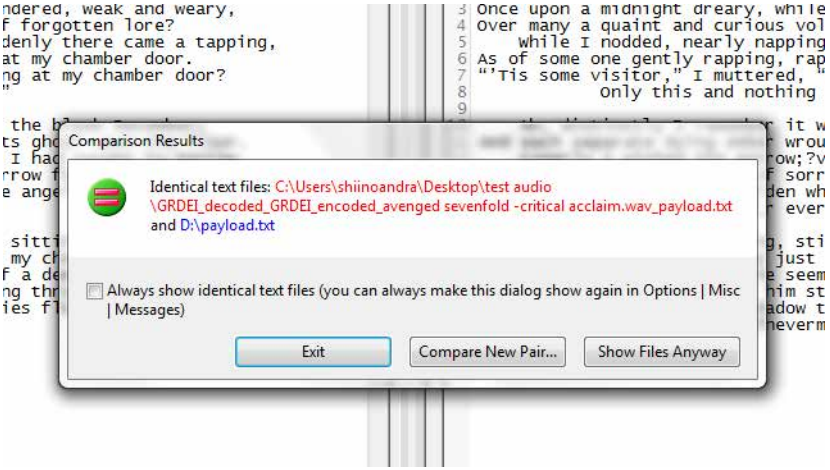

**Lampiran C.10 Plot Gelombang Audio Hasil *Encoding* 5 *Layer* dengan *Threshold* = 80 dan Besar Segmen =40**

# LAMPIRAN D

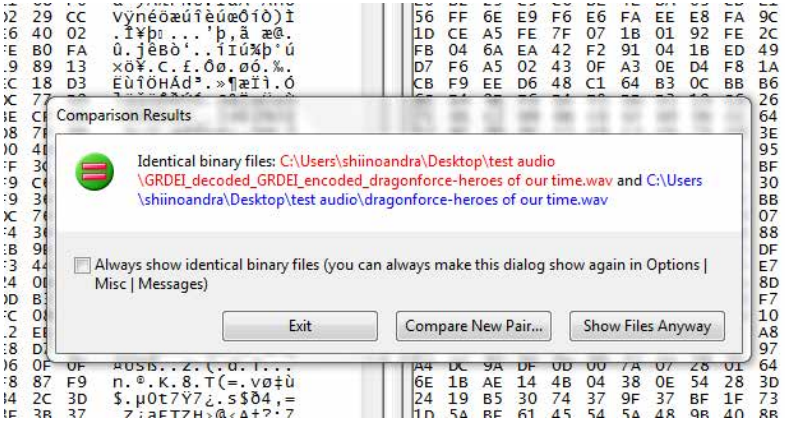
# LAMPIRAN UJI COBA 4



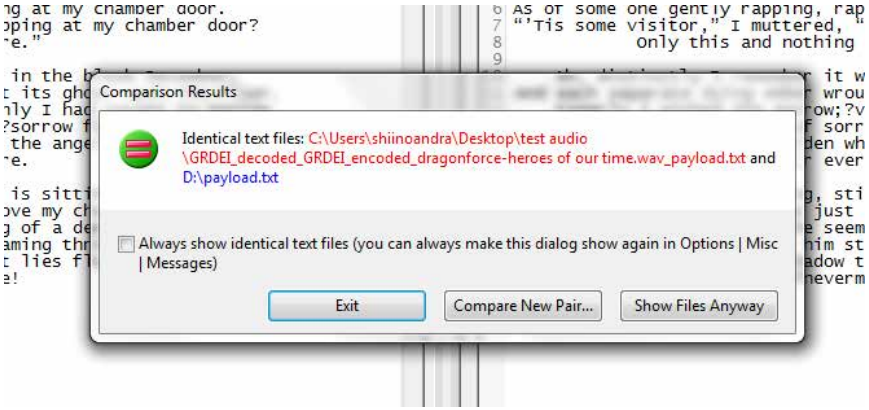

**Lampiran D.1 Perbandingan Berkas Hasil *Decode* dan Berkas Asli dengan *Threshold* = 70 dan Besar Segmen = 40**



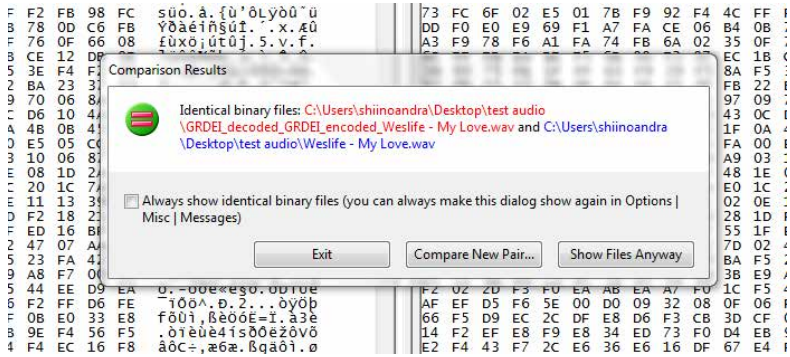

**Lampiran D.2 Perbandingan Berkas *Payload* Hasil *Decode* Dan Berkas Asli dengan *Threshold* = 70 dan Besar Segmen =40**



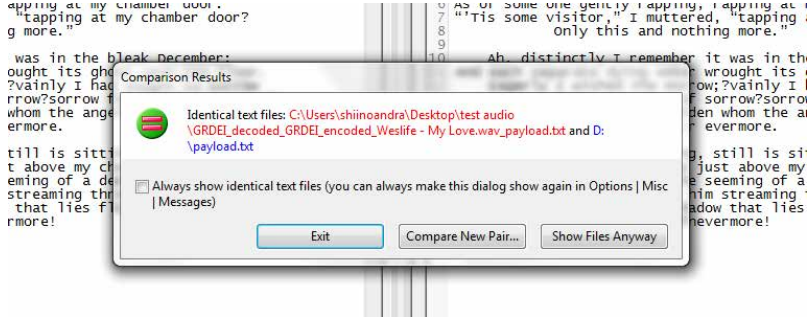

**Lampiran D.3 Perbandingan Berkas Hasil *Decode* dan Berkas Asli dengan *Threshold* = 100 dan Besar Segmen = 20**



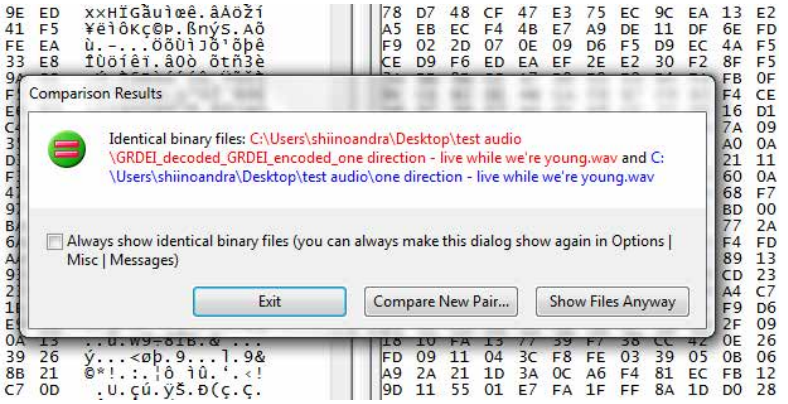

**Lampiran D.4 Perbandingan Berkas *Payload* Hasil *Decode* dan Berkas Asli dengan *Threshold* = 100 dan Besar Segmen =20**



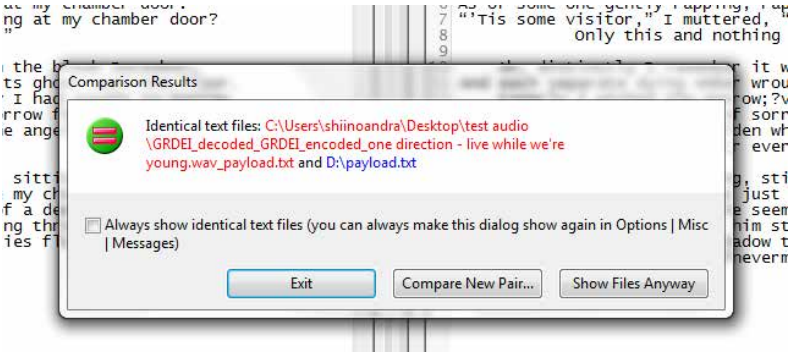

**Lampiran D.5 Perbandingan Berkas Hasil *Decode* dan Berkas Asli dengan *Threshold* = 50 dan Besar Segmen = 30**




**Lampiran D.6 Perbandingan Berkas *Payload* Hasil *Decode* dan Berkas Asli dengan *Threshold* =50 dan Besar Segmen =30**




**Lampiran D.5 Perbandingan Berkas Hasil *Decode* dan Berkas Asli dengan *Threshold* = 80 dan Besar Segmen = 20**




**Lampiran D.6 Perbandingan Berkas *Payload* Hasil *Decode* dan Berkas Asli dengan *Threshold* =80 dan Besar Segmen =20**

# LAMPIRAN E

## KODE SUMBER

```
class WavIO:
    @staticmethod
    def open(path):
        wav = read(path)
        filename =
path.split("/")[len(path.split("/"))-1]
        waveObj = Wave.Wave(filename)
        waveObj.bitrate = wav[0]
        print("jumlah sampel: "+ str(wav[0]))
        samples = wav[1]
        samples = numpy.array(samples,dtype
=numpy.uint16)
        #for i in range(len(samples)):
        #    samples[i] = samples[i] + 32767
        waveObj.samples = samples
        waveObj.max = samples.max
        waveObj.min = samples.min
        waveObj.shape = samples.shape
        waveObj.type = samples.dtype
        return waveObj
```

**Lampiran E.1. Kode Implementasi Pembacaan Berkas Audio**

```
class payloadIO:
    @staticmethod
    def open(path):
        message = open(path).read()
        return message

    @staticmethod
    def write(path,message):
        f = open(path,mode='w')
        f.write(message)
        f.close()
```

**Lampiran E.2. Kode Implementasi Pembacaan Data *Payload***

```
    @staticmethod
    def numToBinary(numArr):
        binaries = []
        for sample in numArr:
            try:
                str = bin(sample)
                sample =
operation.binaryStringtoArray(str[2:])
                while len(sample) < 16:
                    sample.insert(0,0)
                binaries.append(sample)
            except:
                continue
        return binaries
```

**Lampiran E.3. Implementasi Konversi Integer ke Biner**

```
    @staticmethod
    def makeBigit(binaryArr):
        arr = numpy.array(binaryArr,dtype=int)
        bigits = arr.transpose()
        return bigits
```

**Lampiran E.4 Kode Implementasi Pembuatan *Bigit***

```
@staticmethod
    def intel_partition(WaveArray,segLength,P=None):
        groupA=[]
        groupB=[]
        if  P == None :
            bigits = operation.makeBigit(WaveArray)
            variances=[]
            for idx,bigit in enumerate(bigits):
                counter=0
                total_var=0
                while counter < numpy.alen(bigit):
                    seg=
bigit[counter:counter+segLength]
                    var_seg = numpy.var(seg)
```

```
                    total_var += var_seg
                    counter += segLength
                variances.append((idx,total_var))
            #print(variances)
            sorted_variances = sorted(variances,key=
lambda tup:tup[1])
            #print(sorted_variances)
            #print(sorted_variances)

            partition= [0 for i in range(16)]
            #append bit ke 9 , 11 , 13 , 15 ke grup A
            #DONOT FORGET index = bit-1
            groupA.append(sorted_variances[8][0])
            groupA.append(sorted_variances[10][0])
            groupA.append(sorted_variances[12][0])
            groupA.append(sorted_variances[14][0])
            #tandai pada partition
            partition[sorted_variances[8][0]] =1
            partition[sorted_variances[10][0]] =1
            partition[sorted_variances[12][0]] =1
            partition[sorted_variances[14][0]] =1

            #append bit ke 10 , 12 , 14 , 16 ke grup
B
            #DONOT FORGET index = bit-1
            groupB.append(sorted_variances[9][0])
            groupB.append(sorted_variances[11][0])
            groupB.append(sorted_variances[13][0])
            groupB.append(sorted_variances[15][0])
            #tandai pada partition
            partition[sorted_variances[9][0]] =0
            partition[sorted_variances[11][0]] =0
            partition[sorted_variances[13][0]] =0
            partition[sorted_variances[15][0]] =0

            #bit ke 1 dan ke 3 ke grup A
            groupA.append(sorted_variances[0][0])
            groupA.append(sorted_variances[2][0])
            #bit ke 2 dan ke 4 ke grup B
            groupB.append(sorted_variances[1][0])
```

```
            groupB.append(sorted_variances[3][0])
            #tandai pada partition
            partition[sorted_variances[0][0]] =1
            partition[sorted_variances[1][0]] =0
            partition[sorted_variances[2][0]] =1
            partition[sorted_variances[3][0]] =0

            #sisanya
            groupA.append(sorted_variances[4][0])
            groupA.append(sorted_variances[5][0])
            groupB.append(sorted_variances[6][0])
            groupB.append(sorted_variances[7][0])

            #tandai pada partition
            partition[sorted_variances[4][0]] =1
            partition[sorted_variances[5][0]] =1
            partition[sorted_variances[6][0]] =0
            partition[sorted_variances[7][0]] =0

            #sort indeks yang disimpan pada kedua
grup
            groupA = sorted(groupA)
            #print(groupA)
            groupB = sorted(groupB)
            #print(groupB)
            #print(partition)

            #susun kembali bigit berdasarkan index
            M1 = [bigits[i] for i in groupA]
            M1 = numpy.transpose(M1)
            M2 = [bigits[i] for i in groupB]
            M2 = numpy.transpose(M2)
            return (M1,M2,partition)
        else:
            bigits = operation.makeBigit(WaveArray)
            for idx,i in enumerate(P):
                if i == 1:
                    groupA.append(idx)
                else:
                    groupB.append(idx)
```

```
            M1 = [bigits[i] for i in groupA]
            M1 = numpy.transpose(M1)
            M2 = [bigits[i] for i in groupB]
            M2 = numpy.transpose(M2)

            return (M1,M2,P)
```

**Lampiran E.5 Kode Implementasi Penghitungan *Variance* dan Partisi Sampel**

```
            payload_seg1 = payload_i[:kapasitas_M1]
            payload_seg2 =
payload_i[kapasitas_M1:len(payload_i)]
```

**Lampiran E.6 Kode Implementasi Partisi *Payload***

```
    @staticmethod
    def binaryTonum(binaryArr):
        return
[int(operation.binaryArraytoString(sample),2) for
sample in binaryArr]

    @staticmethod
    def binaryArraytoString(binaryArray):
        s= ''.join([str(x) for x in binaryArray])
        return s
```

**Lampiran E.7 Kode Implementasi Konversi Biner ke Integer**

```
for x in range (n/segLength):
        seg =
waveArray[counter_seg:counter_seg+segLength]

        counter_seg += segLength
```

**Lampiran E.8. Kode Implementasi Pembagian Segmen**

```
def checkCapacity(waveArray,segLength,threshold):
    capacity = 0
    counter = 0
    for x in range (len(waveArray)/segLength):
        seg = waveArray[counter:counter+segLength]
        counter += segLength
        flag = checkBlock(seg)
        flag2 = checkThreshold(seg,threshold)
        if (flag == 1 or flag == 0) and flag2!=1:
            capacity+=(segLength-1)

    if(capacity <0):capacity =0
    return capacity
```

**Lampiran E.9. Kode Implementasi Pengecekan Kapasitas**

```
def checkThreshold(waveArray,threshold):
     flag=0
     n = len(waveArray)-1
     for i in range(n):
        diff = waveArray[i+1]-waveArray[0]
        (rdiff,rflag) = rdei(diff)
        if(abs(rdiff)>threshold):
            flag = 1
     return flag
```

**Lampiran E.10 Kode Implementasi Pengecekan Threshold**

```
def checkBlock(waveArray):
    flag=0
    n = len(waveArray)-1
    for i in range(n):
        diff = waveArray[i+1]-waveArray[0]
        (rdiff,rflag) = rdei(diff)
        if rdiff+waveArray[0] <255 or
waveArray[0]+rdiff>0:
            ri =  (2*rdiff)+1
            if waveArray[0]+ ri <=0 or
waveArray[0]+ri >=255:
                flag = 2
    if(flag == 1):
        for i in range(n):
            diff = waveArray[i+1]-waveArray[0]
            (rdiff,rflag) = rdei(diff)
            if (2*math.floor((rdiff/2))) + 1 != rdiff
- 1:
                flag ==2

    return flag
```

**Lampiran E.11. Kode Implementasi Pengecekan Overflow dan Underflow**

```
def rdei(angka_input):
    angka = abs(angka_input)
    if angka>3:
        rd = angka -
pow(2,((math.floor(math.log(angka,2)))-1))  -
pow(2,(math.floor(math.log(angka,2))-2))
        flag =0
        if
math.pow(2,(math.floor(math.log(angka,2)))) /
math.pow(2,(math.floor(math.log(rd,2)))) == 4:
            flag = 1
        if
math.pow(2,(math.floor(math.log(angka,2)))) /
math.pow(2,(math.floor(math.log(rd,2)))) == 1:
```

```
            flag = 2
    else:
        rd = angka
        flag = 3
    if(angka_input<0):
        return (-rd,flag)
    else:
        return(rd,flag)
```

**Lampiran E.12. Implementasi *Improved RDE***

```
def encode_block(waveArray,message,validflag):

    flag = 0
    n = len(waveArray)-1
    diffs = []
    encoded = []
    mapFlag=[]

    for i in range(n):
        diff = waveArray[i+1]-waveArray[0]
        (rdiff,flag) = rdei(diff)
        diffs.append(rdiff)
        mapFlag.append(flag)
    encoded.append(waveArray[0])

    for i in range(n):
        if(i<len(message)):
            if validflag == 0:
                ri =  (2*diffs[i])+ message[i]
            elif validflag ==1:
                ri =  (2*math.floor((diffs[i]/2))) +
message[i]

        else:
            if validflag == 0:
                ri =  (2*diffs[i])
            elif validflag ==1:
```

```
                ri =  (2*math.floor((diffs[i]/2)))

        encoded.append(ui)
    return (encoded,mapFlag)
```

**Lampiran E.13. Kode Implementasi Penyisipan Bit**

```
    def populate(self):
        while len(self.populasi)<self.jumlahPopulasi:
            bits = []
            for i in
range((self.segBitLength+self.thresBitLength+self.lay
erBitLength)):
                bit = random.randint(0,1)
                bits.append(bit)
            stringdata =
op.operation.binaryArraytoString(bits)
            segment_length =
int(stringdata[0:self.segBitLength],2)
            threshold =
int(stringdata[self.segBitLength:self.segBitLength+se
lf.thresBitLength],2)
            jml_layer =
int(stringdata[self.segBitLength+self.thresBitLength:
],2)
            if jml_layer >0 and segment_length >0:
                flag = 0
                for k in self.populasi:
                    if k.data == stringdata:
                        flag = 1
                if flag ==0:
                    self.add_kromosom(stringdata)
```

**Lampiran E.14. Kode Implementasi Pembentukan Populasi Awal**

```
def getFitness(self):
        orig_wav = self.wave
        payload = self.payload
        for k in self.populasi:
            print k.data
            self.outputString+=str(k.data)+"\n"
            segment_length =
int(k.data[0:self.segBitLength],2)
            threshold =
int(k.data[self.segBitLength:self.segBitLength+self.t
hresBitLength],2)
            jml_layer =
int(k.data[self.segBitLength+self.thresBitLength:],2)
            print("seg length: "+str(segment_length))
            self.outputString+="seg length:
"+str(segment_length)+"\n"
            print("threshold: "+str(threshold))
            self.outputString+="threshold:
"+str(threshold)+"\n"
            print("jml layer: "+str(jml_layer))
            self.outputString+="jml layer:
"+str(jml_layer)+"\n"
            output =
Core.Wavegano.multilayer_encode(payload,orig_wav,thre
shold,segment_length,10,"GRDEI",jml_layer)
            new_wave = output[4]
            cap = output[5]
            print("kapasitas : "+str(cap))
            self.outputString+="kapasitas :
"+str(cap)+"\n\n"
            if(len(new_wave)>1):
                k.fitnessVal = cap
                if(k.fitnessVal >
self.bestSolution.fitnessVal):
                    self.bestSolution = k
```

**Lampiran E.15. Implementasi Perhitungan *Fitness Value***

```
    def crossover(self,k1,k2):
        n=len(k1.data)
        index = random.randint(0,n-1)
        front1 = k1.data[0:index]
        front2 = k2.data[0:index]
        rear1 = k1.data[index:n]
        rear2 = k2.data[index:n]
        newK1 = front1+rear2
        newK2 = front2+rear1
        krom1 = kromosom.kromosom()
        krom1.data = newK1
        krom2 = kromosom.kromosom()
        krom2.data = newK2
        return (krom1,krom2)
```

**Lampiran E.16 Kode Implementasi Crossover Populasi**

```
  def mutate(self):
      i = random.randint(0,len(self.data)-1)
      tobe_mutated =
op.operation.binaryStringtoArray(self.data)
      if tobe_mutated[i] == 0:
          tobe_mutated[i] =1;
      else:
          tobe_mutated[i] = 0
      self.data                                          =
op.operation.binaryArraytoString(tobe_mutated)
```

**Lampiran E.17 Kode Implementasi Mutasi Pada Kromosom**

```
    while len(newPopulation) < len(self.populasi) :
            parents = self.selection(sortedpopulasi)
            (newchild1,newchild2) =
self.crossover(parents[0],parents[1])
            if random.randint(1,1001) == 5:
                newchild1.mutate()
            if random.randint(1,1001) == 5:
                newchild2.mutate()
            flag = 0
            for y in newPopulation:
                    if y.data ==newchild1.data or
y.data==newchild2.data:
                        flag = 1
                    segment_length =
int(y.data[0:self.segBitLength],2)
                    jml_layer =
int(y.data[self.segBitLength+self.thresBitLength:],2)
                    if jml_layer ==0  or
segment_length ==0:
                        flag =1
                    if segment_length ==1 or
jml_layer ==1:
                        flag = 1
            if flag ==0:
                newPopulation.append(newchild1)
                newPopulation.append(newchild2)
        self.populasi = newPopulation
```

**Lampiran E.18 Kode Implementasi Pembentukan Populasi Baru**

```
params = {"method" : method ,  "locmap":locmap_list ,
"partition":P , "payload_sizes":payload_sizes ,
"layers":int(self.layersizeval.text()) ,
"segment_size":int(self.seglength.text()) }
            with
open(self.encode_dir+self.wavefile.name+".param",
'w') as outfile:
                json.dump(params, outfile)
```

**Gambar 4.19. Kode Implementasi penulisan berkas parameter**

```
with open(path) as json_file:
                self.params = json.load(json_file)
        segment_size = self.params["segment_size"]
        payload_sizes = self.params["payload_sizes"]
        Partition = self.params["partition"]
        method = self.params["method"]
        n_layer =self.params["layers"]
```

**Lampiran E.20. Kode Implementasi Pembacaan Data Parameter**

```
  seg_counter =0
    flag_counter =0
    reduce_counter =0
    for x in range(n/segLength):
        seg =
waveArray[seg_counter:seg_counter+segLength]
```

**Lampiran E.21. Kode Implementasi Pembagian Sampel Encoded Menjadi Segmen**

```
if locMap[flag_counter] == 0:
            rmap =
reducedMap[reduce_counter:reduce_counter+(segLength-
1)]
            (w,m) =
decode_block(seg,rmap,locMap[flag_counter])
            decoded.extend(w)
            messages.extend(m)
            reduce_counter+=(segLength-1)
        else :
            decoded.extend(seg)
        flag_counter+=1
```

**Lampiran E.22 Kode Implementasi Pengecekan Tanda Pada *Location Map***

```
        if validFlag  == 0:
                rdiff = (diff-w)/2
        elif validFlag==1:
                        rdiff = (diff-w)
```

**Lampiran E.24 Kode Implementasi Pengecekan Tanda Pada *Reduced Map***

```
def g(x):
   return  2*f(x)-x
. . .

for i in range(n):
        diff = waveArray[i+1] -  waveArray[0]
        w = g(diff)
```

**Lampiran E.25 Kode Implementasi Pengembalian Bit Data *Payload***

# BIODATA PENULIS

Penulis dilahirkan di Jakarta, 2 September 1994, merupakan anak tunggal. Penulis telah menempuh pendidikan formal yaitu TK Al-Azhar Rawamangun (1998-2000), SD Islam Al-Azhar Rawamangun (2000-2006), SMP Islam Al-Azhar Kelapa Gading (2006-2008), SMA Negeri 68 Jakarta (2008-2011), dan mahasiswa S1 Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya (2011-2015). Penulis juga aktif mengikuti lomba-lomba berbasis Teknologi Informasi, di antaranya adalah GEMASTIK 2013 bidang Sekuritas Jaringan, GEMASTIK 2014 bidang Sekuritas Jaringan, dan COMPFEST UI 2014 bidang Pembuatan Game. Penulis yang memiliki hobi bermain *game* jejepangan ini merupakan mahasiswa yang aktif dalam organisasi Himpunan Mahasiswa Teknik Computer – Informatika ITS (HTMC) dan juga sering terlihat muncul di berbagai festival budaya jepang. Penulis dapat dihubungi melalui surel shiinoandra@gmail.com