**ANALISIS FAKTOR *ROBUSTNESS* DAN *FIDELITY* PADA METODE *LEAST SIGNIFICANT BIT*, *DISCRETE COSINES TRANSFORM*, DAN *DISCRETE HAAR WAVELET TRANSFORM* DALAM IMPLEMENTASI STEGANOGRAFI PADA CITRA DIGITAL**

**TUGAS AKHIR**

Tugas Akhir ini sebagai salah satu syarat untuk memperoleh gelar sarjana

Informatika Universitas Pembangunan Nasional “Veteran” Yogyakarta

****

Disusun Oleh:

**Ardian Tri Kusuma**

**123160035**

**PROGRAM STUDI INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK INDUSTRI**

**UNIVERSITAS PEMBANGUNAN NASIONAL “VETERAN”**

**YOGYAKARTA**

**2020**

# HALAMAN JUDUL

**ANALISIS FAKTOR *ROBUSTNESS* DAN *FIDELITY* PADA METODE *LEAST SIGNIFICANT BIT*, *DISCRETE COSINES TRANSFORM*, DAN *DISCRETE HAAR WAVELET TRANSFORM* DALAM IMPLEMENTASI STEGANOGRAFI PADA CITRA DIGITAL**

**TUGAS AKHIR**

Tugas Akhir ini sebagai salah satu syarat untuk memperoleh gelar sarjana

Informatika Universitas Pembangunan Nasional “Veteran” Yogyakarta

****

Disusun Oleh:

**Ardian Tri Kusuma**

**123160035**

**PROGRAM STUDI INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK INDUSTRI**

**UNIVERSITAS PEMBANGUNAN NASIONAL “VETERAN”**

**YOGYAKARTA**

**2020**

# HALAMAN PENGESAHAN PEMBIMBING

# HALAMAN PENGESAHAN PENGUJI

# SURAT KARYA ASLI TUGAS AKHIR

# PERNYATAAN BEBAS PLAGIAT

# HAL PERSEMBAHAN

# ABSTRAK

# KATA PENGANTAR

# DAFTAR ISI

[HALAMAN JUDUL i](#_Toc48473944)

[HALAMAN PENGESAHAN PEMBIMBING ii](#_Toc48473945)

[HALAMAN PENGESAHAN PENGUJI iii](#_Toc48473946)

[SURAT KARYA ASLI TUGAS AKHIR iv](#_Toc48473947)

[PERNYATAAN BEBAS PLAGIAT v](#_Toc48473948)

[HAL PERSEMBAHAN vi](#_Toc48473949)

[ABSTRAK vii](#_Toc48473950)

[KATA PENGANTAR viii](#_Toc48473951)

[DAFTAR ISI ix](#_Toc48473952)

[DAFTAR GAMBAR xi](#_Toc48473953)

[DAFTAR TABEL xii](#_Toc48473954)

[DAFTAR MODUL PROGRAM xiii](#_Toc48473955)

[BAB I PENDAHULUAN 1](#_Toc48473956)

[1.1 Latar Belakang 1](#_Toc48473957)

[1.1 Perumusan Masalah 3](#_Toc48473958)

[1.2 Batasan Masalah 4](#_Toc48473959)

[1.3 Tujuan Penelitian 4](#_Toc48473960)

[1.4 Manfaat Penelitian 4](#_Toc48473961)

[1.5 Metodologi Penelitian dan Pengembangan Sistem 5](#_Toc48473962)

[1.5.1 Metodologi Penelitian 5](#_Toc48473963)

[1.5.2 Metode Pengembangan Sistem 5](#_Toc48473964)

[1.6 Sistematika Penulisan 6](#_Toc48473965)

[BAB II TINJAUAN PUSTAKA 7](#_Toc48473966)

[2.1 Citra Digital 7](#_Toc48473967)

[2.1.1 Ruang Warna RGB 8](#_Toc48473968)

[2.2 Steganografi 10](#_Toc48473969)

[2.2.1 Least Significant Bit 11](#_Toc48473970)

[2.2.2 Discrete Cosine Transform 12](#_Toc48473971)

[2.2.3 Discrete Haar Wavelet Transform 14](#_Toc48473972)

[2.2.4 Pengujian Kualitas Metode Steganografi 16](#_Toc48473973)

[2.3 Python 18](#_Toc48473974)

[2.4 Tinjauan Literatur 22](#_Toc48473975)

[BAB III METODOLOGI PENELITIAN DAN PENGEMBANGAN SISTEM 24](#_Toc48473976)

[3.1 Metodologi Penelitian 24](#_Toc48473977)

[3.1.1 Identifikasi Masalah 24](#_Toc48473978)

[3.1.2 Studi Pustaka **Error! Bookmark not defined.**](#_Toc48473979)

[3.2 Metodologi Pengembangan Sistem 25](#_Toc48473980)

[3.2.1 Komunikasi 25](#_Toc48473981)

[3.2.2 Quick Design 28](#_Toc48473982)

[BAB IV HASIL, PENGUJIAN, DAN PEMBAHASAN 63](#_Toc48473983)

[4.1 Hasil Penelitian 74](#_Toc48473984)

[BAB V PENUTUP 79](#_Toc48473985)

[5.1 Kesimpulan 79](#_Toc48473986)

[DAFTAR PUSTAKA 81](#_Toc48473987)

[LAMPIRAN 82](#_Toc48473988)

# DAFTAR GAMBAR

[**Gambar 2.1.** Contoh tata letak pixel dalam matriks 8](#_Toc46260623)

[**Gambar 2.2** Warna RGB 9](#_Toc46260624)

[**Gambar 2.3.** Ilustrasi penyisipan bit LSB 12](#_Toc46260625)

[**Gambar 2.4.** Dekomposisi Wavelet tiga tingkat **Error! Bookmark not defined.**](#_Toc46260626)

[**Gambar 2.5.** DWT dua dimensi **Error! Bookmark not defined.**](#_Toc46260627)

[**Gambar 3.1** Alur Tahap Penelitian 24](#_Toc46260628)

[**Gambar 3.2** Kerangka Kerja Tahap Pengembangan Sistem **Error! Bookmark not defined.**](#_Toc46260629)

[**Gambar 3.3** Perancangan Arsitektur Sistem 28](#_Toc46260630)

[**Gambar 3.4** Flowchart Utama Sistem 29](#_Toc46260631)

[**Gambar 3.5** Flowchart Proses Encode 30](#_Toc46260632)

[**Gambar 3.6** Flowchart sub proses encode LSB 31](#_Toc46260633)

[**Gambar 3.7** Flowchart sub proses DCT 35](#_Toc46260634)

[**Gambar 3.8** Flowchart sub proses Inverse DCT 39](#_Toc46260635)

[**Gambar 3.9** Flowchart sub proses DWT 42](#_Toc46260636)

[**Gambar 3.10** Ilustrasi subband DWT 43](#_Toc46260637)

[**Gambar 3.11** Flowchart sub proses Inverse DWT 46](#_Toc46260638)

[Gambar 3.12 Wireframe Tampilan Awal 54](#_Toc46260639)

[Gambar 3.13 Wireframe Tampilan Encode 55](#_Toc46260640)

[Gambar 3.14 Wireframe Tampilan Hasil Encode 56](#_Toc46260641)

[Gambar 3.15 Wireframe Tampilan Decode 57](#_Toc46260642)

[Gambar 3.16 Wireframe Tampilan Identifikasi 58](#_Toc46260643)

# DAFTAR TABEL

[**Tabel 2.1.** Tinjauan Literatur 22](#_Toc46260648)

[**Tabel 2.2.** Lanjutan Tinjauan Literatur 23](#_Toc46260649)

# DAFTAR MODUL PROGRAM

# BAB I PENDAHULUAN

## Latar Belakang

Pentingnya kerahasiaan suatu informasi telah menjadi suatu perhatian tersendiri dari masa ke masa. Manusia berusaha mencari cara bagaimana merahasiakan informasi terhadap pihak yang dianggap tidak berhak untuk mengetahuinya. Berbagai cara telah dilakukan oleh bangsa – bangsa kuno untuk merahasiakan informasi, karena informasi yang jatuh ke orang yang tidak berhak akan menimbulkan kerugian. Sebagai contoh Julius Caesar, kaisar romawi, telah menggunakan suatu metode untuk mengacak pesan sebelum pesan itu dikirimkan kepada penerimanya. Tujuannya adalah agar orang yang tidak berhak menerima pesan tersebut tidak dapat membacanya (Tarigan, 2015).

Zaman sekarang informasi tidak hanya dapat disandikan, tetapi dapat juga disisipkan kedalam media digital. Teknik menyisipkan pesan dikenal dengan nama steganografi. Steganografi sebagai ilmu dan seni untuk menyembunyikan informasi sehingga informasi yang bersifat rahasia tidak dapat diketahui oleh orang lain, kecuali pengirim dan penerima (Atoum, Ibrahim, & M. Ahmad, 2012). Proses steganografi biasanya melibatkan penyandian atau kriptografi. Proses yang dilakukan yaitu dengan enkripsi *plaintext* terlebih dahulu menjadi *Byte cipher* atau pesan rahasia. Kemudian *Byte cipher* disisipkan pada media digital berupa teks, audio, citra atau protocol.

Beberapa penelitian telah dilakukan berkaitan dengan pengamanan data dengan menerapkan beberapa algoritma. Beberaa metode diantaranya adalah algoritma *Least Significant Bit, Discrete Cosine Transform,* dan *Discrete Wavelete Transform*. Dalam penelitian Danuputri (2018) menggunakan algoritma *Least Significant Bit* yang diperkuat dengan adanya algoritma *Vignere* *Key* untuk melakukan enkripsi terhadap dokumen berekstensi \*.doc, \*.docx, \*.xls, \*.xlsx, \*.txt, dan \*.pdf untuk dihasilkan *plaintext* yang kemudian menggunakan *Least Significant Bit* untuk menyisipkannya ke dalam gambar. Citra stego yang dihasilkan tidak memliki perbedaan yang signifikan dan data yang disisipkan tidak mengalami perubahan ketika diekstrak. Namun terdapat penambahan ukuran file pada citra stego yang dihasilkan. Penelitian serupa terkait algoritma *Least Significant Bit* juga pernah dilakukan oleh Sanawira dan Purnomo (2016) yang menggunakan algoritma tersebut untuk menyembunyikan pesan ke dalam media citra sebagai *cover object* dengan gabungan metode *Redundant Pattern Encoding* dan hasil uji yang didapat tergolong memuaskan. Namun hasil tersebut akan terpengaruh jika *stego object* diserang atau dimanipulasi. Selain itu, terdapat juga penelitian oleh Garno dan Solehudin (2017) yang mengkombinasikan metode *Discrete Cosines Transform* danInterpolasi *Bilinear* untuk menyembunyikan pesan. Pada penelitian tersebut Garno berhasil menyisipkan pesan ke dalam citra digital, namun terjadi distorsi yang cukup signifikan pada *stego object* sehingga nilai hasil uji kurang memuaskan. Penelitian lain tentang steganografi juga pernah dilakukan dengan metode *Discrete Wavelete Transform* dengan tipe *Haar* oleh Zagade dan Bhosale (2014). Citra stego yang dihasilkan tergolong bagus dengan nilai PSNR yang tinggi. Metode DWT dan DCT juga pernah dilakukan oleh Faza dkk. (2016) yaitu dengan menggabungkan DCT dan DWT dan citra yang dihasilkan sudah tergolong bagus. Namun gabungan tersebut tidak terlalu memuaskan bila dibandingkan dengan metode DWT saja.

Penelitian-penelitian tersebut telah menggunakan algoritma-algoritma steganografi yang mempunyai kelebihan masing-masing. Dengan banyaknya metode-metode yang dapat digunakan, maka akan sangat membantu pengguna dalam penyembunyian pesan. Namun, melihat dari kelebihan saja tidaklah cukup. Perlu diketahui bahwa *stego* *object* tidak selamanya aman dan tidak menimbulkan kecurigaan. Ketika *stego* *object* dikirimkan menggunakan perantara pihak ketiga, ada kemungkinan *stego* *object* tersebut rusak atau isinya telah dilihat oleh pihak lain tanpa sepengetahuan pembuat. Hal tersebut terjadi karena setiap metode memiliki tingkat *robustness* (ketahanan) dan *fidelity* (ketidaknampakan perbedaan) yang rendah.

Dari permasalahan yang telah dipaparkan, maka diperlukan perbandingan metode-metode tersebut sehingga dapat diketahui metode mana yang memiliki tingkat *robustness* dan *fidelity* yang baik untuk digunakan dalam steganografi maupun *watermarking* pada suatu objek. Dalam penelitian ini, metode yang akan diteliti adalah *Least* *Significant* *Bit*, *Discrete* *Cosines* *Transform*, *Discrete* *Haar* *Wavelet* *Transform*, serta gabungan dari algoritma tersebut yang menghasilkan sebuah *modified* *algorithm* yang diharapkan dapat meningkatkan tingkat *robustness* dan *fidelity* dari *stego* *object* yang dibuat. Pemilihan metode-metode tersebut dilandasi oleh beberapa penelititan tentang LSB, DCT, maupun DWT sebelumnya yang hanya menggunakan pengujian kualitas PSNR dan MSE untuk menentukan baik tidaknya metode-metode tersebut untuk diimplementasikan serta pengujian ketahanan yang hanya menyebutkan bisa tidaknya pesan rahasia diekstrak kembali tanpa disebutkan apa yang menyebabkan hal tersebut terjadi.

Oleh karena itu disusunlah penelitian dengan judul “Analisis Faktor *Robustness* dan *Fidelity* pada Metode *Least Significant Bit*, *Discrete Cosines Transform*, dan *Discrete Haar Wavelet Transform* dalam Implementasi Steganografi pada Citra Digital”.



## Perumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka dapat diambil rumusan masalah dalam penelitian ini adalah bagaimana perbandingan kualitas *robustness* pada *hidden object* setelah dilakukan serangan pada *stego* *object* yang menggunakan metode *Least Significant Bit*, *Discrete Cosine Transform*, dan *Discrete Haar Wavelet Transform* serta bagaimana perbandingan tingkat *fidelity* dari *stego object* setelah disisipkan *hidden object.*

## Batasan Masalah

Pada penelitian ini, permasalahan dibatasi sebagai berikut:

1. Citra yang digunakan menggunakan ekstensi yang umum digunakan yaitu \*.jpg, \*.png, dan \*.bmp.
2. *Hidden* *object* atau *embedded* *message* berupa teks ukuran pendek dan citra lain sebagai representasi teks ukuran panjang.
3. Penelitian ini lebih berfokus pada tingkat *robustness* dari *hidden object* dan *fidelity* dari *stego object* sehingga *hidden object* tidak perludienkripsi*/*disandikan*.*

## Tujuan Penelitian

Tujuan pada penelitian ini adalah sebagai berikut:

1. Mengetahui hasil identifikasi uji kualitas *robustness hidden object* setelah diimplementasikan metode *Least Significant Bit*, *Discrete Cosine Transform*, dan *Discrete Haar Wavelet Transform* dan manipulasi *stego object.*
2. Mengetahui hasil identifikasi tingkat *fidelity stego object* setelah diimplementasikan metode *Least Significant Bit*, *Discrete Cosine Transform*, dan *Discrete Haar Wavelet Transform* dan menentukan metode manakah yang paling baik dalam menyembunyikan *hidden object*.

## Manfaat Penelitian

Manfaat yang dapat diperoleh dari penelitian ini adalah:

* + 1. Dapat mengetahui kelebihan serta kelemahan pada metode *Least Significant Bit*, *Discrete Cosine Transform*, dan *Discrete Haar Wavelet Transform* berdasarkan faktor *robustness* dan *fidelity* pada *stego* *object*.
    2. Dapat menjadi referensi dalam pemilihan metode steganografi yang dapat digunakan dalam kehidupan sehari-hari sehingga tidak perlu khawatir mengirimkan citra stego melalui perantara pihak ketiga.

## Metodologi Penelitian dan Pengembangan Sistem

Metodologi penelitian berisi langkah-langkah yang digunakan dalam penelitian ini agar terstruktur dengan baik. Pengembangan system merupakan bagian dari metode penelitian yang digunakan pada penelitian ini.

### Metodologi Penelitian

Metode yang digunakan dalam penelitian ini menggunakan penelititan kuantitatif dikarenakan terdapat beberapa pengujian yang akan dilakukan serta data yang digunakan adalah data hasil studi pustaka sebagai dasar teori penelitian yang dilakukan serta data *dummy* atau data sekunder sebagai media yang akan diuji. Data sekunder didapatkan dengan cara membuat sendiri citra uji serta mendapatkan citra uji dengan cara mengunduh citra dari internet.

### Metode Pengembangan Sistem

Metode yang digunakan dalam pengembangan sistem ini adalah metode *Prototype* (purwarupa). Model *prototype* dapat diuraikan secara singkat sebagai berikut:

1. *Communication*/Komunikasi

Tahap menggali informasi kebutuhan sistem yang akan dibuat.

1. *Quick Plan and Modeling Quick Design*/Pemodelan Rancangan Cepat

Tahap merancang sistem.

1. *Construction of Prototype*/Pembuatan Purwarupa

Tahap dimulainya membangun sebuah purwarupa/*prototype* berdasarkan hasil perancangan.

1. *Deployment Delivery and Feedback*/Penyerahan dan Pemberian Umpan Balik

Tahap perilisan produk hasil purwarupa untuk dievaluasi dan akan dilakukan perbaikan.

## Sistematika Penulisan

Sistematika penulisan yang digunakan dalam menyusun laporan penelitian ini adalah sebagai berikut :

**BAB I Pendahuluan**

Pada bagian ini membahas tentang latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

**BAB II Tinjauan Pustaka**

Tinjauan pustaka memuat tentang dasar teori yang digunakan untuk analisis dan perancangan sistem serta implementasi pada penelitian ini. Selain itu juga sebagai bahan referensi dan pondasi untuk memperkuat argumentasi dalam penelitian ini. Teori-teori yang sesuai dengan penelitian ini antara lain steganografi*,* *Least Significant Bit*, *Discrete Cosines Transform ,*dan *Discrete Haar Wavelet Transform*, dan *Peak Signal to Noise Ratio*.

**BAB III Metode Penelitian dan Pengembangan Sistem**

Pada bagian ini akan membahas mengenai analisa dan perancangan sistem hingga implementasi system yang akan dibangun.

**BAB IV Hasil, Pengujian dan Pembahasan**

Pada bab ini akan menyajikan hasil penelitian yang berisi hasil implementasi dari perancangan yang telah dibuat pada bab sebelumnya. Selain itu berisi pengujian terhadap hasil penelitian beserta pembahasannya.

**BAB V Penutup**

Pada bagian ini berisi kesimpulan dari hasil penelitian dan saran yang diajukan oleh penulis untuk pengembangan penelitian selanjutnya.

# BAB II TINJAUAN PUSTAKA

## Citra Digital

Secara harafiah, citra (*image*) adalah gambar pada bidang dwimatra (dua dimensi). Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dwimatra. Sumber cahaya menerangi objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh oleh alat-alat optik, misalnya mata pada manusia, kamera, pemindai (*scanner*), dan sebagainya, sehingga bayangan objek yang disebut citra tersebut terekam (Munir, 2004).

Agar dapat diolah dengan dengan komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi citra dari fungsi malar (kontinu) menjadi nilai-nilai diskrit disebut digitalisasi. Citra yang dihasilkan inilah yang disebut citra digital (*digital* *image*). Pada umumnya citra digital berbentuk empat persegi panjang, dan dimensi ukurannya dinyatakan sebagai tinggi x lebar (atau lebar x panjang). Citra digital yang berukuran NxM lazim dinyatakan dengan matriks yang berukuran N baris dan M kolom sebagai berikut:

|  |  |  |
| --- | --- | --- |
|  | ………………… | ( 2.1 ) |

Indeks baris (i) dan indeks kolom (j) menyatakan suatu koordinat titik pada citra, sedangkan f(i, j) merupakan intensitas (derajat keabuan) pada titik (i, j). Masing-masing elemen pada citra digital (berarti elemen matriks) disebut *image* *element*, *picture* *element* atau *pixel* atau pel. Jadi, citra yang berukuran NxM mempunyai NM buah *pixel*. Sebagai contoh, misalkan sebuah berukuran 256x256 *pixel* dan direpresentasikan secara numerik dengan matriks yang terdiri dari 256 buah baris (di-indeks dari 0 sampai 255) dan 256 buah kolom (di-indeks dari 0 sampai 255) seperti contoh berikut:



**Gambar 2.1.** Contoh tata letak pixel dalam matriks

*Pixel* pertama pada koordinat (0, 0) mempunyai nilai intensitas 0 yang berarti warna *pixel* tersebut hitam, *pixel* kedua pada koordinat (0, 1) mempunyai intensitas 134 yang berarti warnanya antara hitam dan putih, dan seterusnya.

### Ruang Warna RGB

Warna pada dasarnya merupakan hasil persepsi dari cahaya dalam spektrum wilayah yang terlihat oleh retina mata, dan memiliki Panjang gelombang antara 400nm sampai dengan 700nm. Ruang warna atau sering disebut model warna merupakan sebuah cara atau metode untuk mengatur, membuat, dan memvisualisasikan warna (Ford & Roberts, 1998).

Salah satu dari ruang warna adalah RGB. RGB sering digunakan didalam sebagian besar aplikasi komputer karena dengan ruang warna ini, tidak diperlukan transformasi untuk menampilkan informasi di layar monitor. Alasan diatas juga menyebabkan RGB banyak dimanfaatkan sebagai ruang warna dasar bagi sebagian besar aplikasi. Model warna RGB adalah model warna berdasarkan konsep penambahan kuat cahaya primer yaitu *Red*, *Green* dan *Blue*. Dalam suatu ruang yang sama sekali tidak ada cahaya, maka ruangan tersebut adalah gelap total. Tidak ada signal gelombang cahaya yang diserap oleh mata kita atau RGB (0,0,0). Apabila ditambahkan cahaya merah pada ruangan tersebut, maka ruangan akan berubah warna menjadi merah misalnya RGB (255,0.0), semua benda dalam ruangan tersebut hanya dapat terlihat berwarna merah. Demikian juga apabila cahaya diganti dengan hijau atau biru.

Apabila diberikan 2 macam cahaya primer dalam ruangan tersebut seperti (merah dan hijau), atau (merah dan biru) atau (hijau dan biru), maka ruangan akan berubah warna masing-masing menjadi kuning, atau magenta, atau cyan. Warna-warna yang dibentuk oleh kombinasi dua macam cahaya tersebut disebut warna sekunder. Warna tersier adalah warna yang hanya dapat terlihat apabila terdapat tiga cahaya primer, jadi apabila dinonaktifkan salah satu cahaya, maka benda tersebut berubah warna. Contoh warna tersier adalah abu-abu, putih.



**Gambar 2.2** Warna RGB (Pressman, 2010)

Pada perhitungan dalam program-program komputer model warna direpresentasi dengan nilai komponennya, seperti dalam RGB (rgb) masing-masing nilai antara 0 hingga 255 sesuai dengan urusan masing-masing yaitu pertama *red*, kedua *green* dan ketiga adalah nilai *blue* dengan demikian masing-masing komponen ada 256 tingkat. Apabila dikombinasikan maka ada 256 x 256 x 256 atau 16777216 kombinasi warna RGB yang dapat dibentuk.

### Jenis Ekstensi Citra Digital

Menurut Hidayatullah (2017), dalam penyimpanan sebuah citra digital ke dalam file, maka terdapat beberapa format file yang bisa digunakan berdasarkan karakteristiknya yang diantaranya sebagai berikut:

* 1. PNG (*Portable Network Graphics*). Format ini dirancang sebagai format citra yang universal untuk digunakan di internet. PNG mendukung tiga jenis citra, yaitu *true color*, *grayscale*, dan citra terindeks. Selain itu, PNG juga dapat menyimpan nilai alpha untuk transparansi dengan maksimum kedalaman 16-bit. PNG bersifat *lossless compression* yang artinya hasil kompresi memungkinkan data asli untuk disusun kembali karena rasio kompresi tidak terlalu besar.
  2. JPEG (*Joint Photographic Expert Group*). JPEG digunakan untuk menyimpan citra berukuran *file* kecil namun memberi kualitas yang bagus. Format ini sangat baik untuk citra fotografi, namun kurang baik untuk citra non-fotografi. JPEG bersifat *lossy* *compression* yang artinya data hasil kompresi akan berbeda dengan data sebelumnya dikarenakan rasio kompresi yang bergitu besar.
  3. BMP (*Windows* *Bitmap*). BMP merupakan format yang sederhana sebagai *file* citra. Citra ini juga mendukung tiga jenis citra seperti PNG, yaitu *true color*, *grayscale*, dan terindeks. Citra ini biasa didapatkan sebagai file mentah hasil dari kamera digital dengan ukuran *file* yang besar. BMP bersifat *lossless* *compression*.
  4. TIFF (*Tagged Image File Format*). TIFF sering digunakan untuk menyimpan data citra tanpa kompresi. Ukuran file TIFF cukup besar namun memberikan kualitas yang maksimal. Format TIFF banyak digunakan untuk kebutuhan professional di berbagai bidang. TIFF bersifat *lossy compression*.

## Steganografi

Steganografi berasal dari bahasa Yunani yaitu kata *stegos* yang berarti sembunyi dan *graphia* yang berarti tulisan. Steganografi secara umum memiliki arti ilmu dan seni menyembunyikan suatu fakta untuk berkomunikasi. Dengan menggunakan steganografi, pesan rahasia dapat disisipkan ke dalam sebuah informasi yang tidak mencurigakan dan mengirimkannya tanpa ada yang mengetahui keberadaan dari pesan rahasia tersebut (Krenn, 2004). Dalam era digital ini, steganografi berarti penyisipan pesan dalam bentuk digital ke dalam media digital yang ada. Untuk steganografi terbentuk dari dua macam yaitu pesan digital atau sering disebut dengan *message* yang akan disisipkan dan media tempat penyisipan akan dilakukan. Media tempat penyisipan dapat berupa teks, gambar, suara, dan video. Untuk media gambar, gambar yang dijadikan sebagai citra penampung disebut dengan *cover* *image*. Penyembunyian pesan yang berupa teks maupun gambar ke dalam citra digital akan mempengaruhi kualitas citra tersebut.

### Least Significant Bit

Bit atau *binary digit* adalah unit dasar penyimpanan data di dalam komputer, nilai bit suatu data adalah 0 (nol) atau 1 (satu). Semua data yang ada pada komputer disimpan ke dalam satuan bit ini, termasuk gambar, suara, ataupun video hanya saja penerjemahan representasi bit pada masing-masing media yang tentunya akan berbeda. Seperti penjelasan pada bagian sebelumnya bahwa format pewarnaan pada media gambar juga menggunakan satuan bit dalam penyimpanannya. Sebagai contoh pewarnaan *monochrome* menggunakan 1-bit untuk merepresentasikan warna hitam atau putih, pewarnaan *grayscale* menggunakan 8-bit untuk merepresentasikan tingkat keabuan dan pada pewarnaan RGB menggunakan 24-bit (8-bit untuk *Red*, 8-bit untuk *Green*, 8-bit untuk *Blue*).

*Least Significant Bit* (LSB) adalah bagian dari barisan data biner yang mempunyai nilai paling tidak berarti atau paling kecil. Bit LSB letaknya di paling kanan pada barisan biner, karena nilai 1-bit LSB pada barisan biner hanya merepresentasikan nilai 1 desimal, maka bit ini dianggap tidak berarti. Sehingga jika terjadi perubahan pada nilai bit LSB maka tidak akan terjadi perubahan secara signifikan.



**Gambar 2.3.** Ilustrasi penyisipan bit LSB

*Modified Least Significant Bit* (MLSB) atau modifikasi dari Algoritma LSB digunakan untuk meng-*encode* sebuah identitas ke dalam citra asli. MLSB menggunakan manipulasi beberapa bit-bit penyisip sebelum meng-encode pesan tersebut (Zaher, 2011).

### Discrete Cosine Transform

*Discrete Cosine Transform* (DCT) Transformasi DCT merupakan salah satu *transform* *coding* yang akan merubah *byte* dari domain spasial menjadi domain frekuensi dan memisahkan *byte* data tersebut menjadi dua bagian, yaitu frekuensi tinggi (koefisien DC) dan frekuensi rendah (koefisien AC). Pada DCT, koefisien DC digunakan sebagai tempat penyisipan pesan. Hal ini dikarenakan koefisien DC memiliki kapasitas persepsi yang lebih tinggi dari pada koefisien AC sehingga proses penyisipan tidak akan mengubah kualitas gambar secara visual. Selain itu, sinyal proses dan distorsi gambar memiliki pengaruh yang lebih rendah terhadap koefisien DC daripada koefisien AC (Reva, Susilo, & Purwandari, 2016).

Transformasi citra dilakukan dengan menggunakan DCT (*Discrete Cosine Transform*), sehingga dapat dikatakan bahwa penyisipan dilakukan pada ranah DCT. Penyisipan dilakukan terhadap citra bitmap dengan kedalaman warna 24 bit. DCT digunakan untuk metransformasikan nilai intensitas blok 8x8 pikselnya yang berurutan dari *image* menjadi 64 koefisien DCT kedalam frekunesi dasarnya, diubah koefisien- koefisiennya dan kemudian ditransformasikan kembali dengan IDCT (*Inverse Discrete Cosine Transform*). Setiap basis matriks dikarakteristikan oleh frekuensi spatial horizontal dan vertikal. Dalam konteks citra, hal ini menunjukkan tingkat signifikasi secara perseptual, artinya basis fungsi dengan frekuensi rendah memiliki sumbangan yang lebih besar bagi perubahan penampakan citra dibandingkan basis fungsi yang memiliki frekuensi tinggi. Nilai konstanta basis fungsi yang terletak di bagian kiri atas sering disebut basis fungsi DC, dan DCT koefisien yang bersesuaian dengannya disebut koefisien DC (DC *coeficient*). Masukan proses DCT berupa matrik NxN. Persamaan DCT untuk matrik berukuran NxN dapat dituliskan sebagai berikut:

|  |  |  |
| --- | --- | --- |
|  | …….. | ( 2.2 ) |

Sedangkan persamaan untuk invers DCT (IDCT) adalah sebagai berikut:

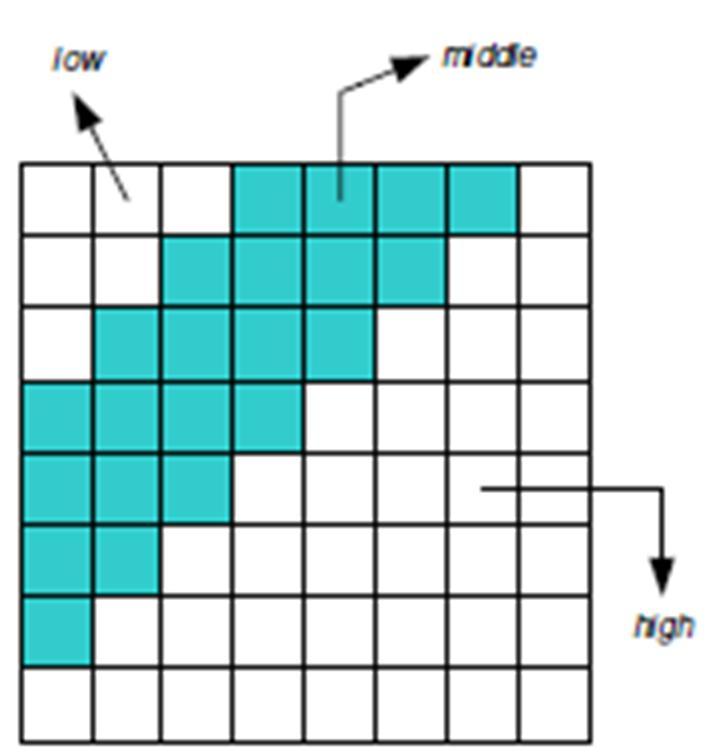
|  |  |  |
| --- | --- | --- |
|  | …….. | ( 2.3 ) |

Dimana:

T (i, j) = Data pada domain frekuensi

*pixel* (x, y) = Data pada domain ruang secara pixel

Output dari fungsi DCT adalah nilai komponen frekuensi tertentu dan output dari fungsi ini ditentukan oleh dua parameter, yaitu i dan j. Begitu pula pada proses IDCT, hanya saja proses tersebut untuk mengembalikan citra semula.



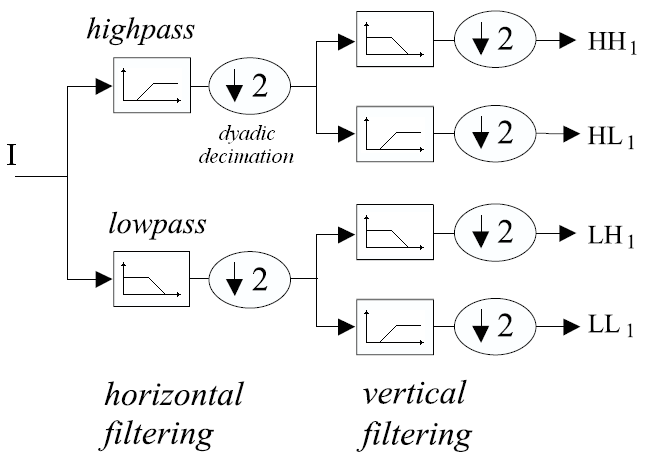
**Gambar 2.4** Ranah frekuensi pada koefisien DCT

Untuk mendapatkan koefisien DCT secara maksimal maka perlu dilakukan kuantisasi citra. Hal ini dilakukan untuk mendapatkan ranah frekuensi rendah, medium, dan frekuensi tinggi pada koefisien DCT. Persamaan matriks kuantisasi yang digunakan adalah sebagai berikut:

|  |  |  |
| --- | --- | --- |
|  | ………………… | ( 2.4 ) |

### Discrete Haar Wavelet Transform

Secara umum Transformasi Wavelet Diskrit merupakan dekomposisi citra pada frekuensi subband citra tersebut. Komponen subband transformasi wavelet dihasilkan dengan cara penurunan level dekomposisi. Implementasi Transformasi Wavelet Diskrit dapat dilakukan dengan melewatkan sinyal melalui sebuah *lowpass* *filter* dan *highpass* filter dan melakukan *downsampling* pada keluaran masing-masing filter. *Highpass* filter digunakan untuk menganalisis frekuensi tinggi dan *lowpass* *filter* digunakan untuk menganalisis frekuensi rendah (Reva, Susilo, & Purwandari, 2016). Proses dekomposisi dapat melalui satu atau lebih tingkatan.



**Gambar 2.5** Dekomposisi Wavelet satu tingkat

Hasil dekomposisi menghasilkan 4 koefisien yaitu LL, LH, HL, dan HH. Berikut adalah 4 *subband* yang dihasilkan pada proses dekomposisi dengan *Haar* *Wavelete*.

* + - 1. LL diperoleh melalui proses *Low* *pass* dan dilanjutkan dengan *Low* *pass*. Hasil dari *subband* ini hampir mirip dengan aslinya. Set koefesiennya adalah:

|  |  |
| --- | --- |
| …...……………………… | ( 2.5 ) |

* + - 1. LH merupakan koefisien yang didapatkan dari proses low pass dan dilanjutkan *High* *pass*. Set koefesiennya adalah:

|  |  |
| --- | --- |
| …...……………………… | ( 2.6 ) |

* + - 1. HL merupakan koefisien kebalikan dari LH, yaitu hasil dari *High* *pass* dan dilanjutkan dengan *Low* *pass*. Set koefesiennya adalah:

|  |  |
| --- | --- |
| …...……………………… | ( 2.7 ) |

* + - 1. HH merupakan hasil dari *High* *pass* dan dilanjutkan dengan *High* *pass* lagi. Set koefesiennya adalah:

|  |  |
| --- | --- |
| …...……………………… | ( 2.8 ) |

Dari uraian tersebut, WLL mewakili koefisien perkiraan, WLH mewakili koefisien perincian horizontal, WHL merupakan koefisien perincian vertical, dan WHH merupakan koefisien detail diagonal. J menunjukkan tingkat maksimum dekomposisi yang dilakukan.

Untuk merekonstruksi ulang menjadi ke keadaan semula, keempat *subband* koefisien tersebut digunakan untuk melakukan *Inverse Discrete Wavelete Transform* (IDWT) yang merupakan kebalikan dari proses DWT. Hasil yang didapatkan dari proses tersebut adalah sinyal asal sebelum dilakukan DWT.

### Pengujian Kualitas Metode Steganografi

Terdapat kriteria-kriteria yang harus diperhatikan dalam penyembunyian data yaitu:

1. *Impercebility* yaitu keberadaan pesan rahasia tidak dapat dipersepsi secara indrawi.

2. *Fidelity* yaitu mutu citra penampung tidak jauh berubah

3. *Robustness* yaitu data yang disembunyikan harus tahan terhadap manipulasi yang dilakukan pada citra penampung

4. *Recovery* yaitu pesan dapat diekstrak kembali.

#### Uji Tingkat Robustness

*Robustness* merupakan salah satu kriteria dalam steganografi yang kurang penting dalam steganografi dikarenakan selama objek stego tidak terdapat perbedaan mencolok dan tidak menimbulkan kecurigaan. Namun kriteria ini tetap dapat dibilang penting apabila menyangkut objek tersembunyi di dalamnya. *Robustness* tidak kalah penting dikarenakan jika terdapat serangan pada objek stego, *hidden object* harus tetap tidak rusak dan dapat memenuhi kriteria *Recovery*.

Dalam pengujiannya, kriteria *robustness* dapat dilakukan dengan beragam cara yang juga disebut sebagai *steganalysis*. Salah satu metode yang dapat diterapkan adalah metode serangan StirMark. StirMark merupakan metode *steganalysis* yang menggabungkan berbagai teknik dasar dalam manipulasi citra seperti rotasi, pemotongan, *resampling*, *resizing*, dan kompresi pada citra *watermark*. Pada umumnya skema *watermarking* berhasil lolos dari manipulasi-manipulasi tersebut dan pada umumnya pula, skema watermark tidak dapat bertahan dari gabungan atau kombinasi dari manipulasi-manipulasi dasar tersebut. Hal ini menjadi dasar pembangunan dari metode StirMark. (Ferdian, 2006)

Dapat dikatakan dalam versi yang paling sederhana, StirMark membangkitkan sebuah proses *resampling*. Proses *resampling* ini dilakukan secara digital. Proses ini dapat diumpamakan dengan cara mencetak sebuah citra ke kertas dan melakukan scanning terhadap citra tersebut, dalam hal ini scanner yang digunakan diasumsikan berkualitas sangat tinggi, sehingga hampir tidak ada perubahan jika hasil resampling dibandingkan dengan sample aslinya.

#### Uji Tingkat Fidelity

*Fidelity* merupakan kriteria paling umum yang sangat penting dalam steganografi. Kriteria *fidelity* menunjukan besaran mutu stego object tidak jauh berubah dari sebelum dimasukkan *hidden* *object* maupun sesudahnya. Hal ini berkaitan tentang kecurigaan pihak lain jika melihat *stego* *object* dan demi keamanan objek yang disembunyikan maka kriteria *fidelity* perlu diperhatikan. Jika objek stego mempunyai nilai *fidelity* yang sangat rendah, kriteria lain seperti *impercebility* juga akan terpengaruh dikarenakan distorsi yang ada tentunya akan disadari oleh inderawi.

Pengujian fidelity umumnya menggunakan penghitungan PSNR dan MSE. *Peak Signal to Noise Ratio* (PSNR) merupakan sebuah parameter yang penting untuk mengukur kualitas proses pengolahan citra. PSNR adalah rasio antara intensitas maksimum citra dengan *Mean Square Error* (MSE) dari citra. Persamaan untuk menghitung nilai PSNR adalah sebgai berikut:

|  |  |
| --- | --- |
| …………………………………………………….... | ( 2.9 ) |

Dengan persamaan MSE adalah sebagai berikut:

|  |  |
| --- | --- |
| ………………………………........... | ( 2.10 ) |

Dalam penghitungan kualitas dua buah citra makin besar nilai PSNR maka makin kecil nilai MSE. Nilai MSE yang kecil mengindikasikan dua buah citra mempunyai sedikit perbedaan. Dari pengalaman empiris, citra dengan PSNR≥30 masih dapat dianggap berkualitas bagus, tetpi jika PSNR < 30 dikatakan kualitas citra terdegradasi secara signifikan (Munir, 2019).

## Prototyping Software Development Life Cycle

Menururt Purnomo (2017)*, Prototyping* merupakan metode pengembangan perangat lunak, yang berupa model fisik kerja sistem dan berfungsi sebagai versi awal dari sistem. Dengan metode *prototyping* ini akan dihasilkan *prototype* sistem sebagai perantara pengembang dan pengguna agar dapat berinteraksi dalam proses kegiatan pengembangan sistem informasi. Agar proses pembuatan *prototype* ini berhasil dengan baik adalah dengan mendefinisikan aturan-aturan pada tahap awal, yaitu pengembang dan penguna harus satu pemahaman bahwa *prototype* dibangun untuk mendefinisikan kebutuhan awal. *Prototype* akan dihilangkan atau ditambahkan pada bagiannya sehingga sesuai dengan perencanaan dan analisis yang dilakukan oleh pengembang sampai dengan uji coba dilakukan secara simultan seiiring dengan proses pengembangan.

Tahapan dalam metode *prototype* menurut pengembang metode *prototype* itu sendiri yaitu Pressman (2010) dapat dilihat pada gambar 2.6.



**Gambar 2.6** Metode Prototyping (Pressman, 2010)

Metode *prototype* cocok digunakan untuk mengembangkan sebuah perangkat yang akan dikembangkan kembali. Metode ini dimulai dengan pengumpulan kebutuhan pengguna. Kemudian membuat sebuah rancangan kilat yang selanjutnya akan dievaluasi kembali sebelum diproduksi secara besar.

*Prototype* bukanlah merupakan sesuatu yang lengkap, tetapi sesuatu yang harus dievaluasi dan dimodifikasi kembali. Segala perubahan dapat terjadi pada saat *prototype* dibuat untuk memenuhi kebutuhan pengguna dan pada saat yang sama memungkinkan pengembang untuk lebih memahami kebutuhan pengguna secara lebih baik.

Berdasarkan model *prototype* yang telah digambarkan diatas, maka dapat diuraikan pembahasan masing-masing tahap dalam model tersebut adalah sebagai berikut:

1. *Communication*/Komunikasi

Perancang perangkat lunak melakukan analisis untuk menentukan kebutuhan perangkat lunak yang saat itu diketahui dan untuk menggambarkan area-area dimana definisi lebih jauh untuk iterasi selanjutnya.

1. *Quick Plan and Modeling Quick Design*/Pemodelan Rancangan Cepat

Pada bagian ini akan membuat perencanaan dan pemodelan secara cepat berdasarkan kebutuhan yang telah diidentifikasi pada tahap sebelumnya. Perencanaan yang akan dilakukan dapat berupa pembagian tugas dan rencana kerja. Sedangkan dari sisi pemodelan akan membuat desain model yang merepresentasikan aspek-aspek yang diinginkan dari pelanggan.

1. *Construction of Prototype*/Pembuatan Prototipe

Dalam pembuatan rancangan cepat berdasarkan pada representasi aspek-aspek perangkat lunak yang akan terlihat oleh *end user* (tampilan antarmuka). Rancangan cepat merupakan dasar untuk memulai konstruksi pembuatan prototipe.

1. *Deployment Delivery and Feedback*/Penyerahan dan Pemberian Umpan Balik

Prototipe kemudian dipresentasikan untuk mengevaluasi prototipe yang telah dibuat sebelumnya dan memberikan umpan balik yang akan digunakan untuk memperbaiki spesifikasi kebutuhan. Iterasi akan terjadi saat pengembang melakukan perbaikan prototipe tersebut.

## Bahasa Pemrograman Python

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python juga didukung oleh komunitas yang besar.

Python mendukung multi paradigma pemrograman, utamanya pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, python umumnya digunakan sebagai bahasa skrip meski pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa skrip. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi.

Python dikembangkan oleh Guido van Rossum pada tahun 1990 di *Stichting Mathematisch Centrum* (CWI), Amsterdam sebagai kelanjutan dari bahasa pemrograman ABC. Versi terakhir yang dikeluarkan CWI adalah 1.2. (Venners, 2003)

Tahun 1995, Guido pindah ke CNRI di Virginia Amerika sambil terus melanjutkan pengembangan Python. Versi terakhir yang dikeluarkan adalah 1.6. Tahun 2000, Guido dan para pengembang inti Python pindah ke BeOpen.com yang merupakan sebuah perusahaan komersial dan membentuk BeOpen PythonLabs. Python 2.0 dikeluarkan oleh BeOpen. Setelah mengeluarkan Python 2.0, Guido dan beberapa anggota tim PythonLabs pindah ke *Digital Creations*.

Saat ini pengembangan Python terus dilakukan oleh sekumpulan pemrogram yang dikoordinir Guido dan *Python Software Foundation*. *Python Software Foundation* adalah sebuah organisasi non-profit yang dibentuk sebagai pemegang hak cipta intelektual Python sejak versi 2.1 dan dengan demikian mencegah Python dimiliki oleh perusahaan komersial. Saat ini distribusi Python sudah mencapai versi 2.7.14 dan versi 3.6.3

## Tinjauan Literatur

Adapun beberapa penelitian yang telah dilakukan dan dijadikan referensi dalam penelitian berikut ini:

**Tabel 2.1.** Tinjauan Literatur

| **No** | **Penulis** | **Judul** | **Metode** | **Hasil** |
| --- | --- | --- | --- | --- |
| 1 | Chyquita Danuputri, M. Kom. 2018 | Pengamanan Data melalui *Cloud Computing* dengan Integrasi Steganografi *Least Significant Bit* dan Kriptografi *Vigenere Key* Berbasis *Android* | *Standart Least Significant Bit* dan enkripsi *Vigenere Key* | Pesan berhasil di sembunyikan pada citra digital dan diimplementasikan pada *cloud drive*. |
| 2 | Rahmandhita Fikri Sannawira dan Agus sidiq Purnomo. 2016 | Penyisipan Citra Pesan ke dalam Citra Berwarna menggunakan Metode *Least Significant Bit* dan *Redundant Pattern Encoding* | *Standart Least Significant Bit Redundant* dan *Pattern Encoding* | Penghitungan kualitas dengan metode PSNR mendapatkan hasil yang maksimal dan tetap memenuhi syarat dianggap bagus sebagai *stego* *object*. Namun manipulasi *stego* *object* akan berpengaruh terhadap *hidden object.* |

**Tabel 2.2.** Lanjutan Tinjauan Literatur

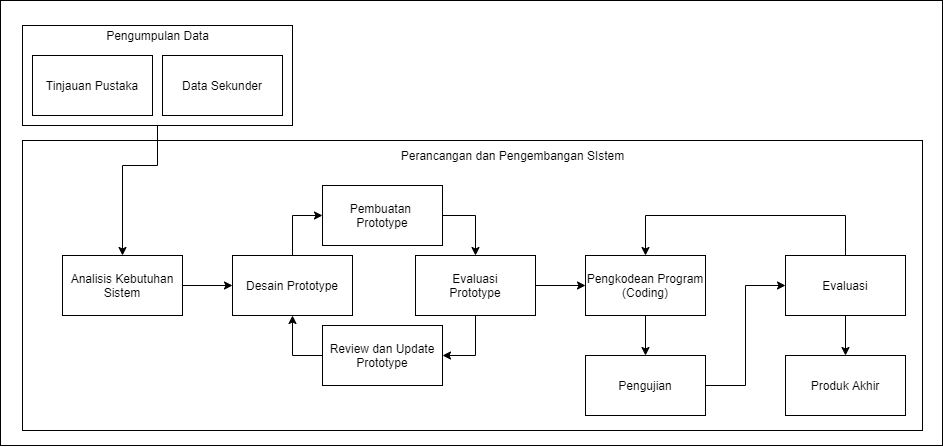
| 3 | Garno dan Arip Solehudin.  2017 | Teknik Steganografi dengan metode *Discrete Cosines Transform* (DCT) pada Citra Interpolasi *Bilinear* untuk Pengamanan Pesan | *Discrete Cosines Transform* danInterpolasi *Bilinear* | Dari penelitian yang telah dilakukan menghasilkan *stego* *object* dengan rata-rata belum mencapai nilai 40 db sehingga terbilang kurang bagus namun memiliki nilai MSE yang relatif kecil. |
| --- | --- | --- | --- | --- |
| 4 | Swapnali Zagade, Smita Bhosale. 2014 | *Secret Data Hiding in Images by using DWT Technique’s* | *Discrete Haar Wavelete Transform* | Dari penelitian yang telah dilakukan menghasilkan *stego* *object* dengan hasil perbandingan PSNR yang tinggi. |
| 5 | Aditya Mahmud Faza, Cepy Slamet, Dian Nursantika. 2016 | Analisis Kinerja Kompresi Citra Digital dengan Komparasi DCT, DWT, dan Hybrid (DCT-DWT) | *Discrete Cosines Transform* dan *Discrete Haar Wavelete Transform* | Peneliatian ini bukan mengenai steganografi namun menggunakan beberapa metode stego ranah frekuensi. Hasil yang didapat tergolong bagus. Namun dari ketiga metode yang diuji hasil DWT jauh lebih unggul. |

Penelitian ini akan menganalisis metode LSB, DCT, dan DWT dalam steganografi ditinjau dari faktor *robustness* dan *fidelity*. Hal yang membedakan dari penelitian sebelumnya adalah objek yang diteliti yang kebanyakan lebih fokus pada performa algoritma berdasarkan waktu eksekusi dan perbandingan visual hasil akhir.

# BAB III METODOLOGI PENELITIAN DAN PENGEMBANGAN SISTEM

## Metodologi Penelitian

Metode penelitian berisi langkah-langkah yang digunakan dalam penelitian ini agar terstruktur dengan baik. Dengan sistematika ini proses penelitian dapat dipahami dan diikuti oleh pihak lain. Penelitian yang dilakukan untuk merancang sistem diperoleh dari pengamatan data-data yang ada. Metode penelititan yang digunakan adalah metode kuantitatif karena penelitian akan melakukan serangkaian pengujian sebagai analisa data sampel yang akan digunakan sebagai dasar penarikan kesimpulan. Adapun langkah-langkah yang tujuan dari penelitian ini dapat dilihat pada gambar 3.1.



**Gambar 3.1** Alur Tahap Penelitian

### Identifikasi Masalah

Tahap awal dalam penelitian ini adalah mengidentifikasi masalah yang akan dijadikan sebagai objek penelitian. Objek dari penelitian ini adalah citra digital yang merupakan salah satu informasi penting di era digital seperti ini. Permasalahan keamanan informasi yang terdapat pada citra digital adalah mengenai hak kepemilikan Citra digital. *Watermarking* merupakan salah satu teknik yang dapat digunakan untuk menyelesaikan permasalahan tersebut. *Watermarking* sendiri merupakan salah satu bagian dari implementasi seni menyembunyikan objek atau biasa disebut steganografi. Steganografi mempunyai banyak metode yang bisa digunakan. Dari metode-metode tersebut terdapat kelebihan dan kekurannya masing-masing.

## Metode Pengembangan Sistem

Metode pengembangan sistem yang digunakan dalam penelitian ini adalah metode *Prototype* yang telah dikembangkan oleh Pressman. Metode *prototyping* menggunakan pendekatan pengembangan system dengan empat tahap utama yaitu komunikasi, *quick* *design*, pembuatan *prototype*, dan *deployment delivery and feedback* (umpan balik kepada pengguna).

Metode *prototype* dimulai dari mengumpulkan segala kebutuhan terhadap perangkat lunak yang akan dibuat. Lalu dibuatlah purwarupa agar sudah terdaapt bayangan tentang perangkat lunak yang akan dibuat. Purwarupa biasanya merupakan program yang belum jadi. Program ini biasanya menyediakan tampilan dan simulasi alur perangkat lunak sehingga tampak seperti perangkat lunak yang sudah jadi. Program *prototype* tersebut selanjutnya dievaluasi sampai ditemukan spesifikasi yang sesuai dengan yang diharapkan (Sukamto & Shalahuddin, 2018).

### Komunikasi

Pada tahap komunikasi, didefinisikan format seluruh perangkat lunak, identifikasi kebutuhan, dan garis besar aplikasi steganografi yang akan dibangun. Tahap ini dibagi menjadi tiga bagian yaitu identifikasi masalah, studi pustaka dan pengumpulan data, dan analisis kebutuhan sistem. Penjelasan tentang identifikasi masalah telah dijelaskan pada sub bab 3.1 tentang metode penelitian.

#### Studi Pustaka

Studi pustaka dilakukan untuk mempelajari teori-teori dan hasil penelitian sebelumnya yang berkaitan dengan masalah pada penelitian ini. Studi pustaka bertujuan untuk memperkuat materi pembahasan maupun rumus-rumus tertentu dalam menganalisa dan mendesain suatu sistem. Sumber-sumber yang di resensi seperti buku-buku, jurnal penelitian, artikel, thesis atau diktat perkuliahan. Tapi ini dilakukan untuk pencarian informasi mengenai steganografi, metode yang digunakan dalam steganografi, dan informasi lainnya untuk mendukung pengembangan aplikasi yang akan dibuat.

#### Pengumpulan Data

Pengumpulan data merupakan proses untuk mengetahui lebih dalam mengenai metode yang akan digunakan untuk dianalisis beserta metode pengujian yang akan dilakukan pada *output* metode-metode tersebut. Tujuan dari pengumpulan data tersebut untuk mendapatkan informasi mengenai hal-hal yang dibutuhkan untuk mendukung pembuatan sebuah sistem. Pengumpulan data pada aplikasi ini dilakukan dengan studi pustaka untuk mengetahui pengembangan aplikasi yang digunakan untuk teknik steganografidan gambar yang digunakan untuk proses penyembunyian objek dalam penelitian ini, serta penggunaan data sekunder berupa citra yang didapatkan dari internet dengan format ekstensi *file* yang sudah ditentukan.

#### Analisis Kebutuhan Sistem

Analisis kebutuhan sistem pada penelitian ini mencakup analisis kebutuhan fungsional dan non-fungsional. Kebutuhan fungsional merupakan kebutuhan yang berisikan proses-proses yang mampu dilakukan oleh aplikasi. Sedangkan kebutuhan non-fungsional merupakan kebutuhan yang menitikberatkan pada properti perilaku yang dimiliki oleh aplikasi. Kebutuhan fungsional dalam pengembangan aplikasi ini diantaranya sebagai berikut:

User dapat memasukkan *cover image* dan *hidden object*.

User dapat melihat *stego image* dan hasil ekstraksi dari *stego image*.

User dapat melihat hasil PSNR dan MSE baik dari *cover image* dan *stego image* maupun *stego image original* dengan *stego image* yang telah diserang.

Kebutuhan non-fungsional melibatkan beberapa perangkat yang mendukung aplikasi. Perangkat tersebut berupa perangkat keras (*hardware*) dan perangkat lunak (*software*). Kebutuhan perangkat tersebut diantaranya sebagai berikut:

Analisis Kebutuhan Perangkat Keras

Perangkat keras (*hardware*) yang dibutuhkan untuk pengembangan aplikasi ini adalah Laptop Asus X456UR yang memiliki spesifikasi sebagai berikut:

*Processor* Intel® Core™ i5 6200U *Processor* @2.30 GHz

*RAM* 8 GB

*Internal Drive* 1 TB HDD+ 512 GBSSD

*Display* 13.3" (1920x1080)

*Graphic* NVIDIA® GeForce® 930MX dengan 2GB DDR3 VRAM

Analisis Kebutuhan Perangkat Lunak

Perangkat lunak (*software*) yang dibutuhkan untuk pengembangan aplikasi ini diantaranya sebagai berikut:

OS Microsoft Windows 10 Pro

Python v3.8.1

OpenCV v4.2.0.32

Numpy v1.18.1

PyWavelets v1.1.1

Flask v1.1.1

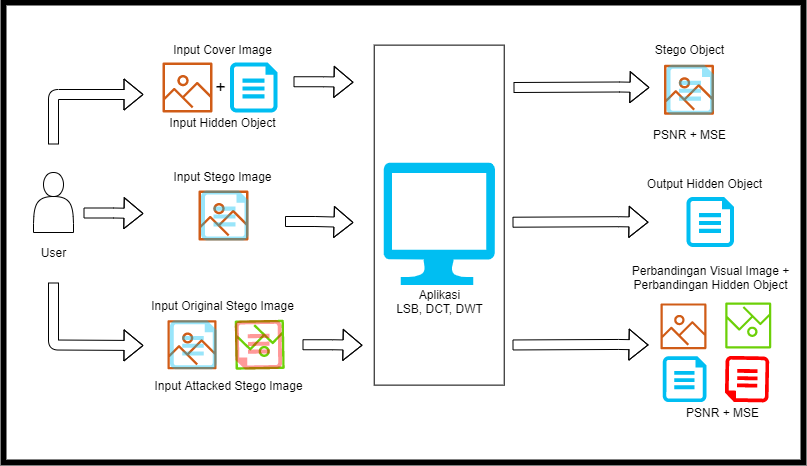
Jinja2 v2.11.1

### Quick Design

Pada tahap *quick* *design*, akan dibuat desain aplikasi steganografi secara umum untuk dikembangkan kembali. Tahap ini dibagai menjadi empat yaitu perancangan arsitektur, perancangan proses, perancangan antarmuka, dan perancangan pengujian.

#### Perancangan Arsitektur

Perancangan asitektur merupakan gambaran proses dari sistem yang akan dikembangkan. Arsitektur dari sistem yang akan dibangun dapat dilihat pada gambar 3.2.

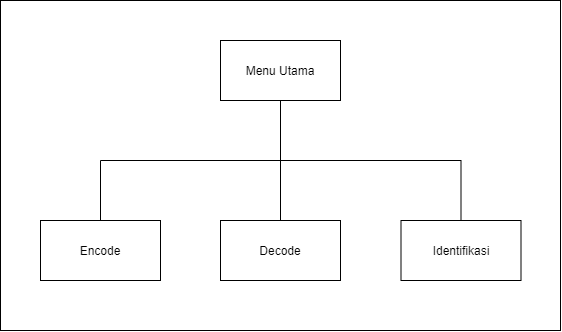


**Gambar 3.2** Perancangan Arsitektur Aplikasi

Pada arsitektur di atas, *user* akan memasukkan memiliki tiga pilihan yang dapat dilakukan pada aplikasi. Utamanya user akan memasukkan citra yang akan dijadikan citra *cover* dan *hidden object* dengan prosedur *encode*. Kemudian aplikasi akan memproses citra digital tersebut menjadi empat buah citra stego yang merupakan hasil dari penyisipan pesan pada citra menggunakan metode LSB, DCT, DWT, dan kombinasi ketiga metode tersebut. Selanjutnya *user* juga dapat melakukan ektraksi pesan pada menu *decode*. Proses ini menghasilkan *hidden* *object* yang didapat dari ekstraksi pesan oleh *stego* *object* dengan metode yang telah dipilih. Terakhir *user* juga dapat membandingkan dua buah *stego* *object* yang mana merupakan satu buah *stego* *object* asli dan *stego* *object* yang telah diserang. Proses ini terdapat pada menu identifikasi dan akan menghasilkan komparasi kedua citra tersebut secara visual, nilai PSNR antara kedua citra, dan ketahanan pesan yang terdapat dalam kedua citra.

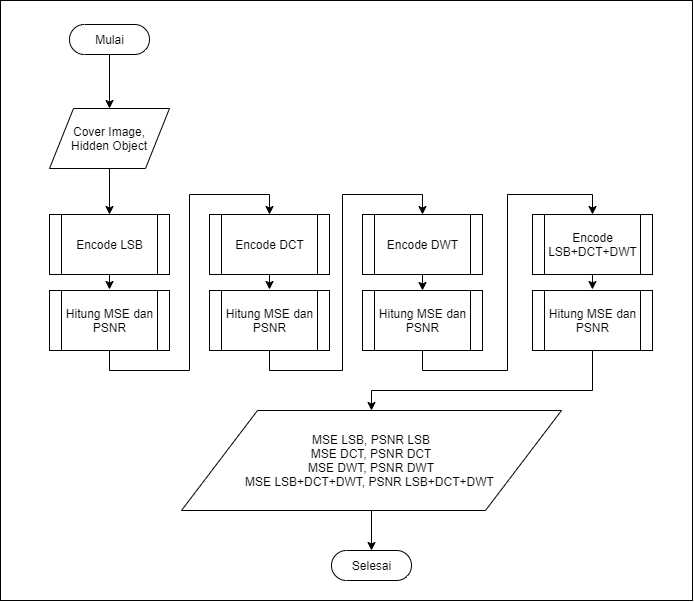
#### Perancangan Proses

Tahapan ini terdiri dari *flowchart* dari aplikasi yang akan dikembangkan yang terdiri dari *flowchart* proses steganografidan *flowchart* sub proses yang menggambarkan langkah-langkah pada steganograficitra digital dengan metode LSB, DCT, dan DWT.



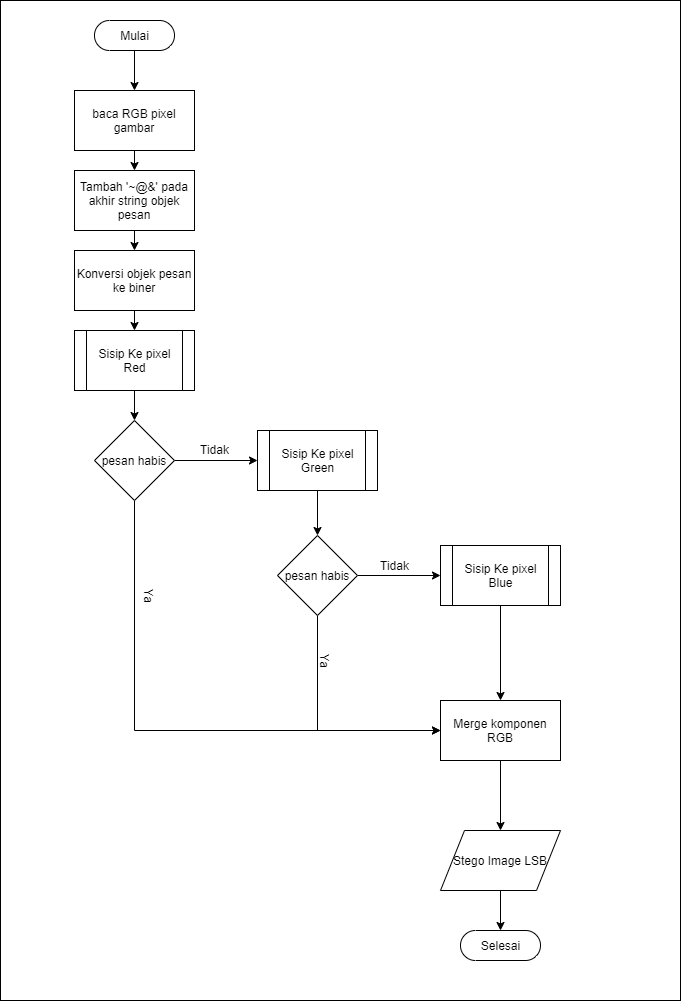
**Gambar 3.3** Struktur Menu Utama aplikasi

Pada gambar 3.3 menunjukan aplikasi yang akan dibuat memiliki tiga pilihan proses utama. Pilihan pertama berupa menu *encode*. Pada pilihan *encode* ini *user* akan memasukkan gambar *cover* dan objek *hidden* lalu akan melakukan proses penyisipan objek dengan semua metode yang ada. Proses ini menghasilkan empat citra berbeda yang meliputi citra yang telah disisipi dengan metode LSB, DCT, dan DWT, serta kombinasi ketiga metode tersebut. Pada pilihan kedua terdapat proses *decode* atau ekstraksi objek dari objek stego. Pada pilihan ketiga adalah proses identifikasi dimana *user* akan memasukkan citra stego asli dan citra stego yang sudah dimodifikasi/serang untuk dibandingkan. Pada proses ini *output* yang dihasilkan berupa komparasi kedua citra tersebut secara visual, ekstraksi objek *hidden*, dan nilai PSNR.



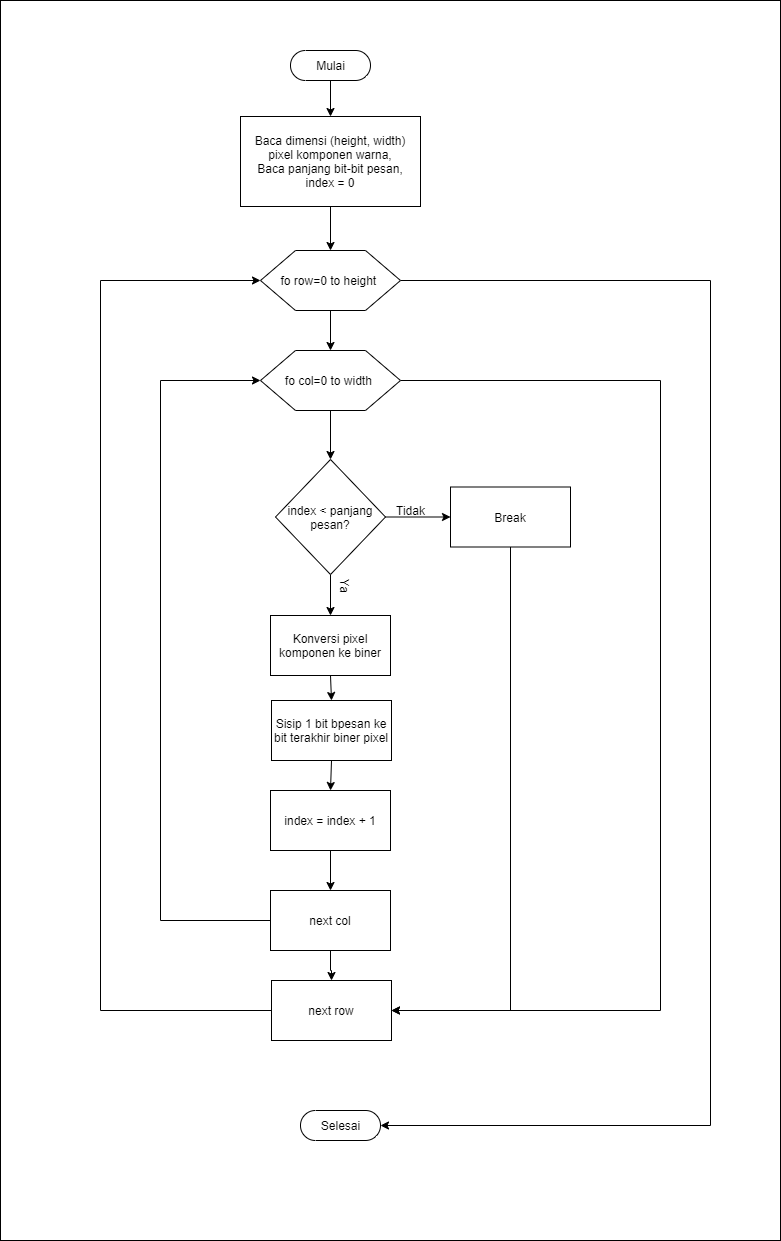
**Gambar 3.4** Flowchart Proses Encode

Pada gambar 3.4 ditunjukkan proses *encode* atau penyematan *hidden* *object* kedalam citra *cover*. Proses tersebut melibatkan sub proses *encode* LSB, DCT, dan DWT. Hasil dari sub proses tersebut merupakan sebuah citra stego dari masing-masing metode yang akan diidentiikasi kedepannya serta nilai PSNR dan MSE yang akan dijadikan acuan dalam menganalisis *fidelity* citra stego tersebut.



**Gambar 3.5** Flowchart subroutine encode LSB

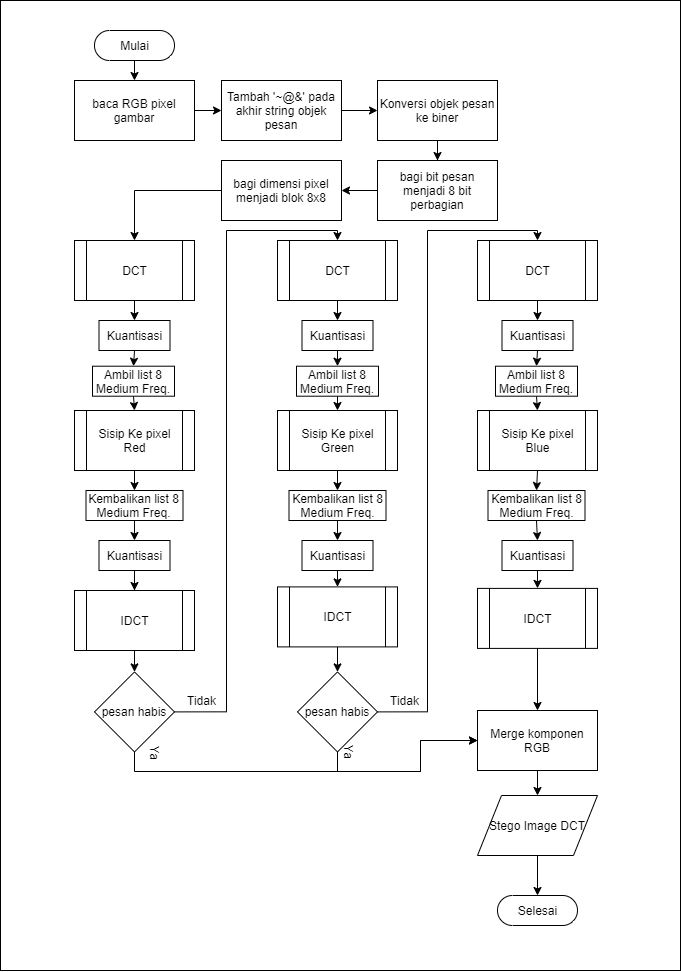
Pada *subroutine* *encode* LSB yang ditunjukkan gambar 3.5, citra *cover* akan dibaca panjang dan lebarnya dan akan dijadikan acuan ukuran untuk batas memasukkan *hidden* *object*. *Hidden* *object* berupa teks akan langsung ditambahkan tanda stop berupa ‘~@&’ yang sekiranya 3 karakter tersbut jarang dikombinasikan dalam kehidupan. Sementara *hidden object* berupa citra akan diubah menjadi *string* dengan *encode* base64() barulah ditambahkan tanda stop. Hal ini dilakukan agar penyematan juga berupa *string*. Pada pilihan *hidden* *object* teks merupakan representasi objek yang pendek dan tidak padat dan *hidden object* citra yang dijadikan string merupakan representasi objek yang panjang dan padat. Kemudian *hidden* *object* yang akan disematkan akan diubah menjadi bit-bit biner. Bit-bit biner tadi akan digunakan untuk penyisipan pada *pixel* R, G, dan B. Komponen-komponen tersebut didapatkan dengan fungsi split() dari *library* Open-CV pada python. Metode penyematan yang digunakan menggunakan keseluruhan tiga komponen warna pada citra. Ketika pesan yang sudah dikonversi menjadi biner belum habis disematkan pada komponen warna *red*, maka akan dilanjutkan penyematan pada komponen warna *green*. Begitu pula seterusnya pada komponen warna *blue*. Pada saat proses penyisipan, sesuai dengan namanya, bit yang diubah merupakan bit terakhir dari 8-bit nilai warna dari *pixel* tersebut seperti yang ditunjukan pada *flowchart* gambar 3.6.



**Gambar 3.6** Flowchart sub proses penyisipan pada pixel komponen warna

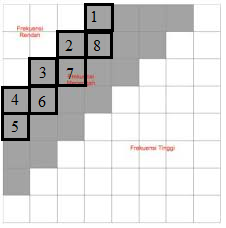
Setelah proses penyisipan selesai maka komponen R, G, dan B akan ditata ulang dengan fungsi merge() dari *library* Open-CV. Hal ini bertujuan menggabungkan komponen RGB menjadi satu kesatuan citra yang utuh. Hal ini juga akan dilakukan pada proses penyisipan dengan metode yang selain LSB.

Pada penyisipan pesan menggunakan metode DCT, proses penyisipan hampir sama dengan LSB dikarenakan steganografi ranah frekuensi tetap membutuhkan proses penyisipan ke bit lokasi pixel dengan penyisipan ranah spasial seperti LSB. Hal tersebut dapat dilihat pada gambar 3.7.

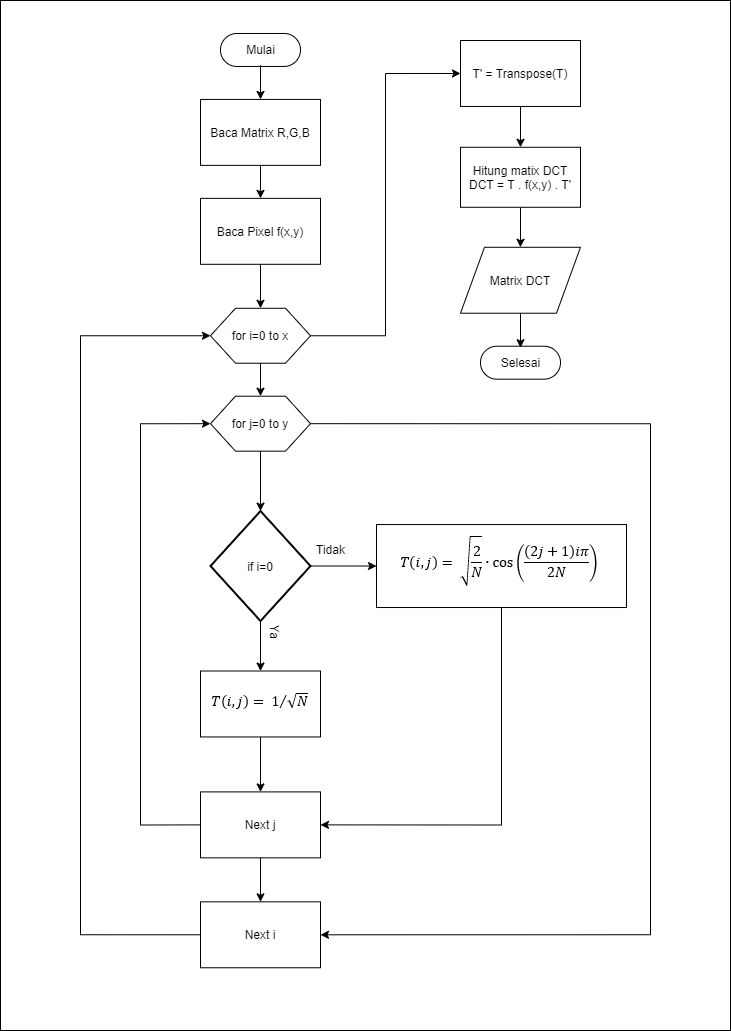


**Gambar 3.7** Flowchart sub proses encode DCT

Sebelum dilakukan penyisipan, bit-bit pesan dibagai menjadi beberapa bagian yang terdiri dari 64-bit perbagiannya. Hal ini dikarenakan citra akan dibagi menjadi blok 8x8 sehingga *pixel*-*pixel* yang akan diubah dengan DCT adalah berukuran 8x8 dari masing-masing blok pada citra. Proses DCT akan menggunakan *library* dari python yaitu *library* OpenCV dengan fungsi dct(). Fungsi tersebut menggunakan tipe DCT sesuai dimensi matriks array yang dimasukkan. Karena matriks yang dimasukkan berbentuk dua dimensi maka selanjutnya matriks hasil DCT akan dikuantisasi menggunakan matriks kuantisasi pada persamaan 2.4 untuk mendapatkan ketiga ranah frekuensi rendah, medium, dan tinggi. Pada proses DCT, kuantisasi yang dilakukan adalah membagi matriks DCT dengan matriks quant secara *element-wise* (operasi setiap per elemen). Selanjutnya penyisipan akan menggunakan ranah medium untuk menyisipkan pesan. Ranah frekuensi medium digunakan karena tidak terlalu bersifat *noise* bagi citra sehingga secara visual citra tidak terlalu berubah dari aslinya. Pada bagian tersebut akan diambil delapan tempat pertama untuk disisipi secara *zig-zag* seperti pada gambar 3.8. Delapan tempat digunakan agar koefisien tidak terlalu berubah secara signifikan. Karena ukuran tersebut maka pada masing-masing bagian menyisipkan 1-bit pada akhir 8-bit mereka.



**Gambar 3.8** Visualisasi tempat penyisipan bit pada koefisien DCT



**Gambar 3.9** Flowchart sub proses DCT

Metode DCT menggunakan pendekatan nilai kosinus yang akan mengubah detail warna dari gambar. Namun karena keterbatasan indra penglihatan manusia, maka perubahan yang terjadi tidak begitu terlihat. Pada gambar 3.9 ditunjukkan proses yang dijalankan berdasarkan prosedur DCT tipe 2 untuk objek dua dimensi. Proses tersebut dimulai dengan membaca nilai RGB dengan *library* Open-CV dengan fungsi split() yang akan memisahkan matriks R, G dan B. Kemudian pada setiap matriks akan dibaca posisinya mulai dari koordinat awal sampai akhir dan menjalankan algoritma yang telah ditentukan dalam metode DCT. Untuk contoh perhitungan manual akan dijeaskan sebagai berikut:

1. Mengambil *pixel* dari komponen R/G/B citra. Sebagai contoh perhitungan ini akan menggunakan ukuran 4x4 *pixel*. Diibaratkan matriks pixel(x,y) = matriks A.

A =

1. Mencari matriks T dengan persamaan 2.2. Berdasarkan matriks A, diketahui bahwa:

N = 4

Representasi posisi pixel (x, y) = (i, j)

Jika x sama dengan 0, maka

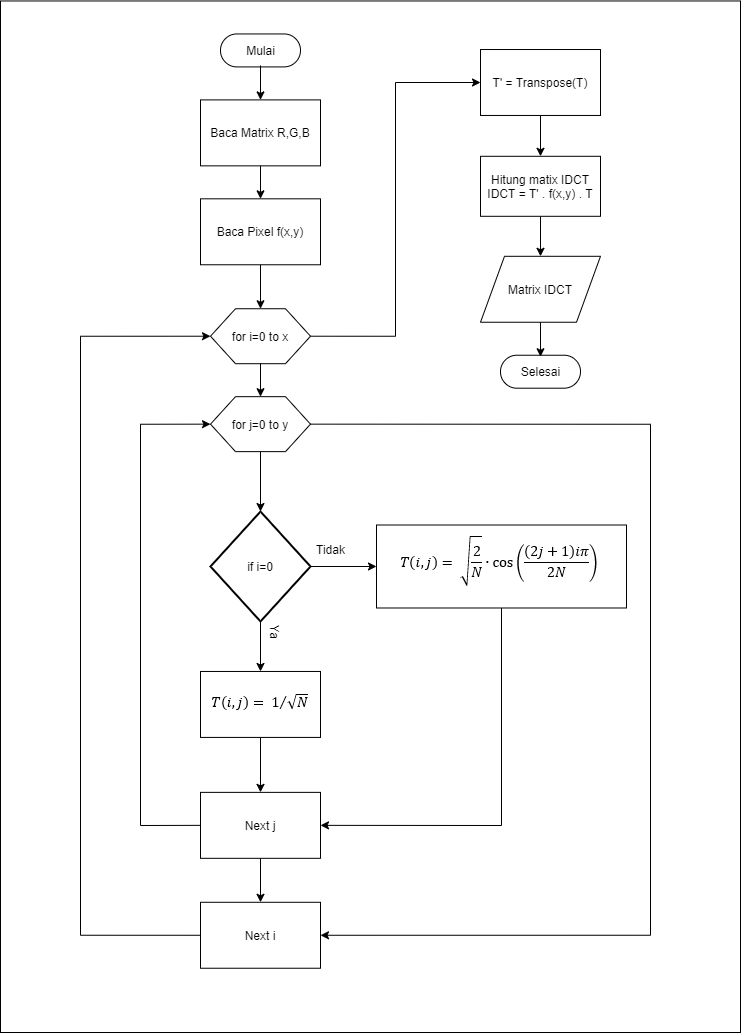
Jika x tidak sama dengan 0, maka

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Sehingga, nilai matriks T secara keseluruhan adalah:

1. Menghitung matriks akhir DCT dengan mengalikan matrik T, matriks A, dan matriks *Transpose* T secara berurutan.
2. Mengubah setiap elemen matriks DCT sehingga bertipe data *integer* agar dapat dikonversi menjadi biner.

Setelah mendapat matriks DCT dengan tipe data *integer*, maka selanjutnya elemen-elemen matriks tersebut dapat disematkan pesan melalui metode LSB. Elemen pada matriks harus bersifat *integer* sedangkan hasil dari perhitungan DCT bertipe data *float*. Maka dari itu harus diubah terlebih dahulu tipe data dari elemen matriks karena untuk konversi biner menggunakan fungsi *integer* dari python yang diwajibkan nilai bersifat *integer* dan bukan *float*. Proses penyisipan menyesuaikan metode yang digunakan. Karena pada DCT dimensi *pixel* dibagi menjadi blok 8x8 maka pesan yang disisipkan pada blok tersebut merupakan 8-bit atau satu karakter pada masing-masing bagian. Hal tersebut berlanjut sampai pesan habis disisipkan. Setelah pesan disisipkan pada matriks DCT tersebut, maka matriks tersebut perlu dikembalikan lagi ke keadaan awal melalui proses IDCT. Proses IDCT (*Inverse Discrete Cosine Transform*) merupakan proses untuk membalikkan matriks hasil DCT ke keadaan awal. Sebelum melakukan IDCT matriks hasil penyisipan akan dikalikan dengan matrik quant secara *element-wise* untuk dilakukan kuantisasi yang bertujuan untuk mengembalikan ke keadaan sebelum pembagaian ranah frekuensi. Untuk lebih penjelasan tentang IDCT lebih lanjut terdapat pada gambar 3.9.



**Gambar 3.10** Flowchart sub proses Inverse DCT

Dalam sub proses IDCT yang membedakan prosesnya dengan *Forward* DCT adalah perkalian matriks yang dilakukan pada akhir algoritma. Jika pada *forward* DCT perkalian melibatkan secara urut hasil perhitungan matriks T, matriks *pixel(x,y)* komponen warna (R/G/B), dan *transpose* dari matriks T (T’), sedangkan pada *inverse* DCT perkalian melibatkan secara urut *transpose* dari matriks T (T’), matrix DCT, dan hasil perhitungan matriks T. Untuk contoh perhitungan manual akan dijeaskan sebagai berikut:

1. Mengambil *pixel* dari komponen R/G/B citra. Sebagai contoh perhitungan ini akan menggunakan ukuran 4x4 *pixel*. Diibaratkan matriks DCT = matriks B.

B =

1. Mencari matriks T dengan persamaan 2.2. Berdasarkan matriks A, diketahui bahwa:

N = 4

Representasi posisi pixel (x, y) = (i, j)

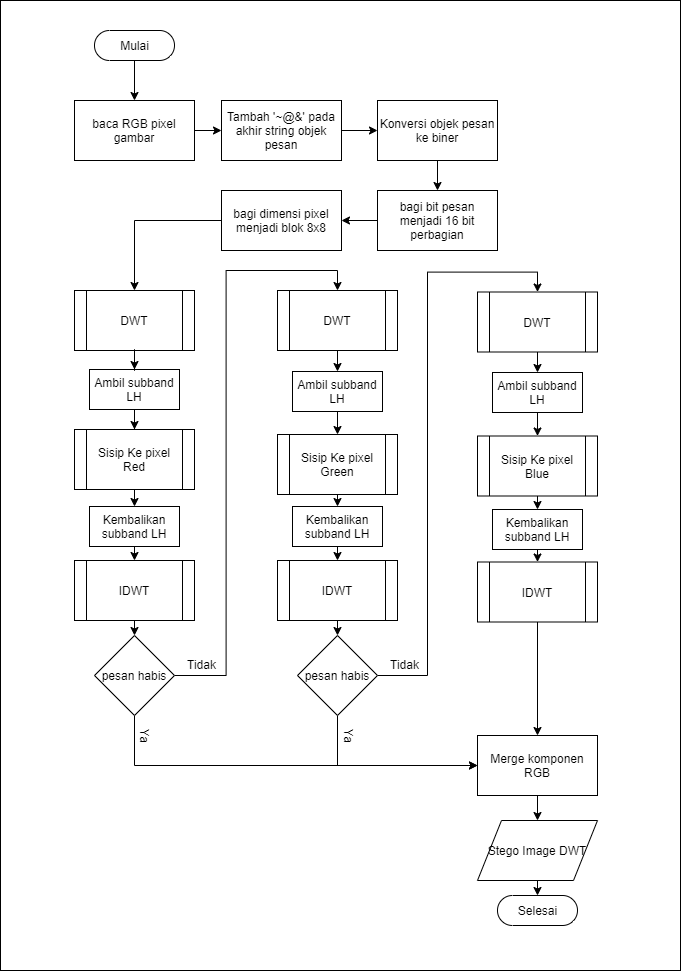
Jika x sama dengan 0, maka

Jika x tidak sama dengan 0, maka

Karena dimensi matriks sama dengan perhitungan DCT, maka hasil matriks T pada perhitungan IDCT merupakan hasil yang sama matriks T pada perhitungan DCT. Sehingga, nilai matriks T secara keseluruhan adalah:

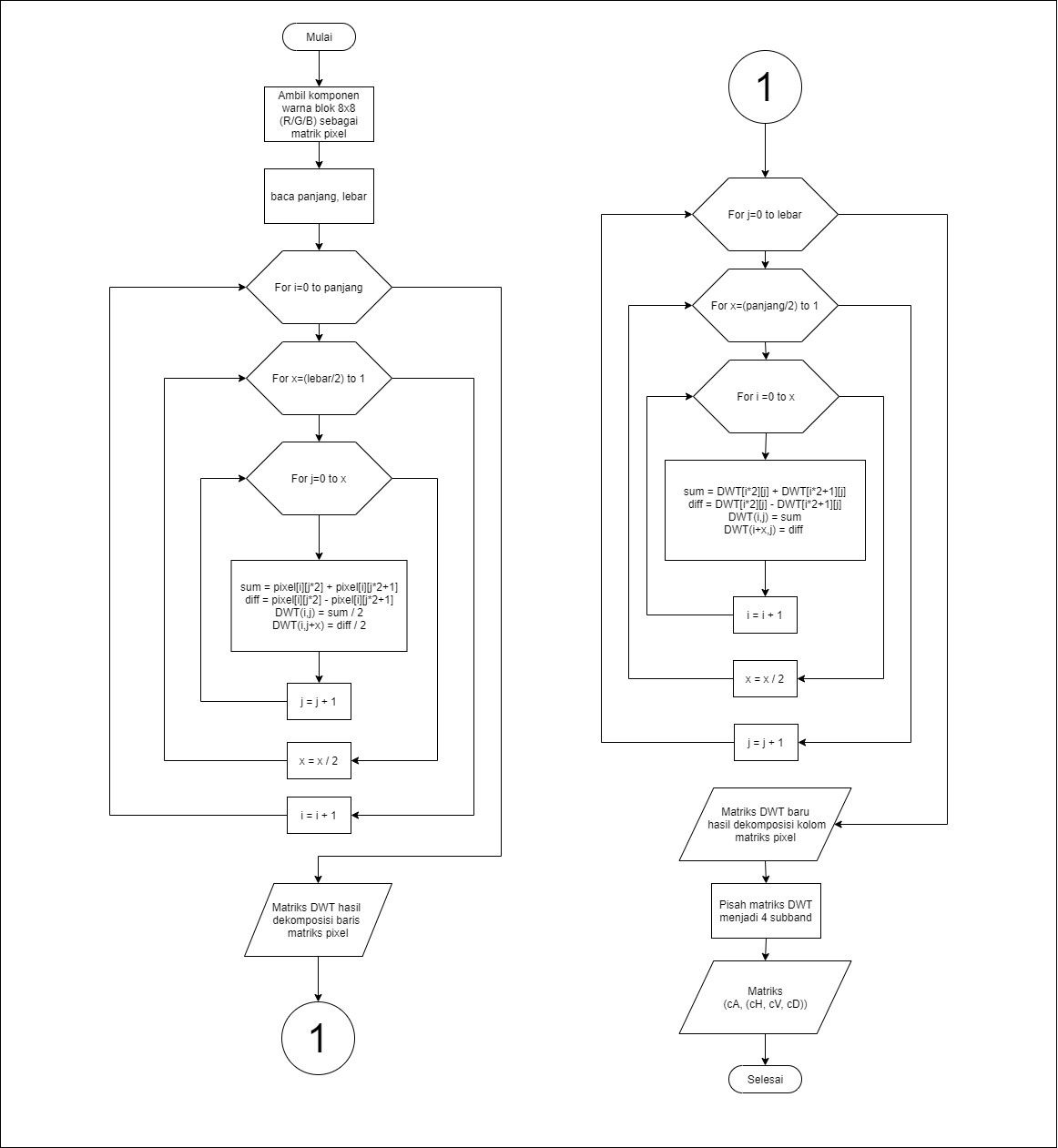
1. Menghitung matriks akhir IDCT dengan mengalikan matrik *Transpose* T, matriks B, dan matriks T secara berurutan. Selanjutnya elemen dijadikan integer untuk proses penggabungan komponen RGB menjadi citra utuh.

Setelah melalui proses IDCT, maka elemen-elemen pada matriks IDCT dikonversi menjadi *integer*. Hal ini dilakukan karena dalam proses penggabungan komponen R, G, dan B harus dimasukkan dalam bentuk *integer*. Terkait dengan penyisipan pesan, pada metode DCT juga menerapkan tiga ruang warna R, G, dan B. Hal ini dilakukan sehingga kapasitas pesan bisa diperbanyak. Ketika pesan belum habis maka penyisipan akan menggunakan komponen warna selanjutnya. Ketika pesan sudah habis disisipkan barulah citra dapat dibentuk kembali melalui proses penggabungan komponen RGB. Sama seperti pada penggabungan komponen citra pada proses LSB, penggabungan ini juga menggunakan fungsi merge() dari *library* Open-CV milik python.



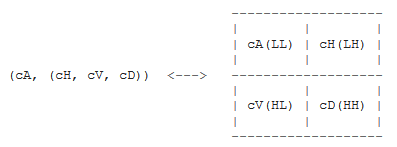
**Gambar 3.11** Flowchart sub proses Encode DWT

Metode *encode* DWT tidak berbeda jauh dengan proses *encode* DCT. Yang membedakan hanyalah jumlah ruang penyematan bit pesan dan metode DWT itu sendiri. Jika pada proses *encode* DCT pesan dibagi menjadi masing-masing bagian mendapat 64-bit maka pada DWT pesan hanya dibagi menjadi masing-masing 16-bit. Hal ini dikarenakan pada metode DWT nantinya citra juga akan dibagi menjadi blok 8x8, lalu blok tersebut akan mengalami dekomposisi sehingga dibagi menjadi 4 bagian yang akan dijelaskan kemudian. Hasil akhir dekomposisi tersebut adalah blok matriks *pixel* berukuran 4x4.



**Gambar 3.12** Flowchart sub proses DWT

Metode DWT (*Discrete Wavelete Transform*) dimulai dengan mengambil salah satu komponen warna RGB pada citra. Kemudian komponen warna tersebut diolah dengan dekomposisi ke arah baris dan ke arah kolom. Dari hasil dekomposisi tersebut akan terbentuk 4 matriks utama yaitu matriks cA, cH, cV, dan cD. Dekomposisi tersebut memanfaatkan fungsi dwt2() milik *PyWavelete*. Fungsi tersebut merupakan fungsi untuk menerapkan DWT pada matriks dua dimensi dengan metode *Haar*. Matriks-matriks tersebut mewakili *subband* dari hasil DWT yaitu *Low-Low*(LL), *Low*-*High*(LH), *High*-*Low*(HL), dan *High*-*High*(HH) seperti pada ilustrasi gambar 3.12.



**Gambar 3.13** Ilustrasi subband DWT

Untuk contoh perhitungan manual akan dijelaskan sebagai berikut:

1. Mengambil *pixel* dari komponen R/G/B citra. Sebagai contoh perhitungan ini akan menggunakan ukuran 4x4 *pixel*. Diibaratkan matriks pixel(x,y) = matriks A.

A =

1. Menentukan matriks hasil dekomposisi baris dari matriks A. Diibaratkan matrik baru tersebut adalah matriks T.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Sehingga didapatkan matrik baru sebagai berikut:

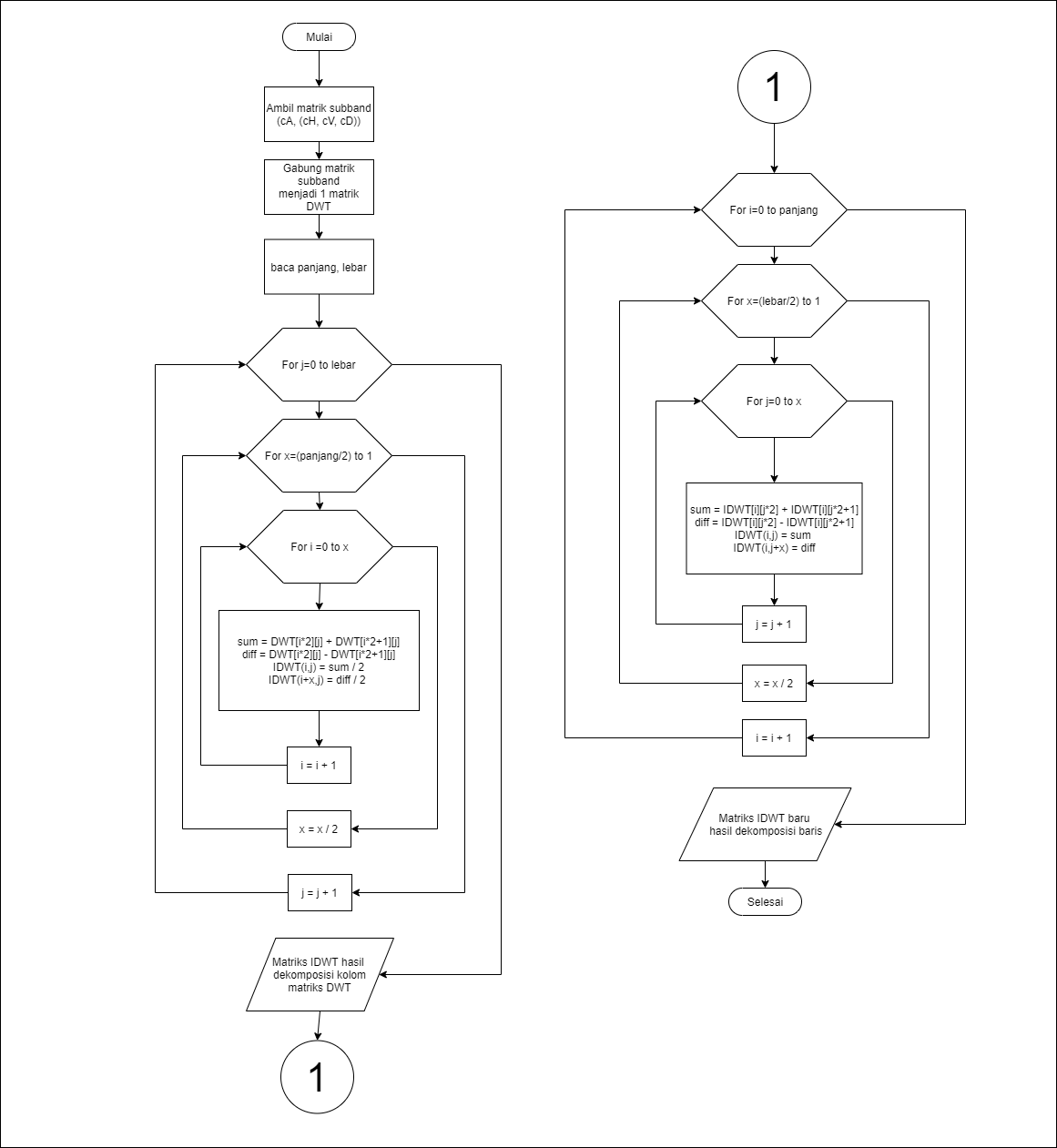
1. Menentukan matriks hasil dekomposisi kolom dari matriks hasil dekomposisi baris. Diibaratkan hasil dari matriks dekomposisi kolom adalah matriks T.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Sehingga matriks baru sebagai berikut:

1. Membagi matriks hasil dekomposisi kolom menjadi empat bagian, yaitu cA, (cH, cV, cD).

Contoh di atas merupakan gambaran penghitungan manual dekomposisi DWT. Bila terjadi pada matriks berukuran 8x8, maka *subband* akan menjadi 4x4 sehingga pesan yang dapat disisipi merupakan 16-bit dari keseluruhan pesan yang ada. Metode penyisipan pesan masih menggunakan cara yang sama, yaitu dengan memanfaatkan metode LSB yang memanfaatkan keseluruhan komponen warna RGB sebagai metode penyisip pesan, hanya saja melalui dekomposisi matriks terlebih dahulu. Tempat yang digunakan untuk metode ini ialah *subband Low-High* (LH) atau cH yang diubah tipe data elemennya menjadi bertipe *integer*. Setelah semua pesan berhasil dimasukkan maka proses selanjutnya adalah melakukan IDWT (*Inverse Discrete Wavelete Transform*) pada *subband-subband* yang telah dimasukkan pesan.



**Gambar 3.14** Flowchart sub proses Inverse DWT

Metode IDWT merupakan kebalikan dari proses DWT. Proses ini berfungsi mirip dengan proses IDCT, yaitu mengembalikan matriks dekomposisi menjadi matriks semula. Singkatnya matriks *subband*-*subband* akan disatukan kembali dan akan mengulang proses dekomposisi. Untuk contoh perhitungan manual akan dijelaskan sebagai berikut:

1. Mengambil *subband-subband* hasil dari proses penyisipan pesan. lalu digabungkan menjadi posisi semula seperti pada gambar 3.12. Diibaratkan matriks tersebut adalah matriks A.

A =

1. Menentukan matriks hasil dekomposisi kolom dari matriks T. Diibaratkan matrik baru tersebut adalah matriks T.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Sehingga didapatkan matrik baru sebagai berikut:

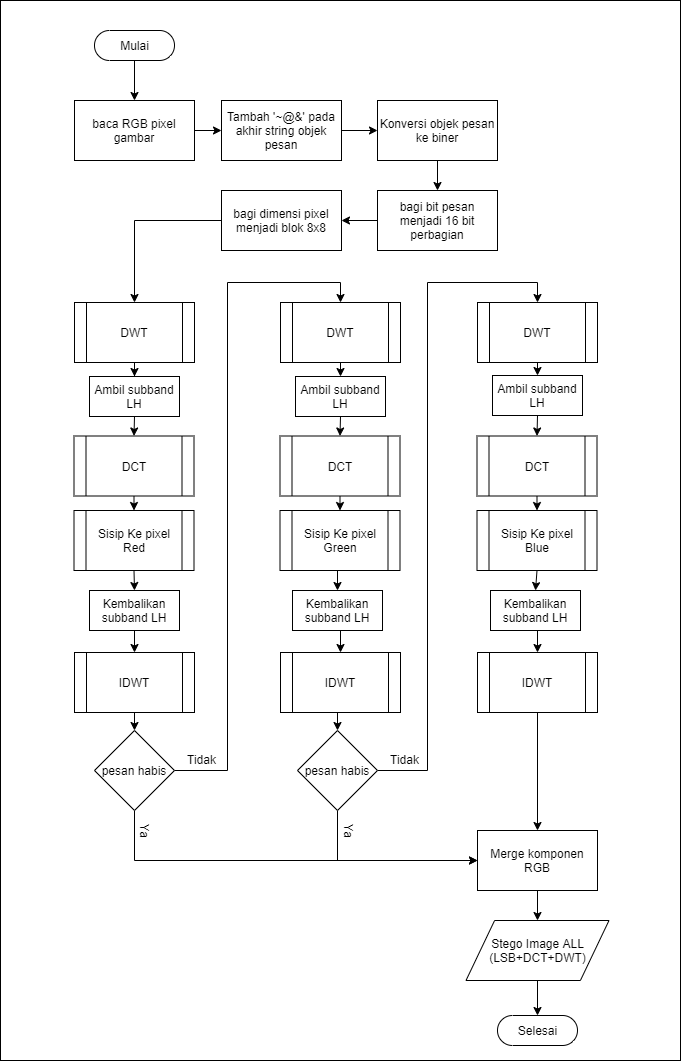
1. Menentukan matriks hasil dekomposisi baris dari matriks A. Diibaratkan hasil dari matriks dekomposisi kolom adalah matriks T.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Sehingga matriks baru sebagai berikut:

1. Mengubah tipe data setiap elemen matriks T menjadi *integer*.

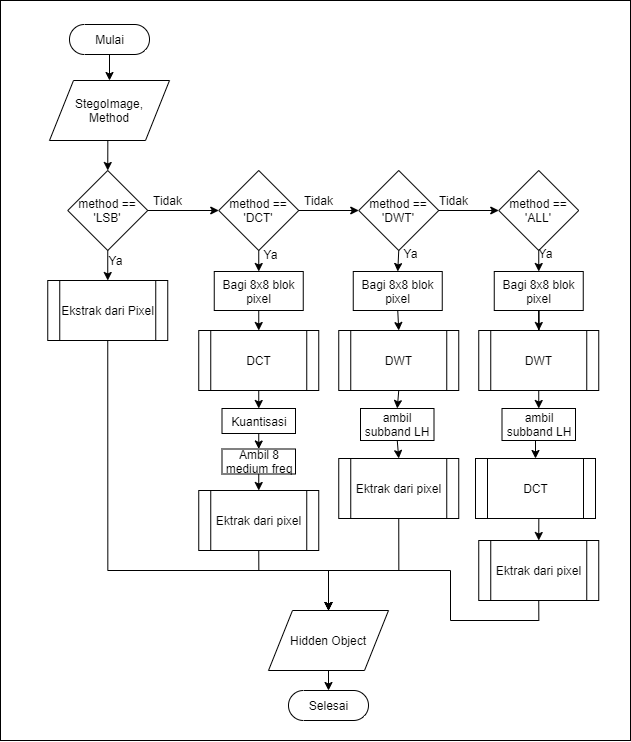
Setelah melewati proses IDWT maka komponen-komponen ruang warna akan digabungkan kembali menggunakan fungsi merge(). Proses penggabunggan RGB membutuhkan satu kesatuan komponen citra yang utuh dan sudah tidak terdekomposisi. Jika tidak, maka komponen-komponen tersebut tidak dapat disatukan menjadi citra utuh.



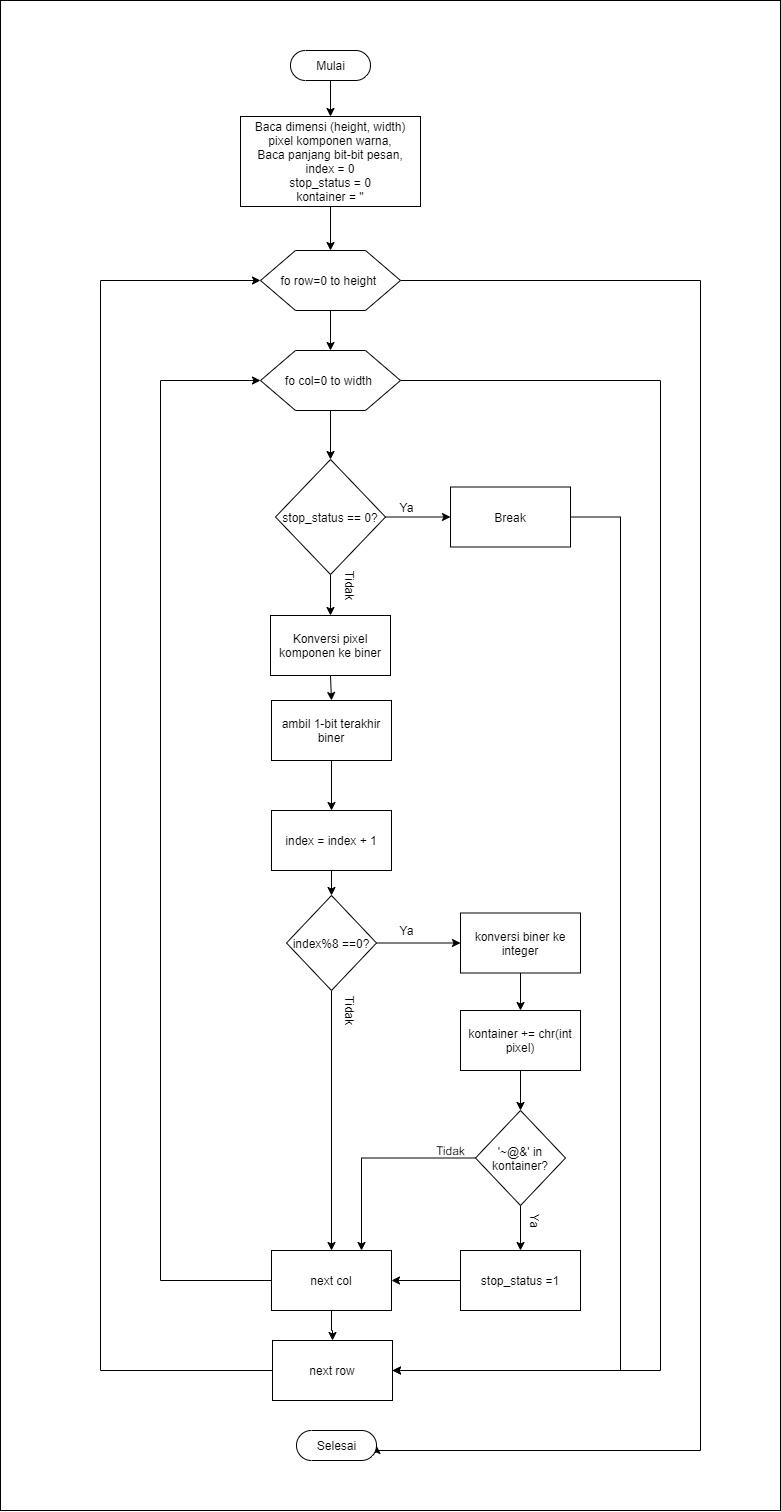
**Gambar 3.15** Flowchart sub proses encode Kombinasi (LSB+DCT+DWT)

Metode kombinasi yang dilakukan adalah menggunakan semua metode yang diajukan pada penelitian ini menjadi satu. Pada sub proses ini, citra akan mengikuti pembagian blok 8x8 dan pembagian pesan 16-bit perbagiannya. Mula-mula citra akan diambil komponen warnanya menggunakan fungsi split() seperti metode lain untuk mendapat data ruang warna RGB pada citra tersebut. Selanjutnya komponen ruang warna tersebut akan di dekomposisi DWT menggunakan fungsi dwt2() terlebih dahulu menghasil kan subband cA, cH, cV, cD. Kemudian subband cH akan dipilih untuk melakukan DCT dengan fungsi dct(). Pada metode ini tidak dilakukan kuantisasi seperti pada *encode* DCT biasa dikarenakan kuantisasi dilakukan secara *elemen wise* pada matriks 8x8 sedangkan yang disisipi adalah matriks *subband* cH berukuran 4x4. Hasil DCT tersebut akan disispkan pesan dengan metode LSB dengan panjang pesan masing-masing 16-bit karena matriks yang tersedia merupakan hasil pembagian blok 8x8 yang didekomposisi oleh DWT sehingga hanya tersedia matriks 4x4 yang berisi 16 elemen. Setelah pesan berhasil disisipkan, tidak seperti *encode* DCT biasa, pada metode ini tidak digunakan IDCT karena pesan akan hancur dikarenakan proses pembulatan oleh fungsi around() tanpa adanya kuantisasi karena alasan yang telah dijelaskan, sehingga selanjutnya dilakukan pengembalian *subband* cH dan melakukan idwt2() sehingga menjadi komponen warna utuh seperti semula. Terakhir maka seluruh komponen warna akan digabung menggunakan fungsi merge() untuk menjadi satu matriks citra utuh kembali.

Setelah semua metode melakukan penyisipan, maka untuk mendapat pesan tersembunyi di dalam sebuah citra perlu adanya ekstraksi pesan dari dalam citra tersebut. Untuk proses ektraksi pesan tersebut proses yang dijalankan tidak jauh berbeda dari proses encode atau penyisipan yang telah dilakukan. Yang membedakan hanyalah pemanggilan fungsi encode() diubah menjadi fungsi decode(). Keduanya memiliki perbedaan pada fungsinya. Apabila fungsi encode adalah menyisipkan pesan pada komponen warna maka fungsi decode() adalah sebaliknya, yaitu mengambil pesan yang berada pada komponen warna yang telah disisipi pesan. Pada proses ini komponen warna akan diambil nomor warnanya dan akan dikonversi menjadi biner 8-bit. Jika pada poses *encode* biner belakang akan diubah menjadi 1-bit pesan maka sebaliknya, pada proses *decode* bit terakhir pada bit nomor warna akan langsung dibaca dan dikumpulkan sampai membentuk 8-bit per karakter dan dikumpulkan hingga terbaca sebuah *stopper*. Fungsi decode() digambarkan dalam *flowchart* seperti yang terlihat pada gambar 3.16.

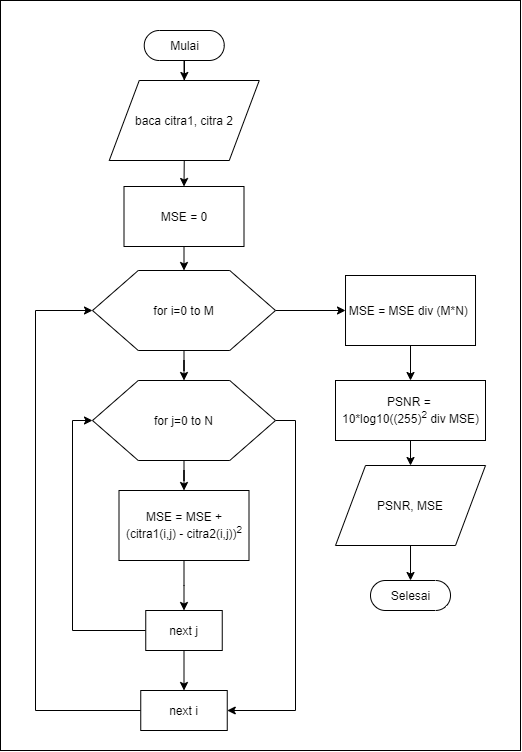


**Gambar 3.16** Flowchart Proses Decode



**Gambar 3.17** Flowchart subroutine ekstraksi pada pixel komponen warna

Metode ektraksi di atas pada dasarnya hampir sama dengan metode penyisipan pesan. Namun pada proses ekstraksi panjang bit tidak diketahui sehingga perlu mencari *stopper* berupa string ‘~@&’ untuk menghentikan ekstraksi pesan. Dalam proses di atas proses tidak akan berhenti sebelum menemukan *stopper*. *Stopper* akan diketahui ketika di kontainer mengandung ‘~@&’ pada string tersebut. Kontainer akan menerima karakter dari penyatuan bit-bit ketika index merupakan kelipatan delapan, maka bit-bit tersebut akan dikonversi menjadi karakter dan dimasukkan kedalam kontainer. Apabila dalam kontainer tidak ditemukan *stopper*, maka perulangan akan terus berjalan sampai pada komponen warna terakhir di akhir matriksnya. Namun apabila *stopper* ditemukan, maka perulangan akan berhenti dan pesan dapat ditampilkan.



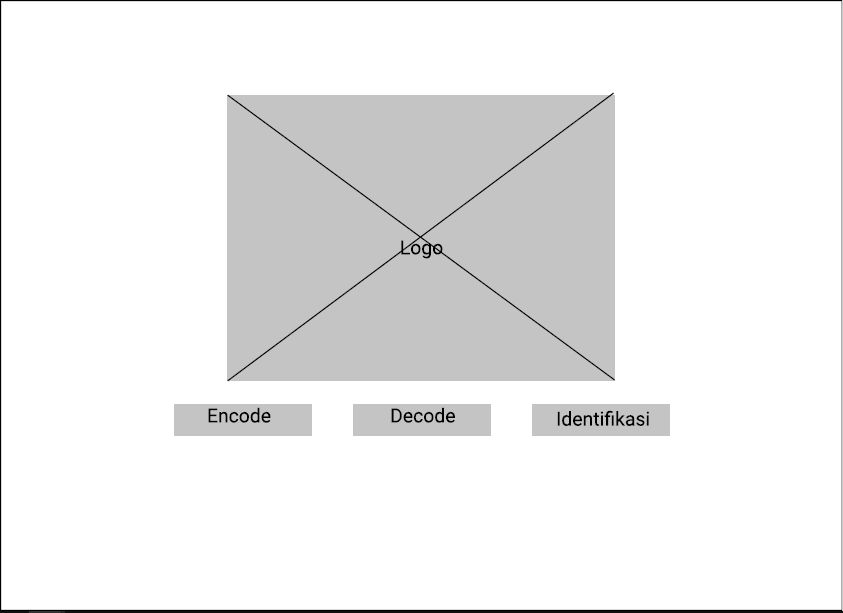
**Gambar 3.18** Flowchart subroutine penghitungan MSE dan PSNR

Pada *subroutine* penghitungan PSNR tedapat dua persamaan yaitu persamaan untuk menghitung MSE (*Mean Square Error*) untuk menghitung tingkat perbedaan dari kedua pixel citra dan PSNR (*Peak Signal to Noise Ratio*) untuk menghitung rasio perbedaan kedua citra. MSE menghitung setiap perbedaan pada pixel citra dari setiap komponen warna dan menghasilkan array yang berisi perbedaan dari setiap pixel. Selanjutnya hasil tersebut akan dirata-rata menggunakan fungsi pada *library* Numpy yaitu fungsi mean(). Makin banyak yang berbeda maka makin tinggi pula MSE dari kedua citra tersebut. Selanjutnya PSNR mendapat acuan tingkat perbedaan dari kedua citra dari hasil penghitungan MSE. Makin tinggi hasil persamaan MSE maka makin kecil nilai persamaan PSNR.

#### Perancangan Antarmuka

Perancangan *interface* atau antarmuka merupakan suatu kebutuhan yang penting dalam pembuatan aplikasi dan merupakan mekanisme komunikasi antara *user* dengan aplikasi. Hal tersebut memiliki tujuan untuk mengkomunikasikan fitur-fitur yang ada didalam sebuah aplikasi tersebut sehingga dapat dengan mudah digunakan. Antarmuka pada aplikasi yang akan dibuat merupakan tampilan *web*-*based* yang diakses menggunakan *desktop* *browser*. Perancangan antarmuka dapat dilihat pada gambar 3.19 sampai dengan gambar 3.24.

##### ***Wireframe* Antarmuka Halaman *Home***



**Gambar 3.19** Wireframe Tampilan Awal

Pada perancangan antarmuka aplikasi ini terdapat tiga menu utama yang dapat diakses oleh pengguna. Adapun penjelasan mengenai tombol-tombol menu yang terdapat pada gambar dijelaskan sebagai berikut:

1. *Encode*

Menu untuk melakukan penyisipan steganografi

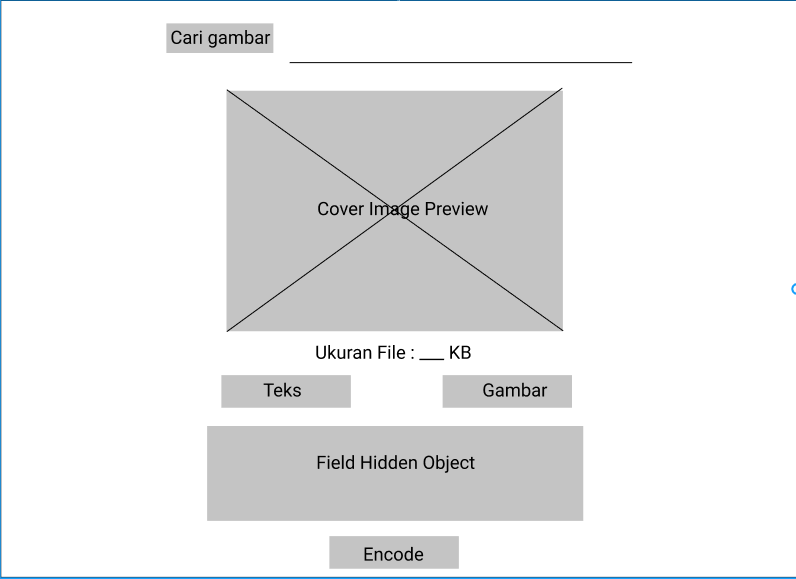
1. *Decode*

Menu untuk melakukan ekstraksi *hidden* *object*

1. Identifikasi

Menu untuk melakukan identifikasi pada dua citra. Baik citra *cover* dan citra stego maupun citra stego original dan citra stego yang diserang.

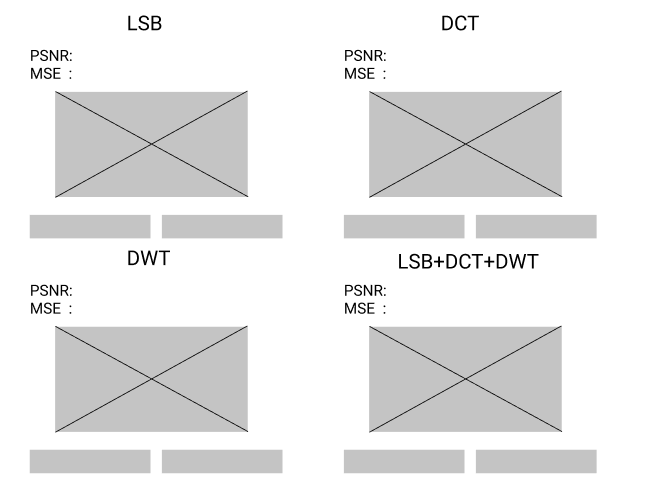
##### ***Wireframe* Antarmuka Halaman *Encode***



**Gambar 3.20** Wireframe Tampilan Encode

Pada halaman *encode* digunakan untuk melakukan penyisipan *hidden* *object* ke dalam *cover* *image* seperti pada gambar 3.20. Pada halaman tersebut terdapat bagian pencarian citra *cover* dan bagian *field* *hidden* *object.* Pada *field* *hidden* *object* dapat disesuaikan sesuai keinginan pengguna akan menyisipkan teks atau gambar. Setelah melakukan penyisipan maka akan terbuka sub halaman berupa kumpulan hasil citra stego beserta hasil perhitungan PSNR pada masing-masing citra stego.

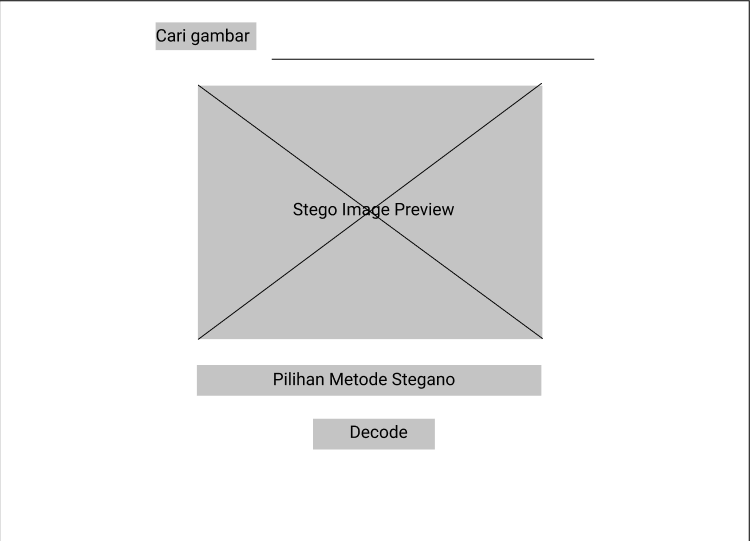
##### ***Wireframe* Antarmuka Halaman Hasil *Encode***



**Gambar 3.21** Wireframe Tampilan Hasil Encode

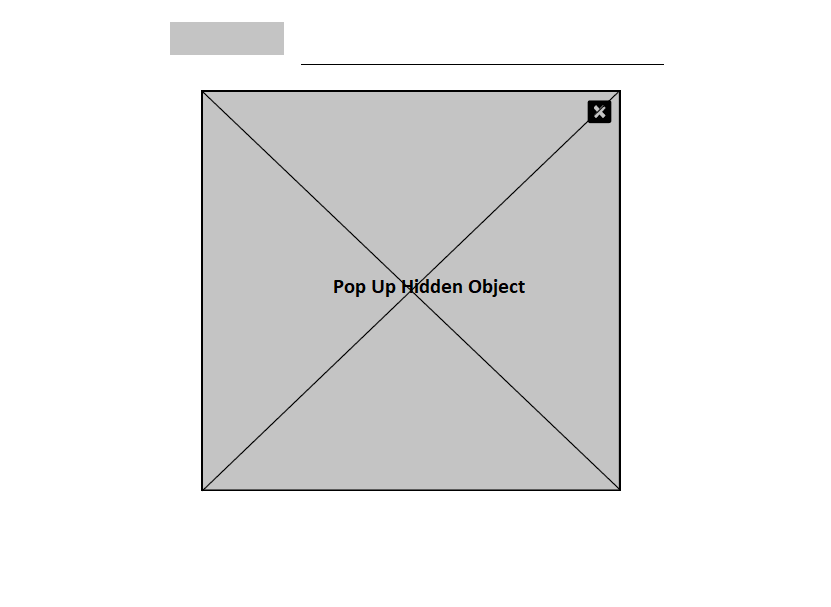
Pada halaman ini terdapat keempat hasil dari encode steganografi dan masing-masing mempuyai komponen yaitu nama metode penyisipan, PSNR, MSE, citra stego, tombol *decode*, dan tombol unduh. Tombol *decode* digunakan untuk memastikan citra stego mempunyai isi atau citra stego sudah disisipi pesan, sedangkan tombol unduh digunakan untuk menyimpan citra.

##### ***Wireframe* Antarmuka Halaman *Decode***



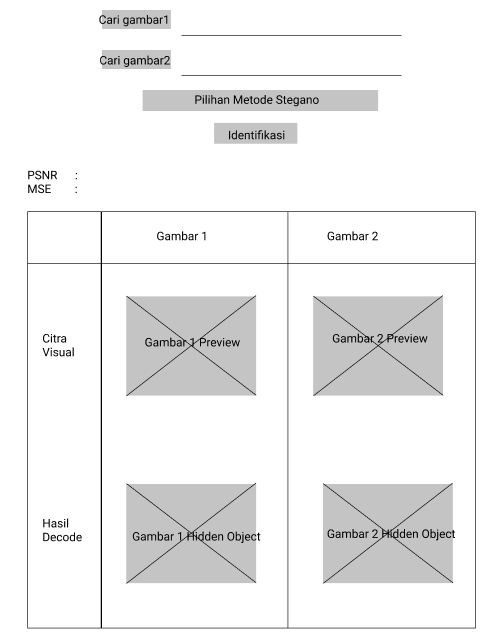
**Gambar 3.22** Wireframe Tampilan Decode

Pada halaman menu *decode* terdapat pencarian citra yang akan digunakan untuk mencari citra stego dan tombol *decode*. Ketika tombol ditekan maka sebuah *pop* *up* yang berisikan hasil dari ekstrak *hidden* *object* akan ditampilkan seperti pada gambar 3.23.



**Gambar 3.23** Wireframe Tampilan Pop Up Hasil Ekstraksi hidden object

##### ***Wireframe* Antarmuka Halaman Identifikasi**



**Gambar 3.24** Wireframe Tampilan Identifikasi

Pada halaman identifikasi akan ditampilkan bagian pencarian dua buah citra dan sebuah table perbandingan dari kedua citra yang telah dimasukkan. Kedua citra tersebut akan dibandingkan dari segi visual, hasil *decode* (*hidden* *object*), dan ukuran *file*. Serta sebelumnya terdapat tingkat PSNR dari kedua buah citra sebagai acuan tingkat perbedaan pada kedua citra.

#### Perancangan Pengujian

Pada penelitian ini pengujian yang dilakukan merupakan pengujian kualitas *robustness* *hidden object* dari citra stego dengan menggunakan *benchmark* *stirmark* serta pengujian *fidelity* citra stego dengan tolak ukur visual, selisih ukuran file, dan PSNR. Pengujian *stirmark* dilakukan setelah system dapat berjalan dan dapat digunakan. Pengujian ini akan memanfaatkan penyerangan pada citra stego berupa *geometric distortion attack* (serangan geometris) dan *compression* yang biasa dilakukan sebagai uji stirmark pada citra steganografi maupun *watermark*. Uji ini akan memanipulasi citra dengan cara dipotong (*crop*), diputar (*rotate*), dibalik (*flip*), dan diubah dimensinya (*resize*). Selanjutnya untuk menyelesaikan masalah utama pada penelitian ini citra stego akan diuji juga menggunakan media yang biasa digunakan pada kehidupan sehari-hari. Citra akan dikirim menggunakan fitur pesan pada aplikasi Whatsapp, Telegram, Line, E-Mail, Google Drive, serta citra juga akan dikirimkan menggunakan *pendrive*/*flashdisk*. Skenario pengujian untuk penelitian ini diantaranya sebagai berikut:

1. Skenario Uji *Robustness*

Skenario uji *robustness* pada citra stego meliputi tujuh tahap utama. Tujuh tahap tersebut diantaranya:

1. Tanpa Serangan

Pesan akan diekstrak tanpa perlu citra stego dimodifikasi atau diserang.

1. Dipotong (*Cropping*)

Citra stego akan dipotong setengah dari citra dari keempat sisi sebelum diekstrak. Skenarionya sebagai berikut:

* 1. 50% Horizontal Atas
  2. 50% Horizontal Bawah
  3. 50% Vertikal Kiri
  4. 50% Vertikal Kanan

1. Diputar (*Rotate*)

Citra stego akan diputar beberapa derajat ke kiri dan kekanan sebelum diekstrak pesan tersembunyinya. Skenarionya sebagai berikut:

* 1. 90° Ke Kiri
  2. 180° Ke Kiri
  3. 90° Ke Kanan
  4. 180° Ke Kanan

1. Dibalik (*Flip*)

Citra stego akan dibalik citranya sebelum dilakukan ekstraksi. Skenario sebagai berikut:

* 1. Horizontal
  2. Vertikal

1. Diubah dimensi (*Resize*)

Dimensi Panjang dan lebar dari citra akan diubah sebelum dilakukan ekstraksi. Skenarionya sebagai berikut:

* 1. 25%
  2. 50%
  3. 75%

1. Penggunaan aplikasi pihak ketiga

Citra stego akan dikirimkan dan/atau diunggah ke dalam aplikasi pihak ketiga. Setelah itu barulah pesan akan diekstrak. Aplikasi yang digunakan untuk scenario sebagai berikut:

* 1. Aplikasi *Chat* (WhatsApp/LINE/Telegram)
  2. Email
  3. Google Drive
  4. *Flashdisk*

Data-data hasil pengujian *robustness* di atas akan dicatat seperti pada table berikut:

**Tabel 3.1** Tabel pengujian robustness

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Citra Stego | Skenario Uji | Hasil Ekstraksi | Keterangan Hasil |
| 1 |  | Tidak diserang |  |  |
| 2 |  | *Cropping* 50% Horizontal Atas |  |  |
| 3 |  | *Cropping* 50% Horizontal Bawah |  |  |
| 4 |  | *Cropping* 50% Vertikal  Kiri |  |  |
| 5 |  | *Cropping* 50% Vertikal Kanan |  |  |
| 6 |  | *Rotate* 90 Kiri |  |  |
| 7 |  | *Rotate* 90 Kanan |  |  |
| 8 |  | *Rotate* 180 |  |  |
| 9 |  | *Flip* Horizontal |  |  |
| 10 |  | *Flip* Vertikal |  |  |
| 11 |  | *Resize* 25% |  |  |
| 12 |  | *Resize* 50% |  |  |
| 13 |  | *Resize* 75% |  |  |
| 14 |  | Kirim WhatsApp (*Image*) |  |  |
| 15 |  | Kirim WhatsApp (*File*) |  |  |
| 16 |  | Kirim LINE (*Standart*) |  |  |
| 17 |  | Kirim LINE (Original) |  |  |
| 18 |  | Kirim Telegram (*Image*) |  |  |
| 19 |  | Kirim Telegram (*File*) |  |  |
| 20 |  | Kirim Gmail |  |  |
| 21 |  | Upload Gdrive |  |  |
| 22 |  | Kirim Flashdisk |  |  |

1. Skenario Uji *Fidelity*

Skenario uji *fidelity* pada citra stego meliputi empat komponen utama. Empat komponen tersebut diantaranya:

1. Perbandingan secara visual

Citra *cover* dan citra stego akan dibandingkan secara visual atau tampilan dan ditaruh sejajar agar mata penguji dapat melihat secara langsung apakah terdapat perbedaan pada kedua citra atau tidak dari segi visual.

1. Perbandingan ukuran *file*

Ukuran *file* dari citra cover dan citra stego akan dibandingkan apakah terdapat selisih diantara kedua citra tersebut.

1. Perbandingan tingkat PSNR

Tingkat PSNR akan menjadi tolak ukur tingkat kualitas perbedaan antara citra *cover* dan citra stego. Nilai PSNR didapatkan menggunakan persamaan (2.9) serta menggunakan penghitungan persamaan (2.10) yaitu rumus MSE untuk mendapatkan nilai PSNR.

1. Perbandingan RGB

Perbandingan RGB akan digunakan untuk membandingkan perbedaan penggunaan ruang warna yang digunakan pada citra asli dan citra stego.

Beberapa pengujian *fidelity* (perbandingan visual dan penggunan ruang warna) akan menggunakan aplikasi Beyond Compare untuk mendapatkan hasil pengujian yang maksimal. Data-data hasil pengujian *fidelity* di atas akan dicatat seperti pada table berikut:

**Tabel 3.2** Tabel pengujian fidelity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Cover image | Stego Image | Skenario Uji | Hasil Uji | Keterangan |
| 1 |  |  | Visual |  |  |
| Selisih Ukuran *File* |  |  |
| PSNR |  |  |
| RGB |  |  |

# BAB IV HASIL, PENGUJIAN, DAN PEMBAHASAN

Hasil, pengujian, dan pembahasan merupakan tahap dimana sistem ini siap dioperasikan pada keadaan yang sebenarnya. Dari pembahasan ini akan diketahui apakah sistem yang dibuat dapat menghasilkan tujuan yang sesuai berdasarkan analisis dan perancangan yang telah dilakukan.

## Hasil Penelitian

Pada tahap ini disajikan implementasi analisis *robustness* dan *fidelity* pada steganografi citra digital dengan metode *Least Significant Bit* (LSB), *Discrete Cosine Transform* (DCT), dan *Discrete Haar Wavelet Transform* (DWT) yang mencakup gambar tampilan dari aplikasi serta modul-modul program pembentuknya. Pada implementasi sistem ini terdapat halaman utama, halaman *encode* beserta hasilnya, halaman *decode*, dan halaman identifikasi. Halaman utama ketika aplikasi pertama kali dijalankan adalah sebagai berikut:



**Gambar 4.1** Halaman utama pada aplikasi

Halaman utama aplikasi ini memiliki tiga tombol yaitu *Encode* untuk menuju halaman *encode*, Decode untuk menuju halaman *decode*, dan Identify untuk menuju halaman identifikasi. Halaman *encode* digunakan untuk menyisipkan pesan ke citra digital. Pada bagian *encode* juga terdapat hasil *encode* yang akan menampilkan hasil citra digital yang telah disisipi pesan. Halaman *decode* digunakan untuk mengekstrak pesan dari citra digital. Halaman identifikasi digunakan untuk mengidentifikasi dua citra yaitu citra stego asli dan citra stego yang telah dilakukan serangan. Modul program untuk tampilan home dapat dilihat pada modul program 4.1 sampai modul program 4.3.

|  |
| --- |
| import os  import base64  import cv2  import numpy as np  import math  from pywt import dwt2, idwt2  from flask import Flask, render\_template, request, redirect, url\_for, flash, send\_from\_directory, jsonify, send\_file |

**Modul Program 4.1** Import Library

|  |
| --- |
| @app.route('/')  def index():  return render\_template('hal\_home.html') |

**Modul Program 4.2** Proses routing menuju halaman home

|  |
| --- |
| <head>  <link rel="stylesheet" href="{{ url\_for('static', filename='css/materialize.min.css') }}">  <!--Import Google Icon Font-->  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">  <!--Import materialize.css-->  <link type="text/css" rel="stylesheet" href="{{ url\_for('static', filename='css/materialize.min.css') }}" media="screen,projection"/>  {% block head %} {% endblock %}  </head>  <body>  <center>  <div class="container">  {% block body %}{% endblock %}  </div>  </center>  <!-- JQuery -->  <script type="text/javascript" src= "https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>  <!--JavaScript at end of body for optimized loading-->  <script type="text/javascript" src="{{ url\_for('static', filename='js/materialize.min.js') }}"></script>  <script src="{{ url\_for('static', filename='js/main.js') }}"></script>  </body>  </html> |

**Modul Program 4.3** Source code base template untuk dasar templating

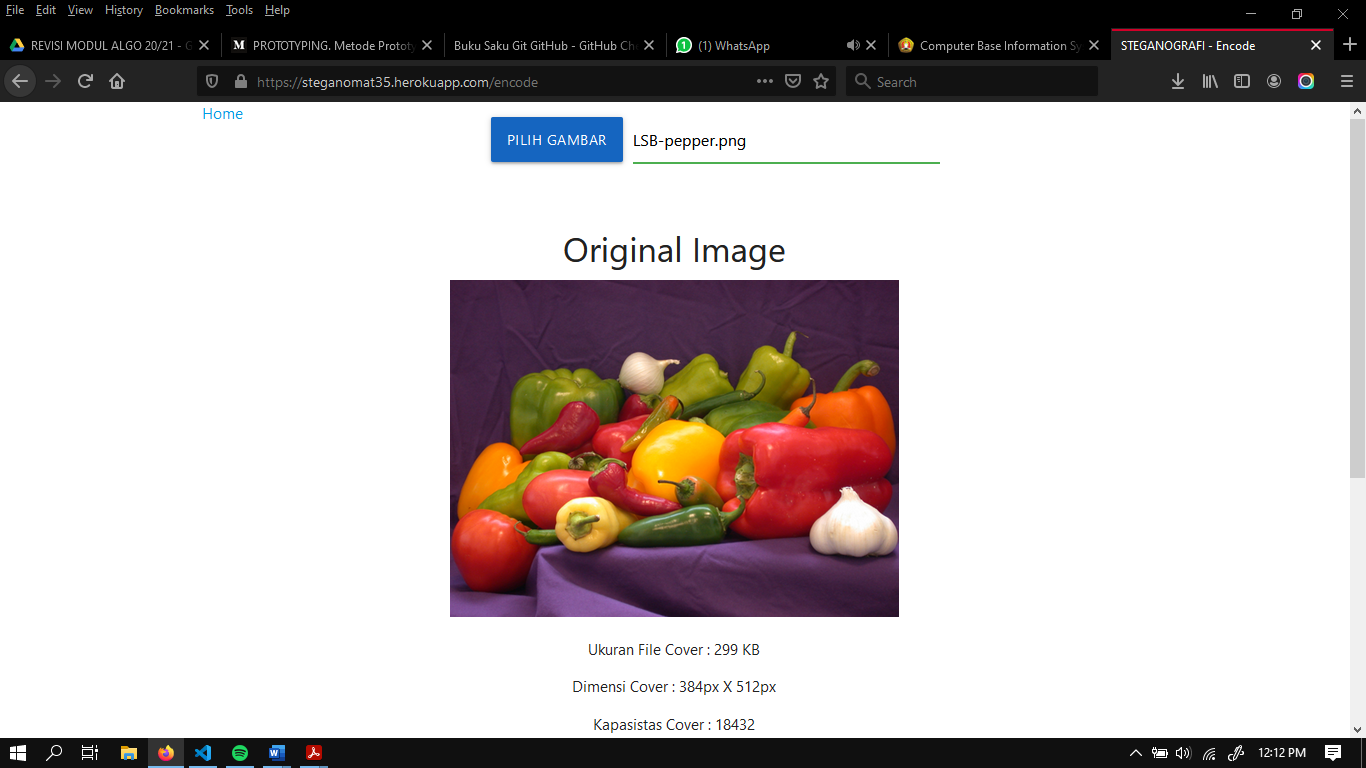
|  |
| --- |
| {% extends 'base.html' %}  {% block head %}  <title>STEGANOGRAFI - Home</title>  {% endblock %}  {% block body %}  <div class="row">  <div class="col m3"></div>  <div class="col s12 m6">  <img class="responsive-img " src="/static/assets/ostrich.png">  <div class="row">  <div class="col s4">  <a class="btn tooltipped blue darken-3" href="/encode" data-position="bottom" data-tooltip="Proses encoding merupakan proses untuk menyematkan suatu objek pada cover">ENCODE</a>    </div>  <div class="col s4">  <a class="btn tooltipped blue darken-3" href="/decode" data-position="bottom" data-tooltip="Proses decoding merupakan proses untuk mengeluarkan objek tersembunyi pada cover">DECODE</a>    </div>  <div class="col s4">  <a class="btn tooltipped blue darken-3" href="/identify" data-position="bottom" data-tooltip="Proses identifikasi bertujuan untuk meneliti hasil encode dan decode sehingga dapat dilakukan perbandingan kualitas serta ketahanan terhadap serangan pada objek stego">IDENTIFY</a>    </div>  </div>  </div>  </div>  {% endblock %} |

**Modul Program 4.4** Source code tampilan home

*Source code* di atas menggunakan dasar jinja *templating* dari flask python sehingga perlu melakukan *import* seluruh *library* terlebih dahulu. Ketika aplikasi diakses hal yang pertama dilakukan adalah melakukan *import* *library* pada modul program 4.1. yang selanjutnya akan melakukan *routing* ke halaman *home* sebahai halaman utama seperti pada modul program 4.2. Proses *routing* akan mengakses *template* base dan hal\_home sehingga dapat menampilkan halaman home. *Source code template* base nantinya juga akan digunakan untuk menampilkan halaman-halaman lain ketika diakses. Pada halaman *home* telah terdapat tiga *button* dengan alamat *routing* yang berbeda sesuai dengan halaman yang akan diakses.

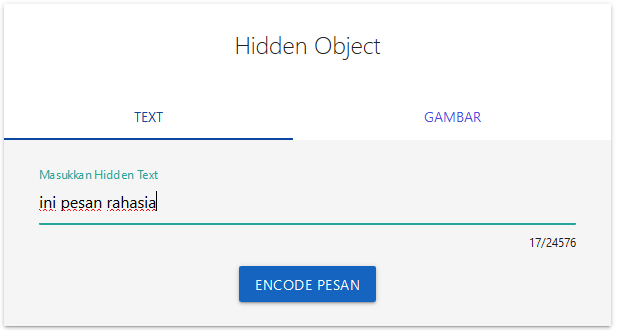
### Halaman Encode

Pada tahap ini terdapat proses input citra dan pesan, proses penyisipan pesan ke citra dan proses penghitungan PSNR. Pada tahap ini mula-mula pengguna akan memasukkan citra *cover* yang akan disisipi pesan beserta objek pesan yang akan disembunyikan. Citra yang digunakan dibatasi hanya yang memiliki ekstensi .jpg, .png, dan .bmp. Tahapan input citra dimulai ketika pengguna pada button cari gambar yang terdapat pada halaman encode. Selanjutnya *browser* akan membuka direktori dan pengguna dapat mencari citra yang akan disisipi pesan. Tampilan setelah citra terpilih dapat dilihat pada gambar 4.2.

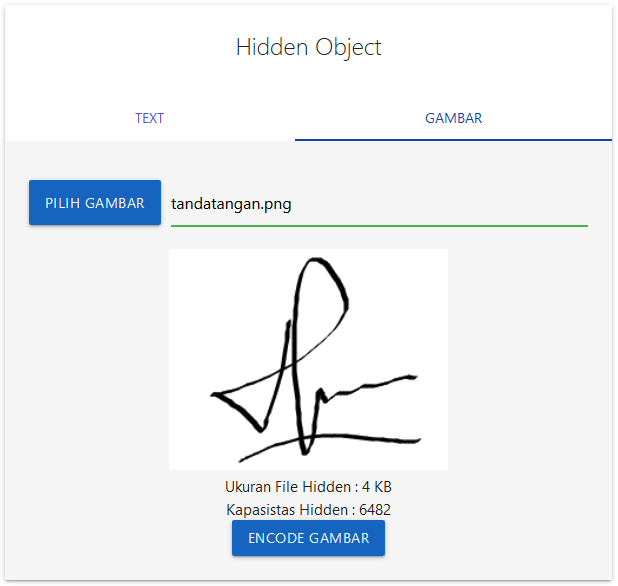


**Gambar 4.2** Halaman encode dengan penyisipan pesan teks

Setelah citra *cover* berhasil dimasukkan, selanjutnya input pesan yang akan disembunyikan. Pesan bisa berbentuk teks atau citra lain. Terdapat perbedaan pada tampilan bagian input pesan berbentuk teks dan pesan berbentuk citra karena masing-masing menggunakan komponen HTML yang berbeda. Untuk tampilan dari input pesan dapat dilihat pada gambar 4.3 untuk pesan berbentuk teks dan gambar 4.4 untuk pesan berbentuk citra.



**Gambar 4.3** Tampilan bagian input pesan berupa teks



**Gambar 4.4** Tampilan bagian input pesan berupa citra

Pada bagian *input* pesan, terdapat pembatas kapasitas pesan yang masuk sesuai dengan kapasitas yang tersedia pada citra *cover*. Hal ini untuk antisipasi terjadi kelebihan kapasitas pada aplikasi kedepannya. Pembatas tersebut ditampilkan sebagai keterangan dibawah bagian input pesan sehingga pengguna dapat melihat apakah pesan yang dimasukkan melebihi kapasitas atau tidak. *Source* *code* tampilan halaman *encode* dapat dilihat pada modul program 4.5 sampai dengan modul program 4.22

|  |
| --- |
| @app.route('/encode')  def show\_encode():  if os.path.exists(RESULT\_FOLDER + encode\_folder):  return render\_template('hal\_encode.html')  else:  os.mkdir(RESULT\_FOLDER + encode\_folder)  return render\_template('hal\_encode.html') |

**Modul Program 4.5** Potongan source code routing menuju halaman encode

|  |
| --- |
| {% extends 'base.html' %}  {% block head %}  <title>STEGANOGRAFI - Encode</title>  {% endblock %}  {% block body %}  <div style="float: left;"><a href="/">Home</a></div>  <form action="/go\_encode" method="POST" enctype=multipart/form-data>  <div class="row center-align">  <div class="col l3"></div>  <div class="col s12 l6">  <div class="file-field input-field">  <div class="btn blue darken-3">  <span>Pilih Gambar</span>  <input type="file" id="image-up" name="ori\_image" accept="image/\*" onchange="original();validateImageSize();">  </div>  <div class="file-path-wrapper">  <input class="file-path validate" type="text" placeholder="Pilih Cover Image">  </div>  </div>  </div>  </div>  <div class="row center-align">  <div class="col l3"></div>  <div class="col center-align s12 l6">  <h4>Original Image</h4>  <img id="ori-image" class="responsive-img">  <p id="sizeOri">Ukuran File : KB</p>  <p id="pxOri">Dimensi : x</p>  <p id="kapOri">Kapasitas : </p>  </div>  </div>  <div class="row">  <div class="col l2"></div>  <div class="col s12 l8">  <div class="card">  <div class="card-content">  <span class="card-title">Hidden Object</span>  </div>  <div class="card-tabs">  <ul class="tabs tabs-fixed-width"> |

**Modul Program 4.6** Source code tampilan halaman encode

|  |
| --- |
| <li class="tab"><a class="active" href="#textField">Text</a></li>  <li class="tab"><a href="#imageField">Gambar</a></li>  </ul>  </div>  <div class="card-content grey lighten-4">  <div id="textField">  <div class="input-field col s12">  <textarea id="textarea2" class="materialize-textarea" data-length="0" name="hidden\_text"></textarea>  <label for="textarea2">Masukkan Hidden Text</label>  </div>  <input type="submit" value="Encode Pesan" name="action" class="btn blue darken-3" id="encode\_text">  <!--a href="/encode\_pesan" class="btn blue darken-3">Encode Pesan</a-->  </div>  <div id="imageField">  <div class="file-field input-field">  <div class="btn blue darken-3">  <span>Pilih Gambar</span>  <input onchange="hiddenimg();validateImageSize();" type="file" name="hidden\_image" accept="image/\*" id="hiding-up">  </div>    <div class="file-path-wrapper">  <input class="file-path validate" type="text" placeholder="Pilih Hidden Image">  </div>  </div>  <img id="hid-image" class="responsive-img" width="50%"><br>  <p id="sizeHidden">Ukuran File : KB</p>  <p id="kapHidden">Kapasitas : </p>  <input type="submit" value="Encode Gambar" name="action" class="btn blue darken-3" id="encode\_gambar">  <!--a href="/encode\_gambar" >Encode Gambar</a-->  </div>  </div>  </div>  </div>  </div>  </form>  {% endblock %} |

**Modul Program 4.7** Lanjutan Source code tampilan halaman encode

Proses *encode* akan dimulai ketika citra *cover* dan pesan sudah dimasukkan dan pengguna menekan tombol *encode* yang berada dibawah bagian input pesan. Proses *encode* akan menggunakan metode LSB, DCT, DWT, dan kombinasi semua metode yang akan dijalankan secara urut. Proses encode dapat dilihat pada modul program 4.8 sampai dengan modul program 4.20 yang merupakan *source code* dalam satu *method* yang dibagi-bagi menjadi beberapa bagian metode *encode*.

|  |
| --- |
| @app.route('/go\_encode', methods=['GET', 'POST'])  def go\_encode():  # check if the post request has the file part  if request.method == 'POST' and 'ori\_image' in request.files:  cover\_img = request.files['ori\_image']  if request.form['action'] == 'Encode Pesan' and request.form['hidden\_text'] != '' and cover\_img.filename != '':  new\_cover\_img = go\_upload(cover\_img,'')  hidden\_text = request.form['hidden\_text']  result\_data = process\_encode(new\_cover\_img,hidden\_text)  if result\_data != '':  return render\_template('hal\_hasilencode.html', result=result\_data)  else:  return render\_template('hal\_error.html')  elif request.form['action'] == 'Encode Gambar' and 'hidden\_image' in request.files and cover\_img.filename != '':  hidden\_img = request.files['hidden\_image']  new\_cover\_img = go\_upload(cover\_img,'')  new\_hidden\_img = go\_upload(hidden\_img,'')  #img = Image.open(UPLOAD\_FOLDER+"/"+hidden\_img.filename)  #proses read dari file  f = open(UPLOAD\_FOLDER+"/"+new\_hidden\_img, 'rb')  img = f.read()  b64 = str(base64.b64encode(img))  b64 = b64.split("'")  b64\_hidden = 'data:image/png;base64, '+b64[1]  result\_data = process\_encode(new\_cover\_img,b64\_hidden)  if result\_data != '':  # return '<img src="'+b64\_hidden+'" />'  return render\_template('hal\_hasilencode.html', result=result\_data)  else:  return render\_template('hal\_error.html')  else:  return render\_template('hal\_error.html')  else:  return render\_template('hal\_error.html') |

**Modul Program 4.8** Potongan source code routing menuju method proses encode

Proses pada modul program 4.8 merupakan proses memasukkan citra *cover* dan pesan sebelum diolah pada method proses *encode*. Hal ini untuk memastikan citra cover dan pesan benar-benar ada dan bukan bernilai *null*. Jika terdapat kesalahan maka proses encode tidak akan dijalankan, melainkan akan dialihkan menuju halaman error.

|  |
| --- |
| def process\_encode(cover\_img, hidden\_text):  # baca gambar  coverImage = cv2.imread(UPLOAD\_FOLDER + "/" + cover\_img,1)  cover = coverImage  # convert BGR ke RGB  coverImage = cv2.cvtColor(coverImage, cv2.COLOR\_BGR2RGB)  # preprocess pesan |

**Modul Program 4.9** Potongan source code proses encode sebelum penyisipan

|  |
| --- |
| hid = hidden\_text+'~@&'  hit\_hidden = len(hid)    # wadah buat pesan  kontainer = ''  kontainer16 = ''  kontainer8 = ''  kont\_div = [['']]  kont\_div16 = [['']]  kont\_div8 = [['']]  x = 0  y = 0  for i in range(hit\_hidden):  # persiapan pembagian bagian2 pesan  kontainer = kontainer + f'{ord(hid[i]):08b}'  kontainer16 = kontainer16 + f'{ord(hid[i]):08b}'  kontainer8 = kontainer8 + f'{ord(hid[i]):08b}'    # pesan dibagi menjadi 8-bit perbagian  if i == 0:  kont\_div8[0] = kontainer8  else:  kont\_div8.append(kontainer8)  kontainer8 = ''  # pesan dibagi menjadi 16-bit perbagian  if (i+1)%2 == 0:  if y == 0:  kont\_div16[y] = kontainer16  else:  kont\_div16.append(kontainer16)  kontainer16 = ''  y = y + 1    if kontainer16 != '':  kont\_div16.append(kontainer16) |

**Modul Program 4.10** Lanjutan potongan source code proses encode sebelum penyisipan

Pada potongan *source* *code* di atas merupakan bagian *preprocessing* dari proses *encode*. Proses ini terdiri dari mendapat matriks citra dalam bentuk BGR (*Blue, Green, Red*), konversi matriks citra BGR ke RGB (*Red, Green, Blue*), mendapatkan pesan, memberi *stopper* (‘~@&’) pada pesan, dan membagi bagian pesan sesuai kebutuhan pada setiap metode. Proses membaca citra dilakukan dengan fungsi imread() dari *library* OpenCV. Hasil pembacaan tersebut berupa matriks citra BGR yang selanjutnya akan dikonversi dengan fungsi cvtColor() dengan parameter COLOR\_BGR2RGB untuk memudahkan logika pengkodean ketika menuliskan kode aplikasi. Pada proses *preprocess* pesan, sebuah *stopper* ditambahkan pada ujung kalimat sebagai pembatas akhir pesan. Pesan selanjutnya akan ditempatkan pada sebuah variabel untuk dibagi-bagi sesuai kebutuhan pada masing-masing metode yang akan diimplementasikan. Sebelum pesan dibagi, terdapat sebuah proses konversi karakter dari bentuk karakter huruf menjadi sekumpulan bit dengan dasar konversi adalah ASCII. Setelah tahap ini selesai maka pesan akan dilakukan *encode* menggunakan metode LSB, DCT, DWT, dan kombinasi ketiga metode tersebut. Untuk proses *encode* tersebut dapat dilihat pada modul program 4.11 sampai dengan modul program 4.21.

|  |
| --- |
| #LSB  #membagi matriks citra sesuai ruang warna RGB  r1, g1, b1 = cv2.split(coverImage)  p,l = np.shape(r1)  x = 0  kont = ''  # membagi pesan bila panjang pesan melebihi kapasitas cover  if len(kontainer)>(p\*l):  for i in range(len(kontainer)):  kont = kont + kontainer[i]  if (i+1)%(p\*l) == 0:  if x == 0:  kont\_div[x] = kont  else:  kont\_div.append(kont)  kont = ''  x = x + 1  print(x)  if kont != '':  kont\_div.append(kont)  else:  kont\_div[x] = kontainer  x = 0  for i in range(3):  if i== 0:  px1 = r1  elif i == 1:  px1 = g1  elif i == 2:  px1 = b1  if x<len(kont\_div):  # proses sisip  px1 = go\_encodeLSB(px1,kont\_div[i])  if i== 0:  r1 = px1  elif i == 1: |

**Modul Program 4.11** Potongan source code proses encode bagian LSB

|  |
| --- |
| g1 = px1  elif i == 2:  b1 = px1  x = x + 1  else:  break  # postprocess citra stego  img = cv2.merge((b1,g1,r1))  stegoLSB = img  cek = cv2.imwrite(RESULT\_FOLDER + encode\_folder +"/LSB-" + cover\_img, img)  mse,psnr = go\_psnr(cover,stegoLSB)  print(mse, " ", psnr)  dataLSB = {  'file\_name':"LSB-"+cover\_img,  'file\_loc':"/static/result/"+ encode\_folder +"/LSB-" + cover\_img,  'method':'LSB',  'mse':mse,  'psnr':psnr  } |

**Modul Program 4.12** Lanjutan potongan source code proses encode bagian LSB

Pada potongan *source code* di atas merupakan implementasi *encode* dengan metode LSB. Potongan di atas belum termasuk proses penyisipan pada bit *pixel*. Penyisipan akan dijelaskan pada modul program 4.19. Proses *encoding* dimulai dengan memisah ruang warna mencadi tiga bagian serta pembacaan dimensi citra *cover*. Pada potongan *source* *code* *encode* LSB terdapat penambahan *preprocess* pesan untuk membagi pesan bila kapasitas ruang warna pada citra tidak cukup untuk kapasistas pada pesan yang akan disisipkan. Proses *encoding* menggunakan penyisipan urut pada ruang warna *Red, Green,* dan *Blue*. Proses yang dilakukan selanjutnya adalah melakukan penyisipan dengan *method* yang telah dibuat dengan nama go\_encodeLSB() dengan parameter yang dibutuhkan adalah variabel yang berisi salah satu ruang warna dan pesan yang akan disembunyikan. Setelah seluruh pesan berhasil disisipkan maka ketiga ruang warna citra akan kembali dijadikan satu menggunakan fungsi merge() dan dijadikan citra kembali secara utuh dengan fungsi imwrite(). Jika proses pembentukan citra sudah selesai maka citra akan dihitung PSNR-nya dengan cara mengirimkan data matriks citra *cover* dan citra stego ke sebuah *method* penghitung PSNR. *Method* penghitungan tersebut dapat dilihat pada modul program 4.21.

|  |
| --- |
| #DCT  r1, g1, b1 = cv2.split(coverImage)  x,y = np.shape(r1)  z=0  for rgb in range(3):  if rgb == 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1  i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  # mengambil pixel dengan dimensi 8x8  px2 = px1[i:i+8,j:j+8]  if z<len(kont\_div8):  # proses DCT dan kuantisasi  px2 = np.around(cv2.dct(np.float32(px2)), 1)  px2 = np.around(px2/quant).astype(int)  #mengambil 8-bit baru  newpx = [px2[0][3],px2[1][2],px2[2][1],px2[3][0],px2[4][0],px2[3][1],px2[2][2],px2[1][3]]  testpx = np.vstack((newpx,px2[1]))  # proses sisip pesan  testpx = go\_encodeLSB(testpx,kont\_div8[z])    #mengembalikan 8-bit ke tempat asal px2[0][3],px2[1][2],px2[2][1],px2[3][0],px2[4][0],px2[3][1],px2[2][2],px2[1][3] = testpx[0]    #proses IDCT dan kuantisasi  px2 = np.around(cv2.idct(np.float32(px2\*quant))).astype(int)  if rgb== 0:  r1[i:i+8,j:j+8] = px2  elif rgb == 1:  g1[i:i+8,j:j+8] = px2  elif rgb == 2:  b1[i:i+8,j:j+8] = px2  z = z + 1  else:  break  j = j + 8  i = i + 8 |

**Modul Program 4.13** Potongan source code proses encode bagian DCT

|  |
| --- |
| # postprocess citra stego  img = cv2.merge((b1,g1,r1))  stegoDCT = img  cek = cv2.imwrite(RESULT\_FOLDER + encode\_folder +"/DCT-" + cover\_img, img)  mse,psnr = go\_psnr(cover,stegoDCT)  print(mse, " ", psnr)  dataDCT = {  'file\_name':"DCT-"+cover\_img,  'file\_loc':"/static/result/"+ encode\_folder +"/DCT-" + cover\_img,  'method':'DCT',  'mse':mse,  'psnr':psnr  } |

**Modul Program 4.14** Lanjutan potongan source code proses encode bagian DCT

Potongan *source* *code* di atas merupakan proses yang dijalankan setelah *encode* LSB. Proses ini hampir sama denga napa yang telah dilakukkan pada encode LSB, yang membedakannya hanyalah pemilihan koordinat *pixel* yang digunakan. Pada proses encode DCT membagi citra menjadi blok-blok dengan ukuran 8x8 *pixel*. Blok-blok tersebut pada encode DCT akan dilakukan DCT dengan fungsi dct() dari *library* OpenCV sehingga mendapat sebuah koefisien DCT dengan dimensi 8x8. Koefisian tersebut selanjutnya akan dikuantisasikan. Kuantisasi adalah proses membagi koefisien DCT menjadi tiga bagian utama, yaitu frekuensi rendah, frekuensi medium, dan frekuensi tinggi. Pada metode ini bagian/ranah yang digunakan adalah frekuensi medium. Koefisien DCT akan dikalikan dengan matrik kuantisasi secara *element-wise* (per elemen sesuai lokasi) sehingga terbentuk bagian frekuensi-frekuensi tersebut. Ketika proses kuantisasi tersebut telah selesai barulah pesan disisipkan pada 8-bit pertama pada ranah frekuensi medium. Proses penyisipan ini juga menggunakan proses yang sama seperti yang dilakukan pada *encode* LSB, yaitu menggunakan *source* *code* pada modul 4.12. Proses yang dilakukan selanjutnya yaitu mengembalikan ketiga ranah frekuensi menjadi bentuk semula koefisien DCT. Kuantisasi yang dilakukan adalah kebalikan dari yang telah dilakukan sebelum penyisipan, yaitu kuantisasi pembagian. Hasil penyisipan yang berdimensi 8x8 dibagi dengan matriks kuantisasi secara *element-wise*, dilanjutkan dengan prose IDCT dengan fungsi idct() dari OpenCV. Hasil tersebut nantinya akan digunakan untuk menggabungkan blok-blok 8x8 menjadi satu buah matriks ruang warna lagi. Terakhir proses yang dilakukan adalah menghitung PSNR dengan *method* yang sama dengan yang digunakan oleh *encode* LSB yang dapat dilihat pada modul program 4.21.

|  |
| --- |
| #DWT  r1, g1, b1 = cv2.split(coverImage)  x,y = np.shape(r1)  z=0    for rgb in range(3):  if rgb == 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1  i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  # mengambil pixel dengan dimensi 8x8  px3 = px1[i:i+8,j:j+8]  # proses DHWT  cA, (cH, cV, cD) = dwt2(px3, 'haar')  # pengambilan subband LH (cH)  cH = np.around(cH).astype(int)  if z<len(kont\_div16):  # proses sisip pesan  cH = go\_encodeLSB(cH,kont\_div16[z])  # mengembalikkan subband LH  px3 = cA.astype(int),(cH.astype(int), cV.astype(int), cD.astype(int))  # proses IDHWT  px3 = idwt2(px3, 'haar')  if rgb== 0:  r1[i:i+8,j:j+8] = px3  elif rgb == 1:  g1[i:i+8,j:j+8] = px3  elif rgb == 2:  b1[i:i+8,j:j+8] = px3  z = z + 1  else:  break    j = j + 8  i = i + 8  # postprocess citra stego  img = cv2.merge((b1,g1,r1))  stegoDWT = img |

**Modul Program 4.15** Potongan source code proses encode bagian DWT

|  |
| --- |
| cek = cv2.imwrite(RESULT\_FOLDER + encode\_folder +"/DWT-" + cover\_img, img)  mse,psnr = go\_psnr(cover,stegoDWT)  print(mse, " ", psnr)  dataDWT = {  'file\_name':"DWT-"+cover\_img,  'file\_loc':"/static/result/"+ encode\_folder +"/DWT-" + cover\_img,  'method':'DWT',  'mse':mse,  'psnr':psnr  } |

**Modul Program 4.16** Lanjutan potongan source code proses encode bagian DWT

Pada *source* *code* di atas yang membedakan proses *encode*-nya ialah pada proses setelah membagi matriks citra menjadi blok 8x8, proses yang dilakukan adalah DWT dengan fungsi dwt2() yang merupakan salah satu fungsi dari *library* PyWaveletes. Fungsi ini menggunakan parameter bagian blok 8x8 dan tipe ‘*haar*’ yang menandakan metode yang digunakan adalah metode *Discrete Wavelet Transform* dengan tipe Haar. Hasil dari proses ini merupakan empat buah *subband* seperti yang telah dijelaskan pada bab 3 mengenai perancangan proses DWT. Proses selanjutnya akan menggunakan cH untuk disisipi pesan. cH memiliki dimensi 4 kali lebih kecil daripada dimensi blok asli, sehingga daya tampungya hanya 16 bagian yang akan diisi 16-bit bit pesan. Metode penyisipan pesan masih menggunakan *method* yang sama dengan LSB dan DCT, yaitu dengan method yang dapat dilihat pada modul program 4.19. Ketika penyisipan selesai maka *subband* hasil penyisipan akan dikembalikan agar dapat dilakukan IDWT dengan fungsi idwt2() untuk mengembalikan kondisi seperti semula. Proses terakhir yaitu melakukan merge() dan imwrite() seperti yang telah dilakukan pada metode-metode sebelumnya dan menghitung nilai PSNR antara citra *cover* dengan citra stego.

|  |
| --- |
| # KOMBINASI  r1, g1, b1 = cv2.split(coverImage)  x,y = np.shape(r1)  z=0  for rgb in range(3): |

**Modul Program 4.17** Potongan source code proses encode bagian Kombinasi

|  |
| --- |
| if rgb == 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1  i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  # mengambil pixel dengan dimensi 8x8  px3 = px1[i:i+8,j:j+8]  if z<len(kont\_div16):  # proses DHWT  cA, (cH, cV, cD) = dwt2(px3, 'haar')  # proses DCT  cH = np.round(cv2.dct(np.float32(cH))).astype(int)  cH = np.around(cH).astype(int)  # proses sisip pesan  cH = go\_encodeLSB(cH,kont\_div16[z])  px3 = cA.astype(int),(cH.astype(int), cV.astype(int), cD.astype(int))  # proses IDWT  px3 = idwt2(px3, 'haar')  if rgb== 0:  r1[i:i+8,j:j+8] = px3  elif rgb == 1:  g1[i:i+8,j:j+8] = px3  elif rgb == 2:  b1[i:i+8,j:j+8] = px3  z = z + 1  else:  break    j = j + 8  i = i + 8  # posprocessing citra stego  img = cv2.merge((b1,g1,r1))  stegoALL = img  cek = cv2.imwrite(RESULT\_FOLDER + encode\_folder +"/ALL-" + cover\_img, img)  mse,psnr = go\_psnr(cover,stegoALL)  print(mse, " ", psnr)  dataALL = {  'file\_name':"ALL-"+cover\_img,  'file\_loc':"/static/result/"+ encode\_folder +"/ALL-" + cover\_img,  'method':'ALL',  'mse':mse,  'psnr':psnr  } |

**Modul Program 4.18** Lanjutan potongan source code proses encode bagian Kombinasi

Pada proses *encode* dengan metode kombinasi LSB, DCT, dan DWT *source* *code* yang digunakan tidak jauh berbeda dengan metode lain. Pada proses *encode* ini membagi blok-blok 8x8 yang akan dilakukan DWT denan fungsi dwt2() terlebih dahulu baru dilakukan DCT dengan fungsi dct(). Pada proses *encode* ini terdapat perbedaan ketika melakukan DCT yaitu tidak ada proses kuantisasi karena dimensi matrik kuantisasi dengan dimensi *subband* berbeda. Sehingga setelah dilakukan DCT blok hasil DWT-DCT dilakukan penyisipan 16-bit bagian pesan. Setelah pesan disisipkan proses selanjutnya juga agak berbeda dengan DCT biasa, yaitu tidak dilakukan IDCT. Hal tersebut dikarenakan pada *pixel* yang disisipi nilainya akan berubah secara signifikan ketika melakukan penghitungan pembulatan hasil IDCT() sehingga tahap IDCT pada proses *encode* kombinasi ditiadakan. Hal yang selanjutnya setelah proses penyisipan dilakukan adalah mengembalikan subband ke asal dan melakukan IDWT dengan fungsi idwt2(). Hasil dari proses tersebut selanjutnya akan digabungkan seperti pada metode-metode yang telah dilakukan sebelumnya untuk dilakukan proses merge dengan fungsi merge() hingga menjadi satu citra utuh. Tahap terakhir untuk proses *encode* ini yaitu penghitungan PSNR dengan *method* pada modul program 4.21.

Keempat proses *encode* di atas pada dasarnya mempunyai persamaan yaitu disisipkan dari ruang warna merah, jika pesan belum habis maka penyisipan dilanjutkan pada ruang warna hijau, dan terkahir pada ruang warna biru jika pesan belum habis. Untuk proses penyisipan pada ruang warna dapat dilihat pada modul program 4.19.

|  |
| --- |
| def go\_encodeLSB(px,kontainer):  hit\_kontainer = len(kontainer)  height,width = np.shape(px)  index = 0;row\_stop = 0  for row in range(height):  for col in range(width):  if index<hit\_kontainer:  bit\_temp = f'{px[row][col]:08b}'  bit\_lsb = list(bit\_temp)  bit\_lsb[-1] = kontainer[index]  bit\_temp = ''.join(bit\_lsb)  px[row][col] = int(bit\_temp,2) |

**Modul Program 4.19** Potongan source code berupa method penyisipan pada pixel

|  |
| --- |
| index = index + 1  else:  row\_stop = 1;break  if row\_stop != 0:  break  return px |

**Modul Program 4.20** Lanjutan potongan source code berupa method penyisipan pada pixel

Pada *method* di atas terdapat proses penyisipan yang berbasis pada penyisipan dengan mengubah bit terakhir pada bit-bit yang ada. Pesan yang dimasukkan akan berupa bit-bit biner yang terdapat pada variabel kontainer dan matriks ruang warna yang terdapat pada variabel px. Penyisipan akan berjalan urut sesuai indeks matriks pada variabel px dan akan berhenti ketika pesan habis atau bit pesan terakhir telah disisipkan pada indeks matriks penampung terakhir yang ada. *Method* tersebut akan mengembalikan nilai berupa matriks variabel px yang telah disisipi pesan.

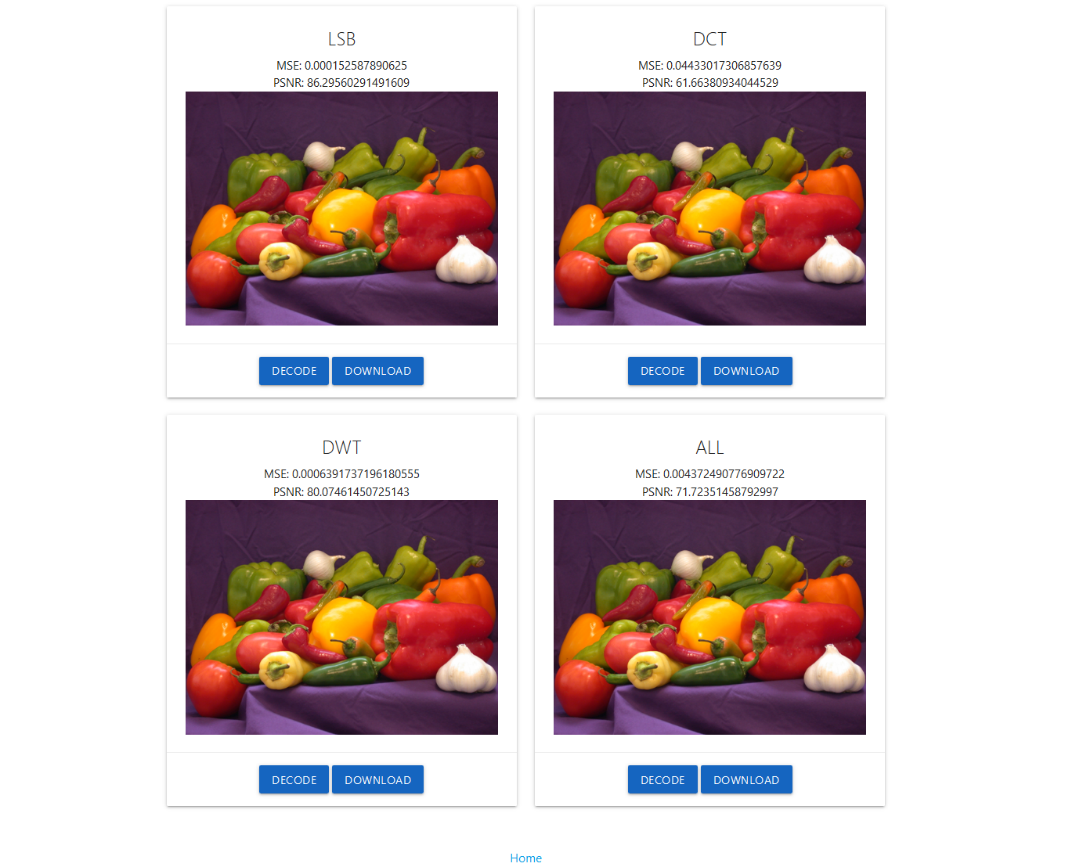
Selain *method* untuk proses penyisipan, terdapat *method* untuk menghitung nilai PSNR dan MSE pada proses *encode*. *Method* ini dapat dilihat pada modul program 4.21.

|  |
| --- |
| def go\_psnr(cover\_img,stego\_img):  dimensiCover = np.shape(cover\_img)  dimensiStego = np.shape(stego\_img)  if (dimensiCover[0]\*dimensiCover[1]) != (dimensiStego[0]\*dimensiStego[1]):  mse = (dimensiCover[0]\*dimensiCover[1]) - (dimensiStego[0]\*dimensiStego[1])  # psnr = "Beda Dimensi"  else:  mse = np.mean((cover\_img - stego\_img) \*\* 2 )  if mse == 0:  psnr = 100  else:  PIXEL\_MAX = 255.0  psnr = 20 \* math.log10(PIXEL\_MAX / math.sqrt(mse))  return mse,psnr |

**Modul Program 4.21** Method penghitungan PSNR dan MSE

*Method* di atas akan menerima dua buah masukkan. Kedua masukkan tersebut berasal dari parameter yang dikirimkan. Parameter pertama adalah matriks citra *cover* dan parameter kedua adalah matriks citra *stego*. Method ini mengacu pada persamaan (2.9) dan persamaan (2.10). Method tersebut juga terdapat *source* *code* logika tambahan yang mengacu pada persamaan (2.10) yang merupakan persamaan penghitungan MSE, *source* *code* ini akan menghitung tingkat perbedaan citra bila terdapat perbedaan dimensi. Hasil akhir dari *method* ini adalah nilai MSE dan PSNR yang akan dikirimkan kembali ke pemanggil *method*. Nilai-nilai tersebut nantinya akan ditampilkan ketika seluruh proses penyisipan telah selesai.

Proses *encode* telah selesai jika semua proses penyisipan dan penghitungan telah dilakukan. Hal yang dilakukan oleh aplikasi selanjutnya adalah menampilkan hasil penyisipan pada halaman hasil encode. Pada halaman ini akan ditampilkan keempat hasil penyisipan dengan masing-masing metode yang ada. Tampilan untuk halaman hasil encode dapat dilihat pada gambar 4.5.



**Gambar 4.5** Tampilan halaman hasil encode

Tampilan pada halaman hasil *encode* terdapat empat buah *cards* yang masing-masing berisi metode yang digunakan, PSNR dan MSE, citra stego, tombol *download*, dan tombol *decode*. *Source* *code* untuk tampilan halaman hasil *encode* dapat dilihat pada modul program 4.22 sampai dengan modul program 4.23.

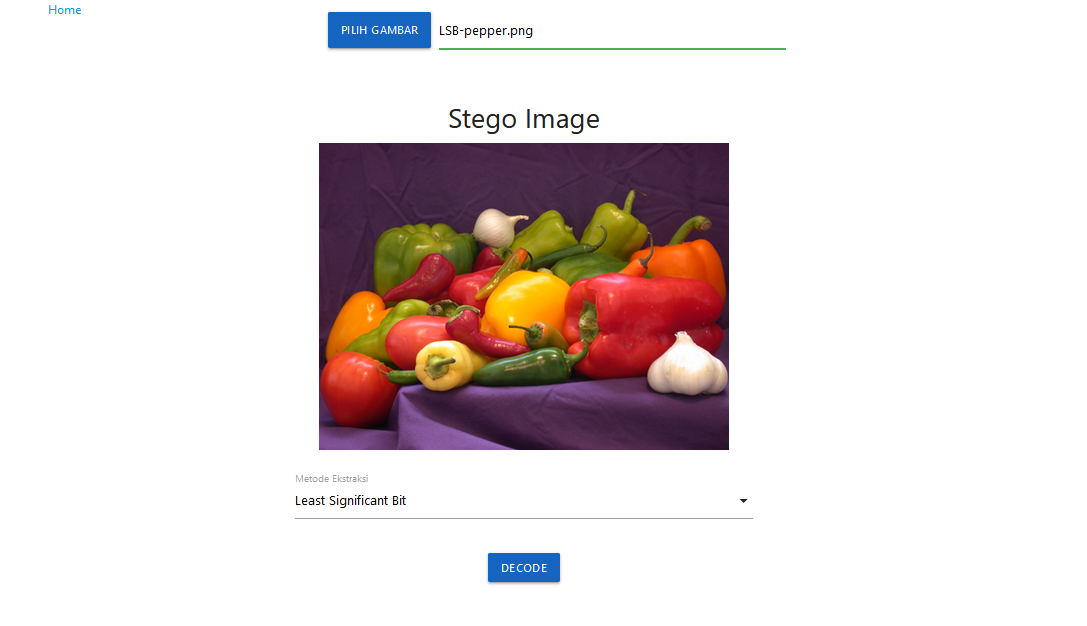
|  |
| --- |
| {% extends 'base.html' %}  {% block head %}  <title>STEGANOGRAFI - Hasil Encode</title>  {% endblock %}  {% block body %}  <div class="row">  <div class="row">  {% for result in result %}  <div class="col s12 m6">  <div class="card large">  <div class="card-content">  <span class="card-title">{{ result.method }}</span>  MSE: {{ result.mse }} <br>  PSNR: {{ result.psnr }}  <img src="{{ result.file\_loc }}" class="responsive-img" style="max-height: 350px;">  </div>  <div class="card-action">  <input type="hidden" value="{{ result.file\_loc }}" name="fileLoc" id="fileLocation{{ result.method }}">  <a class="waves-effect waves-light btn modal-trigger blue darken-3" href="#modal1" id="decodeHasil{{ result.method }}">DECODE</a>  <a href="/return-files/{{ result.file\_name }}" class="btn blue darken-3">DOWNLOAD</a>  </div>  </div>  </div>  {% endfor %}  <div id="modal1" class="modal">  <div class="modal-content">  <h4>Hasil Decode</h4>  <p id="hidden\_object">bla bla kudune plain text po hidden image</p>  </div>  <div class="modal-footer">  <a href="#!" class="modal-close waves-effect waves-green btn-flat">Tutup</a>  </div>  </div>  </div>  </div>  <div style="float: center;"><a href="/">Home</a></div>  {% endblock %} |

**Modul Program 4.22** Source code tampilan hasil encode

Pada *source code* halaman *encode* masih menggunakan teknik templating jinja sehingga untuk mengeluarkan hasil proses *encoding* dapat dilakukan dengan menggunakan perintah {% for [variabel\_akses] in [variabel\_hasilencode] %} dan ditutup dengan {% endfor %}, sedangkan dalam perulangan terdapat komponen *card* dan isi-isinya yang merupakan hasil dari proses *encoding* seperti MSE, PSNR, dan citra stego ditambah dengan dua button yaitu tombol *decode* dan tombol *download*. Tombol decode akan melakukan fungsi ekstraksi pesan dari aplikasi sedangakan tombol *download* akan mengunduh citra stego. fungsi decode atau ekstraksi pesan akan dijelaskan pada sub bab tahap *decode*.

### Halaman Decode

Setelah proses penyisipan pesan maka proses selanjutnya adalah proses ekstraksi pesan dari citra stego yang disebut proses decode. Proses *decode* akan mengeluarkan isi pesan tersembunyi. Proses *decode* mempunyai dua cara akses yaitu dengan menekan tombol *decode* halaman hasil *encode* atau dengan menuju halaman *decode* untuk melakukan tahapan *decode* pada halaman tersebut. Tampilan halaman *decode* dapat dilihat pada gambar 4.6.



**Gambar 4.6** Tampilan halaman decode

*Source code* tampilan halaman *decode* dapat dilihat pada modul program 4.23 sampai dengan modul program 4.33.

|  |
| --- |
| @app.route('/decode')  def show\_decode():  return render\_template('hal\_decode.html') |

**Modul Program 4.23** Source code routing menuju halaman decode

|  |
| --- |
| {% extends 'base.html' %}  {% block head %}  <title>STEGANOGRAFI - Decode</title>  {% endblock %}  {% block body %}  <div style="float: left;"><a href="/">Home</a></div>  <form action="/go\_decode" method="POST" enctype=multipart/form-data>  <div class="row center-align">  <div class="col l3"></div>  <div class="col s12 l6">  <div class="file-field input-field">  <div class="btn blue darken-3">  <span>Pilih Gambar</span>  <input type="file" id="image-up" name="stego\_image" accept="image/\*" onchange="original()">  </div>  <div class="file-path-wrapper">  <input class="file-path validate" type="text" placeholder="Pilih Stego Image">  </div>  </div>  </div>  </div>  <div class="row center-align">  <div class="col l3"></div>  <div class="col center-align s12 l6">  <h4>Stego Image</h4>  <img id="ori-image" class="responsive-img">  </div>  </div>  <div class="row center-align">  <div class="col l3"></div>  <div class="input-field col center-align s12 l6">  <select id="select-metode">  <optgroup label="Ranah Spatial">  <option value="LSB">Least Significant Bit</option>  </optgroup>  <optgroup label="Ranah Frekuensi">  <option value="DCT">Discrete Cosine Transform</option>  <option value="DWT">Discrete Haar Wavelete Transform</option>  </optgroup>  <optgroup label="Kombinasi">  <option value="ALL">Kombinasi LSB+DCT+DWT</option>  </optgroup>  </select>  <label>Metode Ekstraksi</label>  </div>  </div>  <!-- Modal Trigger -->  <a class="waves-effect waves-light btn modal-trigger blue darken-3" href="#modal1" id="decode\_object">DECODE</a> |

**Modul Program 4.24** Source code tampilan halaman decode

|  |
| --- |
| <!-- Modal Structure -->  <div id="modal1" class="modal">  <div class="modal-content">  <h4>Hasil Decode</h4>  <p id="hidden\_object">bla bla kudune plain text po hidden image</p>  </div>  <div class="modal-footer">  <a href="#!" class="modal-close waves-effect waves-green btn-flat">Tutup</a>  </div>  </div>  </form>  {% endblock %} |

**Modul Program 4.25** Lanjutan source code tampilan decode

Proses *decode* dapat dilihat pada modul program 4.15 sampai dengan modul program 4.33.

|  |
| --- |
| @app.route('/go\_decode', methods=['GET', 'POST'])  def go\_decode():  # print(request.files)  if request.method == 'POST' and request.form['type'] == 'loc':  print("mlebu sek location")  stego\_loc = request.form['location']  print(stego\_loc)  stego\_method = request.form['method']  # new\_stego\_img = go\_upload(stego\_img)  hidden\_object = process\_decode(stego\_loc,stego\_method,2)  if hidden\_object != '':  return hidden\_object  else:  return "Tidak Ditemukan"  elif request.method == 'POST' and request.form['type'] == 'file':  print("mlebu sek file")  stego\_img = request.files['file']  stego\_method = request.form['method']  print(stego\_img)  makeID = random.randint(1,100)  new\_stego\_img = go\_upload(stego\_img,makeID)  hidden\_object = process\_decode(new\_stego\_img,stego\_method,1)  if hidden\_object != '':  return hidden\_object  else:  return "Tidak Ditemukan"  else:  return "salah POST" |

**Modul Program 4.26** Source code routing menuju method proses decode

|  |
| --- |
| def process\_decode(stego\_img,stego\_method,mode):  if mode == 1:  stegoImage = cv2.imread(UPLOAD\_FOLDER + "/" + stego\_img,1)  else:  stegoImage = cv2.imread(os.getcwd() + stego\_img,1)  stegoImage = cv2.cvtColor(stegoImage, cv2.COLOR\_BGR2RGB) |

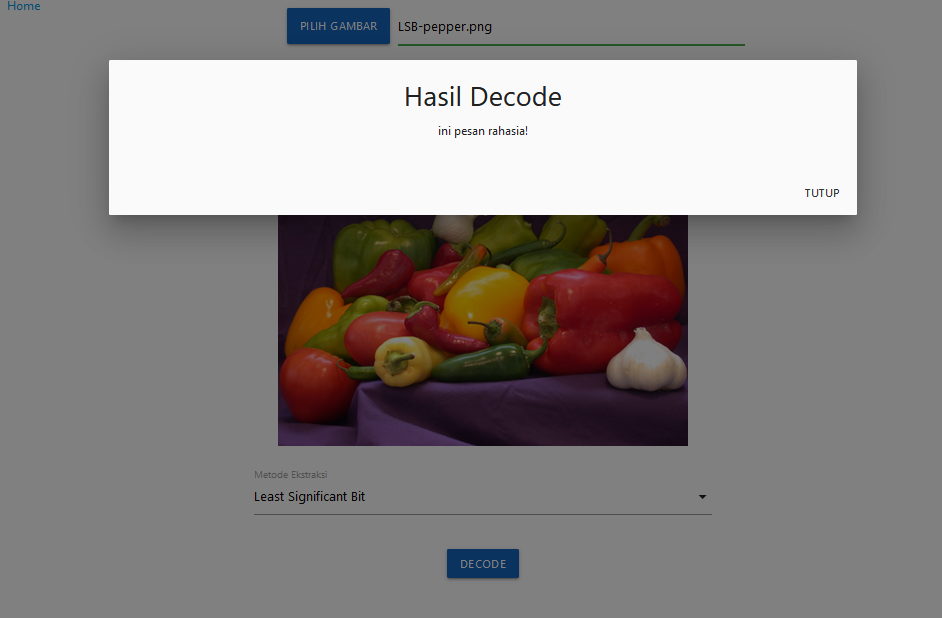
**Modul Program 4.27** Potongan source code proses decode sebelum ekstraksi

Pada *source* *code* di atas terdapat beberapa fungsi yang juga digunakan pada proses sebelum memulai penyisipan pesan pada *encode*. Fungsi-fungsi tersebut antara lain fungsi imread() yang akan membaca citra input dan cvtColor() yang akan mengubah format matriks BGR menjadi RGB. Pada *source code* method di atas terdapat dua parameter yaitu citra stego dan metode steganografi, serta satu parameter tambahan yaitu mode. Citra stego adalah citra berisi pesan tersembunyi, metode steganografi adalah pilihan metode ekstraksi yang dimasukkan oleh pengguna, dan mode adalah format pengambilan citra berdasarkan direktori dikarenakan terdapat direktori sementara hasil proses *encode* pada halaman hasil *encode* dan direktori dari gawai pengguna ketika menggunakan halaman *decode*. Proses yang dilakukan selanjutnya adalah mengambil nilai stego\_method yang berisi metode steganografi untuk dilakukan ekstraksi pesan sesuai dengan pilihan metode yang dimasukkan.

|  |
| --- |
| if stego\_method == 'LSB':  r1, g1, b1 = cv2.split(stegoImage)  x = 0  kontainer = ''  for rgb in range(3):  if rgb== 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1  if '~@&' not in kontainer:  kontainer = kontainer + go\_decodeLSB(px1)  else:  print("Log LSB: Stop Decode")  break    for i in range(len(kontainer)):  if kontainer[i] == '~' and kontainer[i+1] == '@' and kontainer[i+2] == '&':  kontainer = kontainer[0:i]  break |

**Modul Program 4.28** Potongan source code proses decode bagian LSB

Potongan kode di atas merupakan proses *decode* dengan metode LSB. Hal yang pertama dilakukan yaitu memecah matriks RGB dari citra dan selanjutnya dilakukan *looping* ekstraksi. *Looping* akan berisi ekstraksi dengan pemanggilan *method* go\_decodeLSB() yang dapat dilihat pada modul program 4.33 yang akan melakukan ekstraksi dengan mengekstrak bit-bit pada ruang warna dan mencari ‘~@&’ sebagai *stopper*. Jika *stopper* tidak ditemukan pada kontainer (penampung hasil ekstraksi) maka pencarian akan terus dilakukan. Namun apabila *stopper* terdapat pada kontainer atau batas looping terlampau maka *looping* akan melakukan break atau berhenti. Kontainer yang telah berisi hasil ekstraksi akan diperiksa kembali untuk memastikan hasil ekstraksi tidak melebihi *stopper* yang ditemukan. Ketika proses ekstraksi selesai hasil dari kontainer akan dikirim kembali ke pemanggil untuk bisa ditampilkan seperti pada gambar 4.7.



**Gambar 4.7** Tampilan Pop-up Modal yang berisikan hasil ekstraksi pesan

|  |
| --- |
| elif stego\_method == 'DCT':  r1, g1, b1 = cv2.split(stegoImage)  x,y = np.shape(r1)  z=0  i=0  stop\_all = 0  kontainer = ''  for rgb in range(3):  print(rgb)  if rgb== 0:  px1 = r1 |

**Modul Program 4.29** Potongan source code proses decode bagian DCT

|  |
| --- |
| elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1    i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  px2 = px1[i:i+8,j:j+8]  px2 = np.around(cv2.dct(np.float32(px2)), 1)  px2 = np.around(px2/quant).astype(int)  newpx = [px2[0][3] ,px2[1][2] ,px2[2][1] ,px2[3][0] ,px2[4][0] ,px2[3][1] ,px2[2][2] ,px2[1][3]]  testpx = np.vstack((newpx,px2[1]))  if '~@&' not in kontainer:  kontainer = kontainer + go\_decodeLSB(testpx)  kontainer = kontainer[0:-1]  z = z + 1  else:  print("Log DCT: Stop Decode")  stop\_all = 1  break  j = j + 8  if stop\_all != 0:  break  else:  i = i + 8  if stop\_all != 0:  break  for i in range(len(kontainer)):  if kontainer[i] == '~' and kontainer[i+1] == '@' and kontainer[i+2] == '&':  kontainer = kontainer[0:i]  break |

**Modul Program 4.30** Lanjutan potongan source code proses decode bagian DCT

Potongan *source code* di atas merupakan bagian proses *decode* apabila pengguna memilih metode ekstraksi pesan menggunakan metode DCT. Proses dimulai dengan instansiasi beberapa variabel dan ruang warna yang digunakan, selanjutnya dilakukan ektraksi dengan cara membagi salah satu ruang warna citra menjadi blok 8x8. Bagian-bagian tersebut akan dilakukan proses transformasi DCT dan kuantisasi terlebih dahulu seperti pada proses *encode* dan selanjutnya akan dilakukan proses ekstraksi, yaitu mengambil 8 bagian dari frekuensi medium dan akan dikirimkan menuju *method* go\_decodeLSB(). Proses ini hampir sama dengan apa yang telah dilakukan pada proses *decode* LSB, hanya saja bagian yang dikirimkan ke *method* go\_decodeLSB() hanya list berisi 8 element. Setelah proses pada go\_decodeLSB() selesai maka pesan yang berhasil diekstrak akan dikirimkan kembali menuju pemanggil untuk ditampilkan pada halaman decode.

|  |
| --- |
| elif stego\_method == 'DWT':  r1, g1, b1 = cv2.split(stegoImage)  x,y = np.shape(r1)  z=0    stop\_all = 0  kontainer = ''  for rgb in range(3):  if rgb== 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1  i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  px3 = px1[i:i+8,j:j+8]  cA, (cH, cV, cD) = dwt2(px3, 'haar')  cH = np.around(cH).astype(int)  if '~@&' not in kontainer:  kontainer = kontainer + go\_decodeLSB(cH)  # print(kontainer)  z = z + 1  else:  print("Log DWT: Stop Decode")  stop\_all = 1  break  j = j + 8  if stop\_all != 0:  break  else:  i = i + 8  if stop\_all != 0:  break  for i in range(len(kontainer)):  if kontainer[i] == '~' and kontainer[i+1] == '@' and kontainer[i+2] == '&':  kontainer = kontainer[0:i]  break |

**Modul Program 4.31** Potongan source code proses decode bagian DWT

Potongan *source code* di atas merupakan bagian proses *decode* apabila pengguna memilih metode ekstraksi pesan menggunakan metode DWT. Secara proses yang dilakukan hampir sama dengan apa yang dilakukan pada decode DCT, hanya saja setelah ruang warna dibagi menjadi blok 8x8, per blok akan dilakukan transformasi DWT dan selanjutnya bagian yang diambil untuk dikirim menuju *method* go\_decodeLSB() merupakan *subband* LH yang mempunyai 16 elemen matriks. Pesan yang telah berhasil diekstrak dari *method* go\_decodeLSB() akan dikirm kembali untuk ditampilkan pada halaman *decode*.

|  |
| --- |
| elif stego\_method == 'ALL':  r1, g1, b1 = cv2.split(stegoImage)  x,y = np.shape(r1)  z=0    stop\_all = 0  kontainer = ''  for rgb in range(3):  if rgb== 0:  px1 = r1  elif rgb == 1:  px1 = g1  elif rgb == 2:  px1 = b1    i=0  while i<(int(x/8))\*8:  j=0  while j<(int(y/8))\*8:  px3 = px1[i:i+8,j:j+8]  cA, (cH, cV, cD) = dwt2(px3, 'haar')  # cD = np.round(cv2.dct(np.float32(cD))).astype(int)  cH = np.around(cH).astype(int)  if '~@&' not in kontainer:  kontainer = kontainer + go\_decodeLSB(cH)  # print(kontainer)  z = z + 1  else:  print("Log ALL: Stop Decode")  stop\_all = 1  break  j = j + 8  if stop\_all != 0:  break  else:  i = i + 8  if stop\_all != 0:  break  for i in range(len(kontainer)):  if kontainer[i] == '~' and kontainer[i+1] == '@' and kontainer[i+2] == '&':  kontainer = kontainer[0:i]  break |

**Modul Program 4.32** Potongan source code proses decode bagian metode kombinasi

Potongan *source code* di atas merupakan bagian proses *decode* apabila pengguna memilih metode ekstraksi pesan menggunakan metode ALL, yaitu kombinasi metode LSB, DCT, dan DWT. Secara proses yang dilakukan hampir sama dengan apa yang dilakukan pada *decode* DCT dan DWT, hanya saja setelah ruang warna dibagi menjadi blok 8x8, per blok akan dilakukan transformasi DWT dan selanjutnya bagian yang diambil untuk dikirim menuju *method* go\_decodeLSB() merupakan *subband* LH yang telah dilakukan transformasi DCT. Pesan yang telah berhasil diekstrak dari method go\_decodeLSB() akan dikirm kembali untuk ditampilkan pada halaman *decode*.

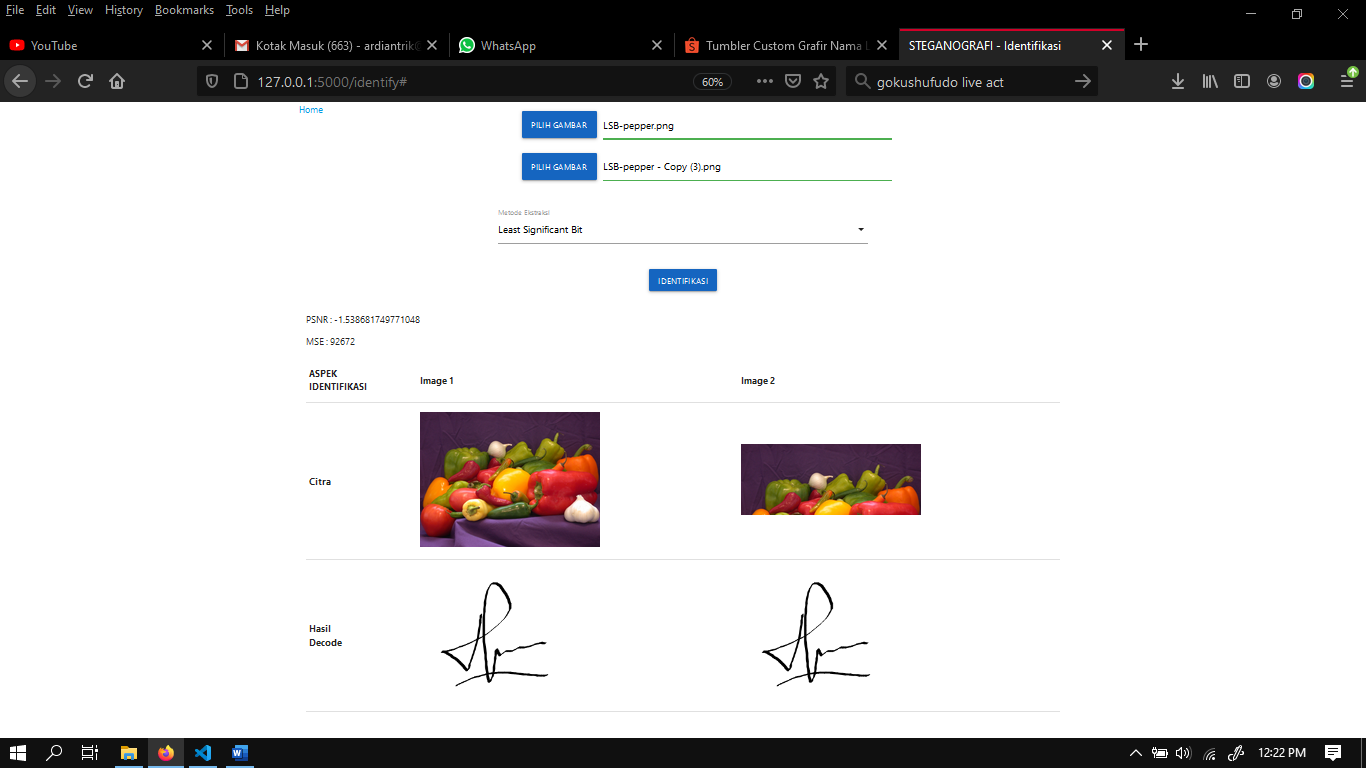
|  |
| --- |
| def go\_decodeLSB(px):  kontainer = ''  stop\_kontainer = ''  height,width = np.shape(px)  index = 0  row\_stop = 0  stop\_status = 0  for row in range(height):  for col in range(width):  if stop\_status == 0:  bit\_temp = f'{px[row][col]:08b}'  kontainer = kontainer + bit\_temp[-1]  index = index + 1  if index%8 == 0:  stop\_kontainer=stop\_kontainer + chr(int(kontainer,2))  if '~@&' in stop\_kontainer:  stop\_status = 999  row\_stop = 1  print("STOP skuy")  break  else:  kontainer = ''  else:  row\_stop = 1  break  if row\_stop != 0:  break  # print(px)  decodeLSB\_result = stop\_kontainer  return decodeLSB\_result |

**Modul Program 4.33** Potongan source code berupa method ektraksi pesan pada pixel

*Source code* di atas merupakan proses ektraksi yang digunakan setiap metode untuk melakukan ekstraksi pesan dari *pixel*. Proses ekstaksi dimulai dengan instansiasi beberapa variabel yang akan digunakan dan selanjutnya akan memulai *looping* ekstraksi. Proses yang berada di dalam *looping* terdapat proses konversi bilangan *pixel* menjadi bit-bit biner dan pengambilan bit terakhir pada biner *pixel*. Bit-bit yang sudah diambil akan disimpan dalam satu kontainer untuk disatukan menjadi pesan tersembunyi. Proses selanjutnya yaitu pengecekan kontainer setelah melakukan pemgambilan bit terakhir biner, apabila ditemukan tanda *stopper* maka pencarian akan dihentikan dan pesan akan dikirimkan kembali menuju *method* proses\_decode().

### Halaman Identifikasi

Halaman identifikasi merupakan halaman untuk melakukan proses identifikasi pada citra stego asli dan citra stego yang telah dimanipulasi. Fungsi dari identifikasi adalah mengetahui kodisi pesan tersembunyi ketika diekstrak apakah pesan tersebut mengalami perubahan atau tidak ketika citra stego dimanipulasi. Perbandingan kedua citra serta hasilnya akan ditampilkan bersamaan untuk dapat dibandingkan secara langsung. Halaman tersebut juga menampilkan nilai PSNR dari kedua citra yang dimasukkan untuk membuktikan bahwa kedua citra tersebut berbeda atau telah mengalami manipulasi/serangan. Halaman identifikasi dapat dilihat pada gambar 4.8.



**Gambar 4.8** Tampilan pada halaman identifikasi

*Source code* tampilan halaman identifikasi dapat dilihat pada modul program 4.34 sampai dengan modul program 4.36.

|  |
| --- |
| @app.route('/identify')  def show\_identify():  return render\_template('hal\_identify.html') |

**Modul Program 4.34** Source code routing menuju halaman identifikasi

|  |
| --- |
| {% extends 'base.html' %}  {% block head %}  <title>STEGANOGRAFI - Identifikasi</title>  {% endblock %}  {% block body %}  <div style="float: left;"><a href="/">Home</a></div>  <form action="/go\_decode" method="POST" enctype=multipart/form-data>  <div class="row center-align">  <div class="col l3"></div>  <div class="col s12 l6">  <div class="file-field input-field">  <div class="btn blue darken-3">  <span>Pilih Gambar</span>  <input type="file" id="image-up1" name="image" accept="image/\*">  </div>  <div class="file-path-wrapper">  <input class="file-path validate" type="text" placeholder="Pilih Citra 1">  </div>  </div>  <div class="file-field input-field">  <div class="btn blue darken-3">  <span>Pilih Gambar</span>  <input type="file" id="image-up2" name="image" accept="image/\*">  </div>  <div class="file-path-wrapper">  <input class="file-path validate" type="text" placeholder="Pilih CItra 2">  </div>  </div>  </div>  </div>  <div class="row center-align">  <div class="col l3"></div>  <div class="input-field col center-align s12 l6">  <select id="select-metode">  <optgroup label="Ranah Spatial">  <option value="LSB">Least Significant Bit</option>  </optgroup>  <optgroup label="Ranah Frekuensi">  <option value="DCT">Discrete Cosine Transform</option> |

**Modul Program 4.35** Source code tampilan halaman identifikasi

|  |
| --- |
| <option value="DWT">Discrete Haar Wavelete Transform</option>  </optgroup>  <optgroup label="Kombinasi">  <option value="ALL">Kombinasi LSB+DCT+DWT</option>  </optgroup>  </select>  <label>Metode Ekstraksi</label>  </div>  </div>  <div class="row">  <div class="col l2"></div>  <div class="col s12 l8">  <a href="#" class="btn blue darken-3" id="identifyHasil">Identifikasi</a>  </div>  </div>  </form>  <div class="row">  <div class="col 13">  <div style="float: left; text-align: left;">  <p id="idnPSNR">PSNR : 0</p>  <p id="idnMSE">MSE : 0</p>  </div>  </div>  <div class="col s12 l12">  <table class="responsive-table">  <thead>  <th>ASPEK <br> IDENTIFIKASI</th>  <th>Image 1</th>  <th>Image 2</th>  </thead>  <tbody>  <tr>  <th>Citra</th>  <td><img id="stegImage1" class="responsive-img" width="300px"></td>  <td><img id="stegImage2" class="responsive-img" width="300px"></td>  </tr>  <tr>  <th>Hasil <br> Decode</th>  <div>  <td><p id="hidObject1" style="width: 300px;overflow: hidden;text-overflow: ellipsis;white-space: nowrap;">plain text po hidden image 1</p></td>  <td><p id="hidObject2" style="width: 300px;overflow: hidden;text-overflow: ellipsis;white-space: nowrap;">plain text po hidden image 2</p></td>  </div>  </tr>  </tbody>  </table>  </div>  </div>  {% endblock %} |

**Modul Program 4.36** Lanjutan source code tampilan halaman identifikasi

Proses yang terdapat pada halaman identifikasi sepenuhnya mengambil method-method yang dilakukan pada proses penghitungan PSNR pada proses *encode* dan proses ekstraksi pesan pada proses *decode*. Proses penghitungan PSNR telah dijelaskan pada sub bab 4.1.1 dan *source code* penghitungan dapat dilihat pada modul program 4.2, sedangkan proses ekstraksi telah dijelaskan pada sub bab 4.1.2 dan *source code* method ektraksi pesan dapat dilihat pada modul program 4.26 sampai dengan modul program 4.33. Proses penghitungan PSNR digunakan untuk menghitung tingkat perbedaan antara citra stego asli dengan citra stego yang telah diserang. Jika nilai PSNR tidak menunjukan nilai 100 maka kedua citra tersebut sudah bukan citra yang sama. Proses *decode* digunakan untuk ektraksi pesan dari kedua citra yang dimasukkan sehingga dapat dibandingkan apakah pesan tersembunyi pada citra stego yang telah dimani pulasi telah berubah atau tidak.

## Pengujian

Tahapan pengujian pada penelitian ini menggunakan metode pendekatan steganalysis StirMark Benchmark untuk uji ketahanan pesan (Robustness) dan uji PSNR, visual, dan perbandingan RGB pada citra cover dengan citra stego untuk uji ketidaknampakan (Fidelity). Hasil pengujian akan dicatat dalam tabel pengamatan seperti yang sudah dijelaskan dalam perancangan pengujian pada bab 3.

### Pengujian Robustness

### Pengujian Fidelity

#Belumrampung #lakukanpengujian #takonpengujianeminimalnangbab4piro

## Pembahasan

asdasdas

# BAB V PENUTUP

## Kesimpulan

Berdasarkan pada penelitian yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut:

1. Kesimpulan 1
2. Kesimpulan 2

Selain itu belum tau

## Saran

Sarannya untuk penelititan kedepan sebagai berikut:

1. Saran 1
2. Saran 2

asdasdasdasd

asdasdasdasd

# DAFTAR PUSTAKA

# LAMPIRAN